The R Journal

Volume 10/2, December 2018

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

Contents

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

Editorial

by John Verzani

On behalf of the editorial board, I am pleased to present Volume 10, Issue 2 of the R Journal.

This issue covers a wide range of topics through its 37 articles. As is typical, many of these are related to packages that provide tools for new statistical modeling in R. Examples in this issue include "**clustMixType**: User-Friendly Clustering of Mixed-Type Data in R" by Szepannek and "**BNSP**: an R Package for Fitting Bayesian Semiparametric Regression Models and Variable Selection" by Papageorgiou.

Several contributions highlight new packages that enhance and extend existing modeling areas. Examples of this are "Forecast Combinations in R using the **ForecastComb** Package" by Weiss, Raviv, and Roetzer; "**testforDEP**: An R Package for Modern Distribution-free Tests and Visualization Tools for Independence" by Miecznikowski, Hsu, Chen, and Vexler; "**NetworkToolbox**: Methods and Measures for Brain, Cognitive, and Psychometric Network Analysis in R" by Christensen; and "**bnclassify**: Learning Bayesian Network Classifiers" by Mihaljevic, Bielza, and Larrañaga".

Some of the contributions include easier interfaces to modeling, such as "SARIMA Analysis and Automated Model Reports with **BETS**, an R Package" by Speranza, Ferreira, and da Costa and "**ShinyItemAnalysis** for Teaching Psychometrics and to Enforce Routine Analysis of Educational Tests" by Martinková and Drabinová.

Other modeling topics are estimation, as in "**SMM**: An R Package for Estimation and Simulation of Discrete-time semi-Markov Models" by Vergne, Barbu, Bérard, Cellier, and Sautreuil; missing data, as in "Profile Likelihood Estimation of the Correlation Coefficient in the Presence of Left, Right or Interval Censoring and Missing Data" by Li, Gillespie, Shedden, and Gillespie; and large data, as with "Basis-Adaptive Selection Algorithm in **dr**-package" by Yoo.

Included in this volume are two submissions extending R's visualization capabilities: "ggplot2 Compatible Quantile-Quantile Plots in R" by Loy, Almeida, and Hofmann and "Geospatial Point Density" by Evangelista and Beskow.

Several new tools are described in the articles. New takes on some programming idioms are given in "Dot-Pipe: an S3 Extensible Pipe for R" by Mount and Zumel; some libraries from other languages have been ported over, such as described in "**sdpt3r**: Semidefinite Quadratic Linear Programming in R" by Rahman; and interfaces to other languages are detailed in "**jsr223**: A Java Platform Integration for R with Programming Languages Groovy, JavaScript, JRuby, Jython, and Kotlin" by Gilbert and Dahl and "**RcppMsgPack**: MsgPack Headers and Interface Functions for R" by Eddelbuettel and Ching.

As usual, we have a few application areas represented in this issue, for example "**stplanr**: A Package for Transport Planning" by Lovelace and Ellison.

Finally, as much as the R Journal helps put some sense of order to R's ecosystem, it can't possibly cover the tremendous growth (CRAN grew to 13,592 packages by year's end; Bioconductor reports 1649 software packages). Silge, Nash, and Graves provide some insight in "Navigating the R Package Universe."

There are notes from several conferences included. These highlight R's expanded reach into Latin America and Poland in addition to its reach into the pharmaceutical and medical professions. The are also the usual reports about changes to CRAN, Bioconductor, and R; and a report from the R Foundation.

This year, Di Cook joins the Editorial Board, as Roger Bivand rotates off. I extend my thanks to Roger for his many contributions. His efforts now allow us to add digital object identifiers (DOIs) for R Journal articles. I am very excited about all that Di will bring. She is already off and running, securing funding from the R Foundation that will allow the journal to have some paid assistance. I also wish to acknowledge the hard work of Norm Matloff and Olivia Lau. The R Journal's popularity has grown–the two volumes of 2018 are over 1100 printed pages–and these two were instrumental in managing the growth.

Finally, I would like to thank the many reviewers that made this edition possible. In my view, one of the biggest contributions of the R Journal is to provide peer review for R's sprawling package ecosystem and this only happens through the dedicated, informed, volunteer efforts of the reviewers.

John Verzani
John.Verzani@r-project.org

stplanr: A Package for Transport Planning

by Robin Lovelace, Richard Ellison

Abstract Tools for transport planning should be flexible, scalable, and transparent. The **stplanr** package demonstrates and provides a home for such tools, with an emphasis on spatial transport data and non-motorized modes. The **stplanr** package facilitates common transport planning tasks including: downloading and cleaning transport datasets; creating geographic "desire lines" from origin-destination (OD) data; route assignment, locally and interfaces to routing services such as `CycleStreets.net`; calculation of route segment attributes such as bearing and aggregate flow; and 'travel watershed' analysis. This paper demonstrates this functionality using reproducible examples on real transport datasets. More broadly, the experience of developing and using R functions for transport applications shows that open source software can form the basis of a reproducible transport planning workflow. The **stplanr** package, alongside other packages and open source projects, could provide a more transparent and democratically accountable alternative to the current approach, which is heavily reliant on proprietary and relatively inaccessible software.

Introduction

Transport planning can broadly be defined as the process of designing and evaluating transport interventions (Ortuzar and Willumsen, 2011), usually with the ultimate aim of improving transport systems from economic, social, and environmental perspectives. This inevitably involves a degree of subjective judgment and intuition. With the proliferation of new transport datasets — and the increasing availability of hardware and software to make sense of them — there is great potential for the discipline to become more evidence-based and scientific (Balmer et al., 2009). Transport planners have always undertaken a wide range of computational activities (Boyce and Williams, 2015), but with the digital revolution the demands have grown beyond the capabilities of a single, monolithic product. The diversity of tasks, and the need for democratic accountability in public decision making, suggests that future-proof transport planning software should be:

- flexible, able to handle a wide range of data formats
- scalable, able to work at multiple geographic levels from single streets to large cities and regions
- robust and reliable, tested on a range of datasets and able to work "out of the box" in a range of real-world projects
- open source and reproducible, ensuring transparency and encouraging citizen science

This paper sets out to demonstrate that open source software with a command-line interface (CLI) can provide a foundation for transport planning software that meets each of these criteria. R provides a strong basis for progress in this direction because it already contains functionality used in common transport planning workflows. The **sp**, **rgeos**, and **rgdal** packages greatly improved R's spatial abilities (Bivand et al., 2013), work that is being consolidated and extended in the recent **sf** package.

Building on these foundations, a number of spatial packages have been developed for applied domains including: disease mapping and modelling, with packages such as **SpatialEpi** and **diseasemapping** (Kim and Wakefield, 2016; Brown and Zhou, 2016); spatial ecology, with the **adehabitat** family of packages (Calenge, 2006); and visualisation, with packages such as **leaflet**, **tmap**, **mapview**, and **mapmisc** (Brown, 2016). However, there has been little prior work to develop R functionality designed specifically for transport planning, with the notable exceptions of TravelR (a package on R-Forge last updated in 2012) and gtfsr (a package for handling General Transit Feed Specification (GTFS) data).

The purpose of **stplanr** is to provide a toolbox rather than a specific solution for transport planning, with an emphasis on spatial data and active modes. This emphasis is timely given the recent emphasis on sustainability (Banister, 2008) and 'Big Data' (Zheng et al., 2016) in the wider field of transport planning.

A major motivation was the lack of R packages, and open source software in general, for transport applications. This may be surprising given the ubiquity of transport problems;[1] R's proficiency at handling spatial, temporal and travel survey data that describe transport systems; and the growing popularity of R in applied domains (Jalal et al.; Moore and Hutchinson). Another motivation is the growth in open access datasets: the main purpose of early versions of the package was to process open origin-destination data (Lovelace et al., 2017).

[1]Many people can think of things that could be improved on their local transport networks, especially for walking, cycling and wheel-chairs, but most lack the evidence to communicate the issues, and potential solutions, to others.

R is already used in transport applications, as illustrated by recent research that applies packages from other domains to transport problems. For instance, Efthymiou and Antoniou (2012) use R to analyse the data collected from an online survey focused on car-sharing, bicycle-sharing, and electric vehicles. Efthymiou and Antoniou (2012) also used R to collect and analyse transport-related data from Twitter using packages including **XML**, **twitteR** and **ggplot2**. These packages were used to download, parse and plot the Twitter data using a method that can be repeated and the results reproduced or updated. More general statistical analyses have also been conducted on transport-related datasets using packages including **muStat** and **mgcv** (Diana, 2012; Cerin et al., 2013). Despite the rising use of R for transport research, there has yet been to be a package for transport planning.

The design of the R language, with its emphasis on flexibility, data processing, and statistical modelling, suggests it can provide a powerful environment for transport planning research. There are many quantitative methods in transport planning, many of which fit into the classic 'four stage' transport model which involves the following steps (Ortuzar and Willumsen, 2011): (1) trip *generation* to estimate trip frequency from origins; (2) *distribution* of trips to destinations; (3) *modal split* of trips between walking, cycling, buses etc.; (4) *assignment* of trips to the transport route network. To this we would like to add two more stages for the big data age: (0) data processing and exploration; and (5) validation. This sequence is not the only way of transport modelling and some have argued that its dominance has reduced innovation. However it is certainly a common approach and provides a useful schema for classifying the kinds of task that **stplanr** can tackle:

- accessing and processing of data on transport infrastructure and behaviour (stage 0)
- analysis and visualisation of the transport network (0)
- analysis of origin-destination (OD) data and the visualisation of resulting 'desire lines'
- the allocation of desire lines to roads and other guideways via routing services
- the aggregation of routes to estimate total levels of flow on segments throughout the transport network
- development of models to estimate transport behaviour currently and under various scenarios of change
- the calculation of "catchment areas" affected by transport infrastructure

The automation of such tasks can assist researchers and practitioners to create evidence for decision making. If the data processing and analysis stages are fast and painless, more time can be dedicated to visualisation and decision making. This should allow researchers to focus on problems, rather than on clunky graphical user interfaces (GUIs), and ad-hoc scripts that could be generalised. Furthermore, if the process can be made reproducible and accessible (e.g. via online visualisation packages such as **shiny**), this could help transport planning move away from reliance on "black boxes" (Waddell, 2002) and empower citizens to challenge decisions made by transport planning authorities based on the evidence (Hollander, 2016).

There are many advantages of using a scriptable, interactive, and open source language such as R for transport planning. Such an approach enables: reproducible research; the automation and sharing of code between researchers; reduced barriers to innovation, as anyone can create new features for the benefit of all planners; easier interaction with non domain experts (who will lack dedicated software); and integration with other software systems, as illustrated by the use of **leaflet** to generate JavaScript for sharing interactive maps for transport planning, used in the publicly accessible Propensity to Cycle Tool (Lovelace et al., 2017). Furthermore, R has a strong user community which can support newcomers (**stplanr** was peer reviewed thanks to the community surrounding ROpenSci). The advantages of using R specifically to develop the functionality described in this paper are that it has excellent geo-statistical capabilities (Pebesma et al., 2015), visualisation packages (e.g. **tmap**, **ggplot2**), support for logit models (which are useful for modelling modal shift), and support for the many formats that transport datasets are stored in (e.g., via the **haven** and **rio** packages).

Package structure and functionality

The package can be installed and attached as follows (see the package's README for dependencies and access to development versions):

```
install.packages("stplanr")
```

```
library(stplanr)
```

stplanr imports both **sp** and its successor **sf**. This means that spatial objects used in and produced by the package work with base R generic functions such as `summary`, `aggregate`, and `plot` (Bivand et al., 2013). Furthermore, output objects of class `"sf"` are *mostly* compatible with the popular data processing package **dplyr**.

Core functions and classes

The package's core functions are structured around three common types of spatial transport data:

- Origin-destination (OD) data, which report the number of people travelling between origin-destination pairs. This type of data is not explicitly spatial (OD datasets are usually represented as data frames) but represents movement over space between points in geographical space. An example is provided in the `flow` dataset.
- Line data, one dimensional linear features on the surface of the Earth. These are typically stored as a `"SpatialLinesDataFrame"` object.
- Route data are special types of lines which have been allocated to the transport network. Routes typically result from the allocation of a straight "desire line" allocated to the route network with a `route_` function. Route network represent many overlapping routes. All are typically stored as a `"SpatialLinesDataFrame"` object.

A convention has been developed whereby function names are prefixed depending on the the input data type (`od_`, `line_` and `route_` respectively, although `route_` functions do not take routes as inputs, they output them). A selection of these is presented in Table 1 (`lsf.str("package:stplanr")` returns a list of all functions). Additional "core functions" could be developed, such as those prefixed with `rn_` (for working with route network data) and `geo_` functions for geographic operations such as buffer creation on lat/lon projected data (this function is currently named `buff_geo`).

Function	Input data type(s)	Output data type
`od_dist`	Data frame	Numeric vector
`od_id_order`	Data frame	Data frame
`line_bearing`	Spatial line	Numeric vector
`line_midpoint`	Spatial line	Spatial points
`route_cyclestreet`	Coordinates, spatial point or text	Spatial lines
`route_graphhopper`	Coordinates, spatial point or text	Spatial lines

Table 1: Selection of functions for working with or generating OD, line and route data types.

We aim to preserve type in some functions: `line2route`, for example, takes spatial lines objects and returns spatial lines objects. Type stability has its limitations with spatial data, however: it would be wasteful for functions such as `line_bearing` (which returns the bearing of a line) to duplicate the spatial data contained in its input, for instance. Generic classes enable **stplanr** to handle objects of class `"sf"` from the new **sf** package in a type preserving way.

A class system has not been developed for each data type (this option is discussed in the final section) and more classes would be possible: transport datasets are diverse. This diversity helps explain why some functions have more ad-hoc names. Rather than attempting a systematic description of each of **stplanr**'s functions, the remainder of this paper shows how the package can be used for transport planning, beginning with data access and ending with visualisation.

Road traffic casualty data

Gaining access to data is often the problem in transport planning. It can be a long and protracted process but is becoming easier thanks to the "open data" movement and packages such as **tigris** and **osmdata** (Walker, 2016).

The **stplanr** package helps import data with functions including `read_table_builder`, for importing data from the Australian Bureau of Statistics (ABS), and `dl_stats19` — which has now been split-out into the package **stats19** — for downloading datasets from the UK's Stats19 road traffic casualty system (Lovelace et al., 2019). A brief example of the latter is demonstrated below, which begins with downloading the data (warning this downloads ~100 MB of data):

```
dl_stats19() # download and extract stats19 road traffic casualty data

#> [1] "Data saved at: /tmp/RtmpppF3E2/Accidents0514.csv"
#> [2] "Data saved at: /tmp/RtmpppF3E2/Casualties0514.csv"
#> [3] "Data saved at: /tmp/RtmpppF3E2/Vehicles0514.csv"
```

Once the data has been saved in the default directory, determined by `tempdir`, it can be read-in and cleaned with the `read_stats19_` functions (note these call `format_stats19_` functions internally to clean the datasets and add correct labels to the variables):

```
ac <- read_stats19_ac()
ca <- read_stats19_ca()
ve <- read_stats19_ve()
```

The resulting datasets (representing accident, casualty, and vehicle level data, respectively) can be merged and made geographic, as illustrated below:[2]

```
library(dplyr)
ca_ac <- inner_join(ca, ac)
ca_fatal <- ca_ac %>%
  filter(Casualty_Severity == "Fatal" & !is.na(Latitude)) %>%
  select(Age = Age_of_Casualty, Mode = Casualty_Type, Longitude, Latitude)
ca_sp <- sp::SpatialPointsDataFrame(coords = ca_cycle[3:4], data = ca_cycle[1:2])
```

Now that this casualty data has been cleaned, subsetted (to only include fatal crashes) and converted into a spatial class system, we can analyse the data using geographical datasets of the type commonly used by **stplanr**. The following code, for example, geographically subsets the dataset to include only crashes that occurred within the bounding box of a sample route network dataset contained as a dataset in **stplanr** for illustrative purposes. Note the use of `bb2poly`, which converts a spatial dataset into a box, represented as a rectangular `SpatialPolygonsDataFrame`:

```
data("route_network")
sp::proj4string(ca_sp) <- sp::proj4string(route_network)
bb <- bb2poly(route_network)
sp::proj4string(bb) <- sp::proj4string(route_network)
ca_local <- ca_sp[bb,]
```

The above code chunk shows the importance of understanding geographical data when working with transport data. It is only by converting the casualty data into a spatial data class, and adding a coordinate reference system (CRS), that transport planners and researchers can link this important dataset back to the route network. We can now perform GIS operations on the results. The next code chunk, for example, finds all the fatalities that took place within 100 m of the route network, using the function `buff_geo`:

```
rnet_buff_100 <- buff_geo(route_network, width = 100)
ca_buff <- ca_local[rnet_buff_100,]
```

These can be visualised using base R graphics, extended by **sp**, as illustrated in Figure 1. This provides a good start for analysis but for publication-quality plots and interactive plots, designed for public engagement, we recommend using dedicated visualisation packages that work with spatial data such as **tmap**.

```
plot(bb, lty = 4)
plot(rnet_buff_100, col = "grey", add = TRUE)
points(ca_local, pch = 4)
points(ca_buff, cex = 3)
```

Reading census data

National censuses are one common source of transport data, that frequently include questions on where people live and work. This is often accompanied by a question on what mode was taken. These data are generally provided in standard tables that can be downloaded for free. For the Australian census, both these standard tables are available as well as custom tables that can be produced using a service named TableBuilder. TableBuilder generates Excel or CSV files that contain additional lines that make reading these data into R difficult. The **stplanr** package provides the `read_table_builder` function to read in these files.

Using the example `SA1Population.xlsx` file included with **stplanr** that contains the population by SA1 zone (a statistical area used for the Australian census), we can use the `read_table_builder` function to read and format the table for use in R. The function automatically removes the additional lines and sets the column headers and data types as appropriate. The result is a data frame that can be used with GIS boundary files and other datasets produced for SA1 zones.

[2]Note the `inner_join` function from the **dplyr** package was used because it is substantially faster and more effective than the equivalent function, `merge`, in base R. This is communicated in a number of places, including on the website zevross.com. For this reason we import **dplyr** and use it internally for joins.

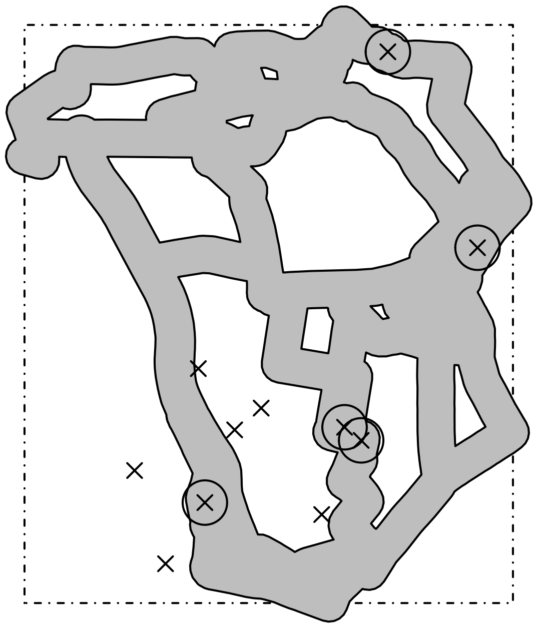


Figure 1: Road traffic fatalities in the study area downloaded with with stplanr (crosses). Deaths that happened within 100 m of the route network are represented by circles.

```
data_dir <- system.file("extdata", package = "stplanr")
t2 <- read_table_builder(file.path(data_dir, 'SA1Population.xlsx'),
                         filetype = 'xlsx', sheet = 1, removeTotal = TRUE)
```

Bicycle share data

The **stplanr** package can also be used in conjunction with complementary R packages for downloading data from Open Street Maps (OSM) using **osmdata** and bicycle share data using the **bikedata** package.

The bicycle share data that can be accessed using the **bikedata** package is particularly well suited for integration with **stplanr** as it produces origin-destination (OD) flows from bicycle sharing systems. This data can be used together with the sum_network_links function to generate the likely paths and how these overlap. This can be used to generate heatmaps of a road network showing modelled common routes such as in Figure 2.[3]

Creating geographic desire lines

Origin-destination (OD) data, which represent the number of people travelling between geographical zones, is a key input for transport planning (Calabrese et al., 2011). OD data usually represent an aggregate data source, and are therefore able to represent the travel patterns of an entire country in a file of manageable size (see wicid.ukdataservice.ac.uk/ for example). They can be stored as a (sparse) matrix or (more commonly) a long table of OD pairs. The long form is illustrated in the code chunk below which shows a sample of the flow object. flow is a data frame representing the number of home-work commutes by mode between residential areas in the UK, provided provided in **stplanr** for teaching and demonstration purposes (see ?flow to see how this dataset was created):

```
data("flow", package = "stplanr")
head(flow[c(1:3, 12)])
```

```
#>        Area.of.residence Area.of.workplace All Bicycle
#> 920573         E02002361         E02002361 109       2
#> 920575         E02002361         E02002363  38       0
#> 920578         E02002361         E02002367  10       0
#> 920582         E02002361         E02002371  44       3
#> 920587         E02002361         E02002377  34       0
#> 920591         E02002361         E02002382   7       0
```

[3]The figure is based on the shortest path, although other criteria could be used by setting weights in the network accordingly — see https://github.com/ropensci/stplanr/issues/194 in the package's issue tracker for details.

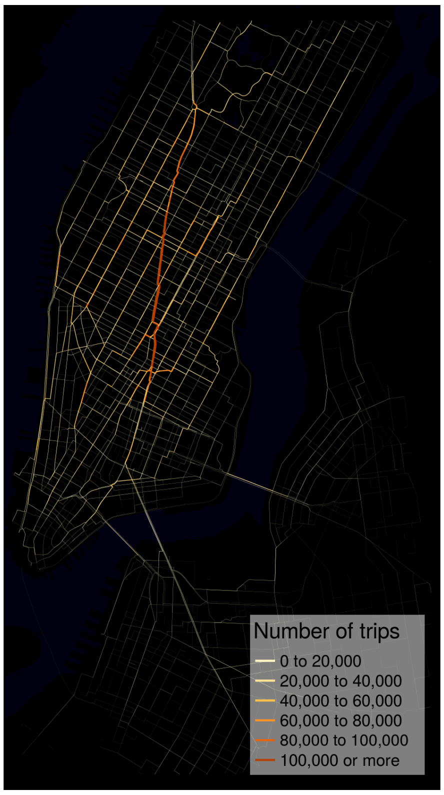


Figure 2: Modelled common routes for bicycle share trips in New York City.

Although the flow data displayed above describes movement over geographical space, it contains no explicitly geographical information. Instead, the coordinates of the origins and destinations are linked to a separate geographical dataset (represented by the `cents` dataset), as illustrated below:[4]

```
data("cents", package = "stplanr")
as.data.frame(cents[1:3, -c(3,4)])
```

```
#>       geo_code  MSOA11NM coords.x1 coords.x2
#> 1708 E02002384 Leeds 055 -1.546463  53.80952
#> 1712 E02002382 Leeds 053 -1.511861  53.81161
#> 1805 E02002393 Leeds 064 -1.524205  53.80410
```

A common task is linking an OD dataset (e.g., `flow`) to a geographic dataset representing zone centroids (e.g. `cents`). We use `od2line` to combine them, as illustrated in the code chunk below, which creates an object named `l`, a spatial object that will be visualised in the next section:

```
l <- od2line(flow = flow, zones = cents)
```

A larger example represents flights from New York in 2013 from the **nycflights13** package. The code chunk below makes use of the function `geo_buffer`, and highlights **stplanr**'s ability to work with `"Spatial"` or `"sf"` objects interchangeably. Figure 3 demonstrates how the resulting spatial object can be plotted interactively using packages such as **tmap** (see the accompanying R file for visualisation code).

```
library(nycflights13)
airports_sf <- sf::st_as_sf(airports, coords = c("lon", "lat"), crs = 4326)
ny_buff <- geo_buffer(airports_sf[airports_sf$faa == "NYC",], dist = 1e6)
airports_near <- airports_sf[ny_buff,]
flights_near <- flights[flights$dest %in% airports_near$faa,]
flights_agg <- dplyr::group_by(flights_near, origin, dest) %>%
  dplyr::summarise(Flights = n())
flights_sf = od2line(flow = flights_agg, zones = airports_near)
plot(flights_sf, lwd = flights_sf$Flights / 1e3)
```

[4]Such geographic datasets are best represented as in a spatial class system, explaining **stplanr**'s close integration with R's spatial packages.

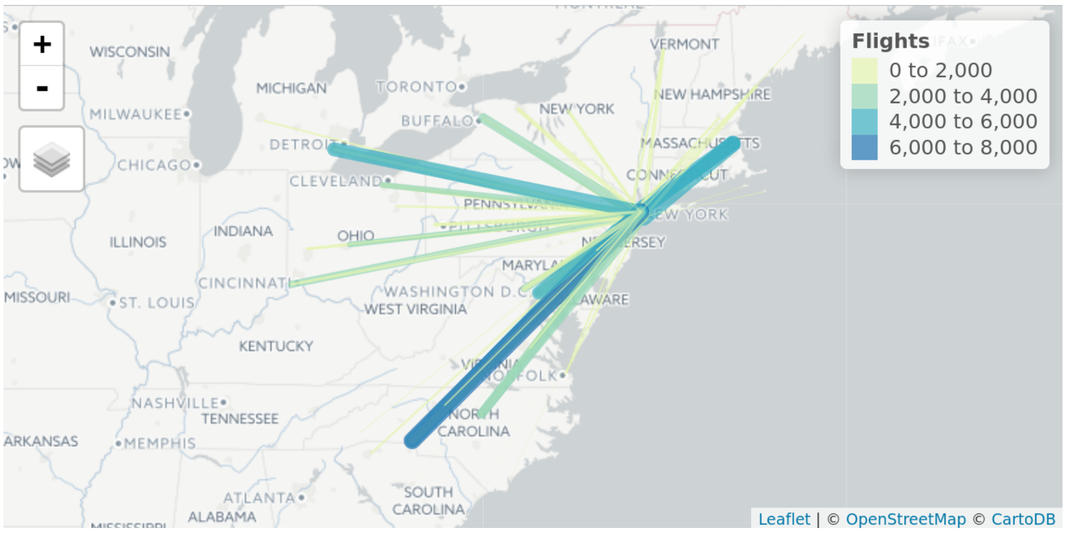


Figure 3: Flights from New York to airports within 1000 km. Data from the **nycflights** package converted to geographic desire lines with the `od2line` function.

Allocating flows to the transport network

A common problem faced by transport researchers is network allocation: converting the 'as the crow flies' lines illustrated in Figure 3 into routes. These are the complex, winding paths that people and animals make to avoid obstacles such as buildings and to make the journey faster and more efficient (e.g. by following the route network).

This is difficult (and was until recently near impossible using free software) because of the size and complexity of transport networks, the complexity of realistic routing algorithms and need for context-specificity in the routing engine. Inexperienced cyclists, for example, would take a very different route than a heavy goods vehicle. The **stplanr** package tackles this issue in two ways: by providing routing functionality that can work on locally stored data and by using third party APIs. Each approach has advantages: local routing functionality is fast and free, whereas online routing services can be more sophisticated (e.g., taking into account local traffic conditions) and work anywhere in the world without needing to download unwieldy datasets and new software libraries such as OSRM. A disadvantage of relying on online services is that they may break or change without warning in the future.

Route allocation is undertaken by `route_` functions such as `route_cyclestreets` and `route_graphhopper`. These allocate a single OD pair (represented by `from` and `to` arguments as coordinates, spatial point objects, or text strings to be "geo-coded") to the transport network. This is illustrated below with `route_cyclestreet`, which uses the CycleStreets.net API, a routing service "by cyclists for cyclists" which provides "fastest", "quietest," and "balanced" routes:[5]

```
route_bl <- route_cyclestreet(from = "Bradford, Yorkshire", to = "Leeds, Yorkshire")
route_c1_c2 <- route_cyclestreet(cents[1,], cents[2,])
```

The raw output from routing APIs is usually provided as a JSON or GeoJSON text string. By default, `route_cyclestreet` saves a number of key variables (including length, time, hilliness and busyness variables generated by CycleStreets.net) from the attribute data provided by the API. If the user wants to save the raw output, the `save_raw` argument can be used:

```
route_bl_raw <- route_cyclestreet(from = "Bradford", to = "Leeds", save_raw = TRUE)
```

Additional arguments taken by the `route_` functions depend on the routing function in question. By changing the `plan` argument of `route_cyclestreet` to `fastest`, `quietest`, or `balanced`, for example, routes favouring speed, quietness or a balance between speed and quietness will be saved, respectively.

To automate the creation of route-allocated lines over many desire lines, the `line2route` function loops over each line, wrapping any `route_` function as an input. The output is a `SpatialLinesDataFrame` with the same number of dimensions as the input dataset (see the right panel in Figure 4).

[5] An API key is needed for this function to work. This can be requested (or purchased for large scale routing) from cyclestreets.net/api/apply. See `?route_cyclestreet` for details. Thanks to Martin Lucas-Smith and Simon Nuttall for making this possible. Note: an updated interface to `cyclestreets.net` has been made available via the package **cyclestreets**.

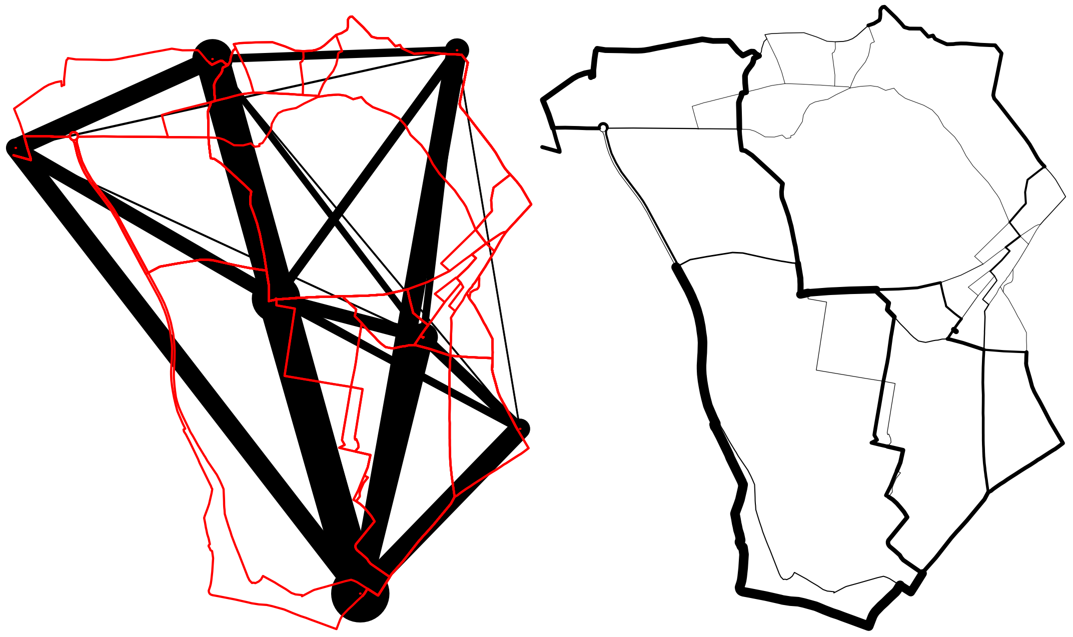


Figure 4: Visualisation of travel desire lines, with width proportional to number of trips between origin and destination (black) and routes allocated to network (red) in the left-hand panel. The right hand panel shows the route network dataset generated by overline().

```
routes_fast <- line2route(l = l, route_fun = route_cyclestreet)
```

The result of this 'batch routing' exercise is illustrated in Figure 4. The red lines in the left hand panel are very different from the hypothetical straight 'desire lines', highlighting the importance of this route-allocation functionality (see `vignette("introducing-stplanr")` and kateto.net for more sophisticated visualisation methods).

```
plot(route_network, lwd=0)
plot(l, lwd = l$All / 10, add = TRUE)
lines(routes_fast, col = "red")
routes_fast$All <- l$All
rnet <- overline(routes_fast, "All", fun = sum)
rnet$flow <- rnet$All / mean(rnet$All) * 3
plot(rnet, lwd = rnet$flow / mean(rnet$flow))
```

To estimate the amount of capacity needed at each segment on the transport network, the `overline` function demonstrated above, is used to divide line geometries into unique segments and aggregate the overlapping values. The results, illustrated in the right-hand panel of Figure 4, could be used to inform the decision making process at the route network level, such as where to create new bus routes cycle paths.

Limitations with the `route_cyclestreet` routing API include its specificity to one mode (cycling) and a single region (the UK and part of Europe). To overcome these limitations, additional routing APIs were added with the functions `route_graphhopper`, `route_transportapi_public`, and `viaroute`. These interface to Graphhopper, TransportAPI, and the Open Source Routing Machine (OSRM) routing services, respectively. Advanced users can set-up local routing services (as demonstrated in a guide by DigitalOcean), reducing the disadvantages of using online routing services (they rely on on potentially slow internet connections, changeable APIs, and variable/high prices).

A short example of finding the route by car and bike between New York and Oaxaca demonstrates how `route_graphhopper` can collect geographical and other data on routes by various modes, anywhere in the world. The output, shown in Table 2, shows that the function also saves time, distance, and (for bike trips) vertical distance climbed for the trips.

```
ny2oaxaca1 <- route_graphhopper("New York", "Oaxaca", vehicle = "bike")
ny2oaxaca2 <- route_graphhopper("New York", "Oaxaca", vehicle = "car")
rbind(ny2oaxaca1@data, ny2oaxaca2@data)
```

mode	time	dist	change_elev
Bike	17522.73	4885663	87388.13
Car	2759.89	4754772	NA

Table 2: Results obtained from the Graphhopper API using the `route_graphhopper` function to estimate the time taken and route distance to travel between New York and Oaxaca by cycling and driving.

Modelling travel catchment areas

Catchment areas are useful analytic and visual tools in transport planning. They can help who will benefit from a particular transport intervention (such as a new bus stop) and illustrate the geographic area that is covered by (or omitted from) a particular service or transport system, to help prioritize new investment. Passengers are often said to be willing to walk up to 400 metres to a bus stop or 800 metres to a railway station (El-Geneidy et al., 2014), leading to surrounding smaller or larger polygons representing the catchment within which people would be willing to travel. Such catchment areas have been criticised as being arbitrary or as underestimating the true sphere of influence of public transport nodes (El-Geneidy et al., 2014; Daniels and Mulley, 2013). However, they nonetheless represent a good starting point from which the geographical and social distribution of service provision can be estimated.

Often catchment areas are calculated using straight-line (or "as the crow flies") distances. This approach is appealing because it requires little additional data and is simple to compute (using buffer function) and understand.

The **stplanr** package provides functionality that calculates catchment areas using straight-line distances with the `calc_catchment` function. This function takes a `"SpatialPolygonsDataFrame"` object that contains the population (or other) data, typically from a census, and a `Spatial*` layer that contains the geometry of the transport facility. These two layers are overlayed to calculate statistics for the desired catchments including "proportioning" polygons to account for the proportion located within the catchment area.

To illustrate this functionality, the following code chunk reads-in sample datasets stored in the common ESRI Shapefile format using the `readOGR` function from **rgdal**. The resulting object `smallsa1` contains population data for Statistical Area 1 (SA1) zones in Sydney, Australia. The `testcycleway` data set contains hypothetical cycleways aligned to streets in Sydney:

```
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, 'smallsa1.zip'))
unzip(file.path(data_dir, 'testcycleway.zip'))
sa1income <- rgdal::readOGR(".", "smallsa1")
testcycleway <- rgdal::readOGR(".", "testcycleway")
# Remove unzipped files
file.remove(list.files(pattern = "^(smallsa1|testcycleway).*"))
```

The calculation of catchment areas requires two parameters: a vector containing column names to calculate statistics and a distance. Since proportioning the areas assumes projected data, unprojected data are automatically projected to either a common projection (if one is already projected) or a specified projection. It should be emphasized that the choice of projection is important and has an effect on the results meaning setting a local projection is recommended to achieve the most accurate results. The catchment area is calculated as follows:

```
catch800m <- calc_catchment(
  polygonlayer = sa1income,
  targetlayer = testcycleway,
  calccols = c('Total'),
  distance = 800,
  projection = 'austalbers',
  dissolve = TRUE
)
```

The result can be used to calculate the total population within the catchment areas of the cycleway, with the command '`sum(catch800m$Total)`': 39418 people. The catchment area can then be visualised (see Figure 5):

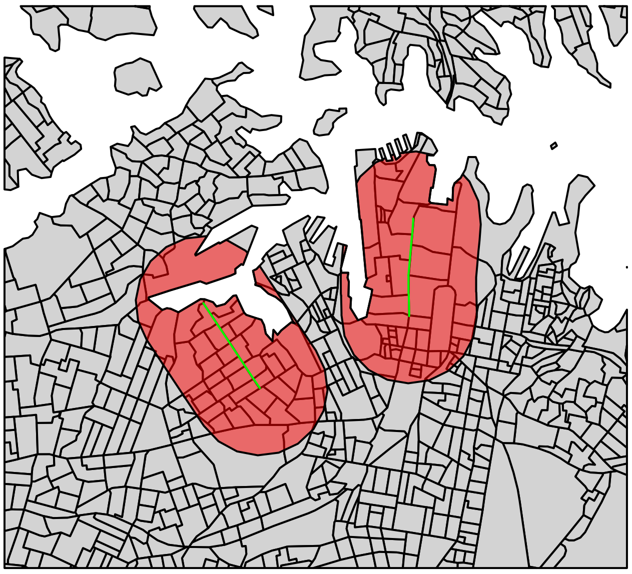


Figure 5: An 800 metre catchment area (red) associated with a cycle path (green) using straight-line distance in Sydney.

```
plot(sa1income, col = "light grey")
plot(catch800m, col = rgb(1, 0, 0, 0.5), add = TRUE)
plot(testcycleway, col = "green", add = TRUE)
```

This simplistic catchment area is useful when the straight-line distance is a reasonable approximation of the route taken to walk (or cycle) to a transport facility. However, this is often not the case. The catchment area in Figure 5 initially appears reasonable but the red-shaded catchment area includes an area that requires travelling around a bay to access from the (green-coloured) cycleway: users may have *access* but that does not mean it's *accessible*. To allow for more realistic catchment areas for most situations, **stplanr** provides the `calc_network_catchment` function that uses the same principle as `calc_catchment` but also takes into account the transport network.

To use `calc_network_catchment`, a transport network, as a `"SpatialLinesNetwork"` object, must be provided. This combines a `"SpatialLinesDataFrame"` object with a graph network (using the **igraph** package) to allow routing (estimation of shortest paths). The network is used to calculate the shortest actual paths within the specific catchment distance, as demonstrated in the following code chunk:

```
unzip(file.path(data_dir, 'sydroads.zip'))
sydroads <- rgdal::readOGR(".", "roads")
file.remove(list.files(pattern = "^(roads).*"))
sydnetwork <- SpatialLinesNetwork(sydroads)
```

The network catchment is then calculated using a similar method as with `calc_catchment` but with a few minor changes. Specifically these are including the `SpatialLinesNetwork`, and using the `maximpedance` parameter to define the distance, with distance being the additional distance from the network. In contrast to the distance parameter that is based on the straight-line distance in both the `calc_catchment` and `calc_network_catchment` functions, the `maximpedance` parameter is the maximum value in the units of the network's weight attribute. In practice this is generally distance in metres but can also be travel times, risk or other measures.

```
netcatch800m <- calc_network_catchment(
  sln = sydnetwork,
  polygonlayer = sa1income,
  targetlayer = testcycleway,
  calccols = c('Total'),
  maximpedance = 800,
  distance = 100,
  projection = 'austalbers'
)
```

Once calculated, the network catchment area can be used just as the straight-line network catchment. This includes extracting the catchment population of 23457 and plotting the original catchment

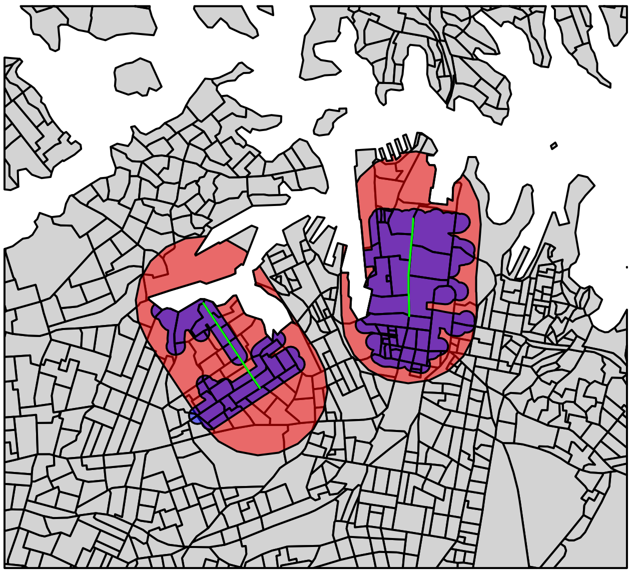


Figure 6: A 800 metre network catchment are (blue) compared with a catchment area based on Euclidean distance (red) associated with a cycle path (green).

area together with the original area with the results shown in Figure 6:

```
plot(sa1income, col = "light grey")
plot(catch800m, col = rgb(1, 0, 0, 0.5), add = TRUE)
plot(netcatch800m, col = rgb(0, 0, 1, 0.5), add = TRUE)
plot(testcycleway, col = "green", add = TRUE)
```

The `calc_catchment` and `calc_network_catchment` functions are complementary functions allowing for the computation of catchment areas. Although for many transport applications it would be better to use the `calc_network_catchment` function that uses the true network distances, it may still be useful to use the straight-line version in some cases. These include calculating the population that may be affected by noise levels from transport infrastructure (such as a motorway) where the effects of noise are not limited by the network. In situations where network data is not available it would be possible to use the `calc_catchment` function together with a fixed assumption about what straight-line distance corresponds to the desired network distance (i.e., 1km network = 700m straight-line). It may also be appropriate to use the straight-line distance when the network is not a reasonable constraint. This may be because the infrastructure of interest is accessed primarily by walking or cycling in an area where these are not limited to the road network (such as parks or shopping areas with passthroughs or overpasses available to pedestrians).

Modelling and visualisation

Analysing mode use

Route-allocated lines allow estimation of *route distance* and therefore *circuity (Q)* (route distance divided by Euclidean distance) (Levinson and El-Geneidy, 2009):

$$Q = \frac{d_{Rf}}{d_E},$$

where (d_E) and (d_{Rf}) represent Euclidean and fastest route distance respectively.

These variables can help model the rate of flow between origins and destination, as illustrated in the left-hand panel of Figure 7. The code below demonstrates how objects generated by **stplanr** can be used to undertake such analysis, with the `line_length` function used to find the distance, in meters, of lat/lon data.

```
l$d_euclidean <- line_length(l)
l$d_rf <- routes_fast@data$length
plot(l$d_euclidean, l$d_rf,
  xlab = "Euclidean distance", ylab = "Route distance")
abline(a = 0, b = 1)
abline(a = 0, b = 1.2, col = "green")
abline(a = 0, b = 1.5, col = "red")
```

The left hand panel of Figure 7 shows the expected strong correlation between Euclidean (d_E) and fastest route (d_{Rf}) distance. However, some OD pairs have a proportionally higher route distance than others, as illustrated by distance from the black line in the plot.

An extension to the concept of circuity is the "quietness diversion factor" (QDF) of a desire line (Lovelace et al., 2017), the ratio of the route distance of a quiet route option (d_{Rq}) to that of the fastest:

$$QDF = \frac{d_{Rq}}{d_{Rf}}.$$

Thanks to the "quietest" route option provided by `route_cyclestreet`, we can estimate average values for both metrics as follows:

```
routes_slow <- line2route(l, route_cyclestreet, plan = "quietest")

l$d_rq <- routes_slow$length # quietest route distance
Q <- mean(l$d_rf / l$d_euclidean, na.rm = TRUE)
QDF <- mean(l$d_rq / l$d_rf, na.rm = TRUE)
Q

#> [1] 1.298767

QDF

#> [1] 1.034721
```

The results show that cycle paths are not particularly direct in the study region by international standards (CROW, 2007). This is hardly surprisingly given the small size of the sample and the short distances covered: Q tends to decrease at a decaying rate with distance, meaning longer paths tend to be more direct. What is surprising is that the quietness diversion factor is close to unity, which could imply that the quiet routes are constructed along direct, and therefore sensible routes. When time is explored, we find that the "quietness diversion factor with respect to time" (QDF_t) is slightly larger, although larger datasets would be needed for any inferences to be made:

```
(QDFt <- mean(routes_slow$time / routes_fast$time, na.rm = TRUE))

#> [1] 1.052855
```

Analyzing and estimating trip flows

The second step in the four-stage transport model (after trip generation) is trip distribution. This involves identifying destinations for trips generated in each zone (creating OD data), which can be done with a gravity model, logit model, or other technique under the broad banner of "spatial interaction model" (SIM). The **stplanr** package is not intended primarily as an SIM package, but could be extended in this direction, as illustrated by the `od_radiation` function, which implements a SIM proposed by Simini et al. (2012). Further development in this direction (or integration with a package dedicated to SIMs) would complement **stplanr**'s focus on geographic functions.

At present there are no functions for modelling distance decay, but this is something we would like to add in future versions of **stplanr**. Although it is only one of several methods used for modelling trip distribution as part of four-step models, distance decay is an especially important concept for sustainable transport planning due to physical limitations on the ability of people to walk and cycle large distances (Iacono et al., 2010). It can also be used to model relationships between locations in situations that are sensitive to distance (cycling facilities for instance) in addition to its use in four-step models.

We can explore the relationship between distance and the proportion of trips made by walking using the same object `l` generated by **stplanr**.

```
l$pwalk <- l$On.foot / l$All
plot(l$d_euclidean, l$pwalk, cex = l$All / 50,
  xlab = "Euclidean distance (m)", ylab = "Proportion of trips by foot")
```

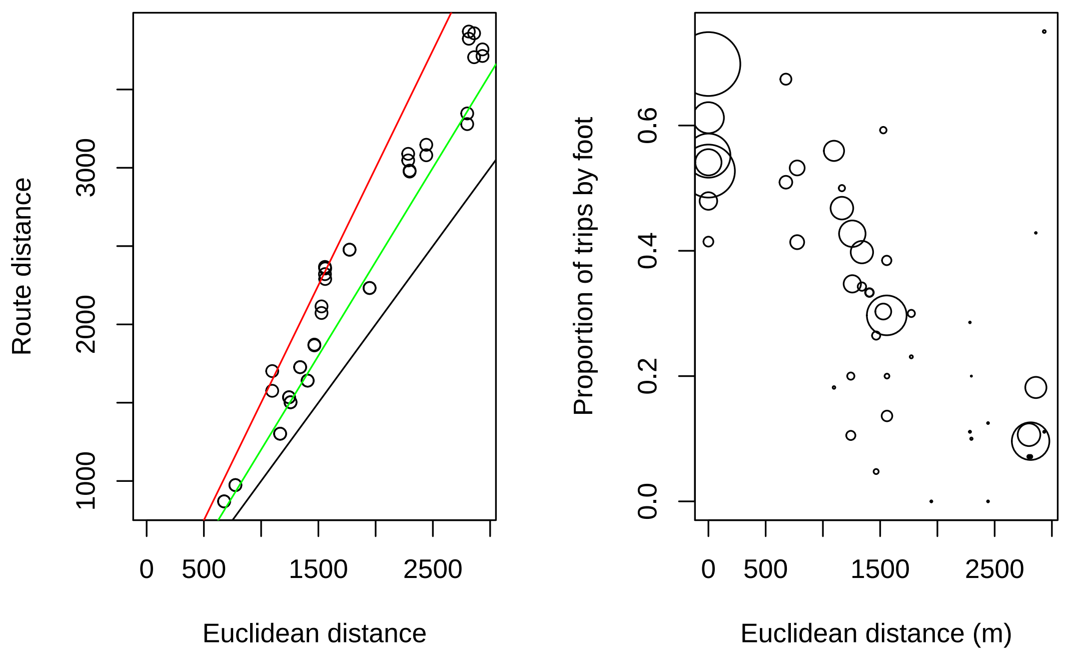


Figure 7: Euclidean and fastest route distance of trips in the study area (left) and Euclidean distance vs the proportion of trips made by walking (right).

Based on the right-hand panel in Figure 7, there is a clear negative relationship between distance of trips and the proportion of those trips made by walking. This is unsurprising: beyond a certain distance (around 1.5km according the the data presented in the figure above) walking is usually seen as too slow and other modes are considered. This "distance decay" is non-linear and can be approximated by a range of functional forms (Martínez and Viegas, 2013). From the range of options we test below just two forms. We will compare the ability of linear and log-square-root functions to fit the data contained in l for walking.

```
lm1 <- lm(pwalk ~ d_euclidean, data = l@data, weights = All)
lm2 <- lm(pwalk ~ d_rf, data = l@data, weights = All)
lm3 <- glm(pwalk ~ d_rf + I(d_rf^0.5),
           data = l@data, weights = All, family = quasipoisson(link = "log"))
```

The results of these regression models can be seen using `summary()`. Surprisingly, Euclidean distance was a better predictor of walking than route distance, but no strong conclusions can be drawn from this finding, with such a small sample of desire lines (n = 42). The results are purely illustrative, of the possibilities created by using **stplanr** in conjunction with R's modelling capabilities (see Figure 8).

```
plot(l$d_euclidean, l$pwalk, cex = l$All / 50,
  xlab = "Euclidean distance (m)", ylab = "Proportion of trips by foot")
l2 <- data.frame(d_euclidean = 1:5000, d_rf = 1:5000)
lm1p <- predict(lm1, l2)
lm2p <- predict(lm2, l2)
lm3p <- predict(lm3, l2)
lines(l2$d_euclidean, lm1p)
lines(l2$d_euclidean, exp(lm2p), col = "green")
lines(l2$d_euclidean, exp(lm3p), col = "red")
```

Visualization

Visualization is an important aspect of any transport study, as it enables researchers to communicate their findings to other researchers, policy-makers, and, ultimately, the public. It may therefore come as a surprise that **stplanr** contains no functions for visualisation. Instead, users are encouraged to make use of existing spatial visualisation tools in R, such as **tmap**, **leaflet**, and **ggmap** (Cheshire and Lovelace, 2015; Kahle and Wickham, 2013).

Furthermore, with the development of online application frameworks, such as **shiny**, it is now easier than ever to make the results of transport analysis and modelling projects available to the public.

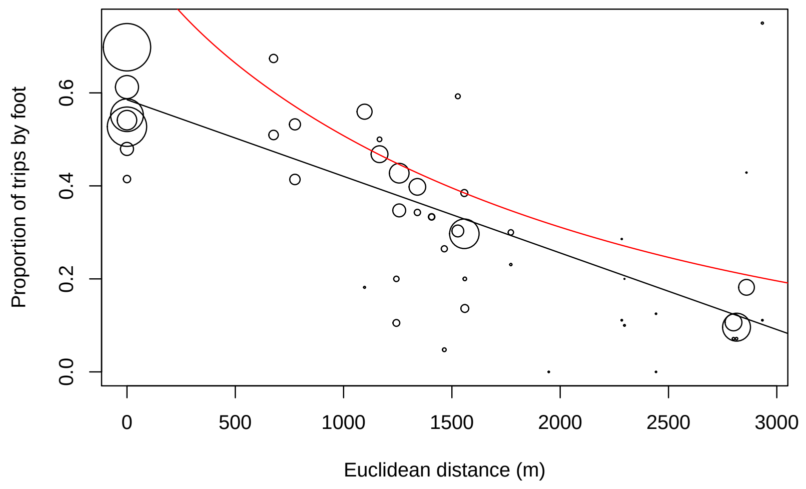


Figure 8: Relationship between Euclidean distance and walking.

There is great potential to expand on the principle of publicly accessible transport planning tools via "web apps", perhaps through new R packages dedicated to visualising and communicating transport data.

Future directions of travel

This paper demonstrates that transport planning can be done with R. We set out reasons for using open source software in general and command-line interfaces in particular for reproducibility, transparency, and democratic accountability. This is, to our knowledge, the first R package aimed at enabling practitioners and researchers to design more sustainable transport systems. Much more is possible in this direction.

The **stplanr** package focuses on geographic analysis of transport systems, a niche that is apparent from existing tools. Proprietary software such as INRO, PTV Visum, and TransCAD dominate geospatial methods in applied transport planning. This differs from other fields that use geospatial methods such as spatial epidemiology and ecology, where open source software dominate. Furthermore, the successful "UNIX approach" to software development implies that each piece of software does a limited number of things well. To avoid the pitfall of "feature creep" it makes sense to specialise.

The lack of existing packages that do the same job direct our attention towards **stplanr**'s basic functionality rather than more advanced features. A good example of this is the `SpatialLinesNetwork` class which could provide a basis for future developments including the recently implemented `route_local` function. This direction of "incremental growth" could proceed by adding interfaces to more transport planning APIs, including Routino and Directions provided by the mapping company Mapbox.

Another example of the potential to develop basic functionality rather than new advanced features is the the possibility of creating an `"OD"` class to represent origin-destination data, which could ease the geographical analysis of OD data via generic methods. Building on the example data presented in previous sections the idea is for a single `OD` object to be able represent `flowlines`, `routes_fast` and `routes_quiet`. The recent development of the **sf**, with its support for multiple geometry columns, makes this option more feasible.

Another direction of travel is towards better support for large datasets. We have demonstrated **stplnr**'s capabilities on deliberately small datasets for ease of understanding. However, the third criterion of transport planning software set-out in the introduction — *scalability* — is vital to package's real world utility in the age of transport-related "Big Data" (Lovelace et al., 2016). Some functions use C++ (via **Rcpp**) for computationally intensive operations, and we plan to continue the drive toward performant code. Opportunities for improving performance include: integration with transport databases (e.g., via the **bikedata** package); support for batch routing APIs; parallel implementations of time-consuming functions; and the refactoring of slow functions (e.g., `overline`, which takes hours to run on large datasets).

Rather than a one-size-fits-all package, we see **stplanr** as an additional tool in the transport planner's cabinet. It is part of a wider movement that is making transport planning a more open and democratic process. Other developments in this movement include the increasing availability of open data (Naumova, 2016) and the rise of open source products for transport modelling, such as

SUMO, MATSim, MITSIMLAB (Saidallah et al., 2016), and pgRouting. In this context, **stplanr**'s focus on GIS operations represents a niche in the market, allowing it to complement such software and help make better use of new open data sources. The **stplanr** package could also be used alongside other R packages that use spatial transport data such as **aspace** and **MCI** (Wieland, 2017).

The **stplanr** package was first developed to generate data for the Propensity to Cycle Tool (PCT), which estimates cycling potential down to the street level across all major cyclable routes in England and Wales (Lovelace et al., 2017). We believe there are many other national and international transport planning challenges the package could be used to solve. In the context of the increasing availability of open access data[6] packages for analyzing transport datasets. The **stplanr** package could help ensure that the evidence on which transport planning decisions are based is reproducible, systematic, and therefore democratically accountable.

Acknowledgements

We would like to thank a number of people and organisations who have supported the development of **stplanr**: the UK's Department for Transport (DfT) for funding the Propensity to Cycle Tool (see www.pct.bike), which instigated code which eventually became **stplanr**; Colin Gillespie, whose teaching helped turn the code into a package; ROpenSci, for hosting the package (special thanks to Scott Chamberlin who reviewed the package); to Maëlle Salmon, Mark Padgham, Martin Lucas-Smith, and Marcus Young for commenting on early drafts of this paper; and to transport planning professionals Tom van Vuren (Mott MacDonald), John Parkin (University of the West of England), Helen Bowkett (Welsh Government), and Yaron Hollander, who provided input on the paper and ideas for future priorities.

Bibliography

M. Balmer, M. Rieser, and K. Nagel. MATSim-T: Architecture and simulation times. *Multi-agent systems for traffic and transportation engineering*, pages 57–78, 2009. URL https://svn.vsp.tu-berlin.de/repos/public-svn/publications/vspwp/2008/08-03/3aug08.pdf. [p7]

D. Banister. The sustainable mobility paradigm. *Transport Policy*, 15(2):73–80, 2008. URL https://doi.org/10.1016/j.tranpol.2007.10.005. [p7]

R. S. Bivand, E. J. Pebesma, and V. Gómez-Rubio. *Applied spatial data analysis with R*. Springer, New York, 2013. [p7, 8]

G. Boeing. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017. URL https://doi.org/10.1016/j.compenvurbsys.2017.05.004. [p21]

D. E. Boyce and H. C. W. L. Williams. *Forecasting Urban Travel: Past, Present and Future*. Edward Elgar Publishing, 2015. [p7]

P. E. Brown. Maps, Coordinate Reference Systems and Visualising Geographic Data with mapmisc. *The R Journal*, 8(1):64–91, 2016. URL https://journal.r-project.org/archive/2016/RJ-2016-005/index.html. [p7]

P. E. Brown and L. Zhou. diseasemapping: Modelling Spatial Variation in Disease Risk for Areal Data, 2016. URL https://CRAN.R-project.org/package=diseasemapping. R package version 1.4.2. [p7]

F. Calabrese, G. Di Lorenzo, L. Liu, and C. Ratti. Estimating Origin-Destination Flows Using Mobile Phone Location Data. *IEEE Pervasive Computing*, 10(4):36–44, 2011. URL https://doi.org/10.1109/MPRV.2011.41. [p11]

C. Calenge. The package adehabitat for the R software: tool for the analysis of space and habitat use by animals. *Ecological Modelling*, 197:1035, 2006. [p7]

E. Cerin, C. H. P. Sit, A. Barnett, M. C. Cheung, and W. M. Chan. Walking for recreation and perceptions of the neighborhood environment in older Chinese urban dwellers. *Journal of Urban Health*, 90(1): 56–66, 2013. URL https://doi.org/10.1007/s11524-012-9704-8. [p8]

[6]Two examples of this are the **osmdata** CRAN package (Padgham et al., 2017) and the Python package OSMnx (Boeing, 2017), for downloading and processing open access datasets from OpenStreetMap.

J. Cheshire and R. Lovelace. Spatial data visualisation with R. In C. Brunsdon and A. Singleton, editors, *Geocomputation*, pages 1–14. SAGE Publications, 2015. URL https://github.com/geocomPP/sdv. [p19]

CROW. *Design manual for bicycle traffic*. Kennisplatform, Amsterdam, 2007. URL http://www.crow.nl/publicaties/design-manual-for-bicycle-traffic. [p18]

R. Daniels and C. Mulley. Explaining walking distance to public transport: The dominance of public transport supply. *Journal of Transport and Land Use*, 6(2):5, 2013. URL https://doi.org/10.5198/jtlu.v6i2.308. [p15]

M. Diana. Studying Patterns of Use of Transport Modes Through Data Mining. *Transportation Research Record: Journal of the Transportation Research Board*, 2308:1–9, 2012. URL https://doi.org/10.3141/2308-01. [p8]

D. Efthymiou and C. Antoniou. Use of Social Media for Transport Data Collection. *Procedia - Social and Behavioral Sciences*, 48:775–785, 2012. URL http://dx.doi.org/10.1016/j.sbspro.2012.06.1055. [p8]

A. El-Geneidy, M. Grimsrud, R. Wasfi, P. Tétreault, and J. Surprenant-Legault. New evidence on walking distances to transit stops: Identifying redundancies and gaps using variable service areas. *Transportation*, 41(1):193–210, 2014. URL https://doi.org/10.1007/s11116-013-9508-z. [p15]

Y. Hollander. *Transport Modelling for a Complete Beginner*. CTthink!, 2016. [p8]

M. Iacono, K. J. Krizek, and A. El-Geneidy. Measuring non-motorized accessibility: issues, alternatives, and execution. *Journal of Transport Geography*, 18(1):133–140, 2010. URL https://doi.org/10.1016/j.jtrangeo.2009.02.002. [p18]

H. Jalal, P. Pechlivanoglou, E. Krijkamp, F. Alarid-Escudero, E. Enns, and M. G. M. Hunink. An Overview of R in Health Decision Sciences. *Medical Decision Making*, page 0272989X16686559. URL https://doi.org/10.1177/0272989X16686559. [p7]

D. Kahle and H. Wickham. ggmap: Spatial Visualization with ggplot2. *The R Journal*, 5(1):144–161, 2013. URL https://journal.r-project.org/archive/2013/RJ-2013-014/index.html. [p19]

A. Y. Kim and J. Wakefield. SpatialEpi: Methods and Data for Spatial Epidemiology, 2016. URL https://CRAN.R-project.org/package=SpatialEpi. R package version 1.2.2. [p7]

D. Levinson and A. El-Geneidy. The minimum circuity frontier and the journey to work. *Regional Science and Urban Economics*, 39(6):732–738, 2009. URL https://doi.org/10.1016/j.regsciurbeco.2009.07.003. [p17]

R. Lovelace, M. Birkin, P. Cross, and M. Clarke. From Big Noise to Big Data: Toward the Verification of Large Data sets for Understanding Regional Retail Flows. *Geographical Analysis*, 48(1):59–81, 2016. URL https://doi.org/10.1111/gean.12081. [p20]

R. Lovelace, A. Goodman, R. Aldred, N. Berkoff, A. Abbas, and J. Woodcock. The Propensity to Cycle Tool: An open source online system for sustainable transport planning. *Journal of Transport and Land Use*, 10(1), 2017. URL https://doi.org/10.5198/jtlu.2016.862. [p7, 8, 18, 21]

R. Lovelace, M. Morgan, L. Hama, and M. Padgham. Stats19: A package for working with open road crash data. *Journal of Open Source Software*, 2019. doi: 10.21105/joss.01181. [p9]

L. M. Martínez and J. M. Viegas. A new approach to modelling distance-decay functions for accessibility assessment in transport studies. *Journal of Transport Geography*, 26:87–96, 2013. URL https://doi.org/10.1016/j.jtrangeo.2012.08.018. [p19]

R. D. D. Moore and D. Hutchinson. Why Watershed Analysts Should Use R for Data Processing and Analysis. *Confluence: Journal of Watershed Science and Management*, 1(1). URL http://confluence-jwsm.ca/index.php/jwsm/article/view/2. [p7]

I. Naumova. *Building Traffic Models Using Freely Available Data*. PhD thesis, University of Hasselt, 2016. [p20]

J. d. D. Ortuzar and L. G. Willumsen. *Modelling Transport*. John Wiley & Sons, 2011. [p7, 8]

M. Padgham, R. Lovelace, M. Salmon, and B. Rudis. Osmdata. *The Journal of Open Source Software*, 2 (14), 2017. URL https://doi.org/10.21105/joss.00305. [p21]

E. Pebesma, R. Bivand, and P. Ribeiro. Software for spatial statistics. *Journal of Statistical Software, Articles*, 63(1):1–8, 2015. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v063.i01. [p8]

M. Saidallah, A. El Fergougui, and A. E. Elalaoui. A comparative study of urban road traffic simulators. *MATEC Web Conf.*, 81:05002, 2016. URL https://doi.org/10.1051/matecconf/20168105002. [p21]

F. Simini, M. C. González, A. Maritan, and A.-L. Barabási. A universal model for mobility and migration patterns. *Nature*, pages 8–12, 2012. URL https://doi.org/10.1038/nature10856. [p18]

P. Waddell. UrbanSim: Modeling urban development for land use, transportation, and environmental planning. *Journal of the American Planning Association*, 68(3):297–314, 2002. [p8]

K. Walker. tigris: An R Package to Access and Work with Geographic Data from the US Census Bureau. *The R Journal*, 8(2):231–242, 2016. URL https://journal.r-project.org/archive/2016/RJ-2016-043/index.html. [p9]

T. Wieland. Market Area Analysis for Retail and Service Locations with MCI. *The R Journal*, 9(1): 298–323, 2017. URL https://journal.r-project.org/archive/2017/RJ-2017-020/index.html. [p21]

X. Zheng, W. Chen, P. Wang, D. Shen, S. Chen, X. Wang, Q. Zhang, and L. Yang. Big data for social transportation. *IEEE Transactions on Intelligent Transportation Systems*, 17(3):620–630, 2016. URL http://ieeexplore.ieee.org/abstract/document/7359138/. [p7]

Robin Lovelace
University of Leeds
Leeds Institute for Transport Studies
Leeds Institute for Data Analytics
34-40 University Road
LS2 9JT, UK
r.lovelace@leeds.ac.uk

Richard Ellison
University of Sydney
378 Abercrombie Street
Darlington, NSW 2008, Australia
richard.ellison@sydney.edu.au

The utiml Package: Multi-label Classification in R

by Adriano Rivolli and Andre C. P. L. F. de Carvalho

Abstract Learning classification tasks in which each instance is associated with one or more labels are known as multi-label learning. The implementation of multi-label algorithms, performed by different researchers, have several specificities, like input/output format, different internal functions, distinct programming language, to mention just some of them. As a result, current machine learning tools include only a small subset of multi-label decomposition strategies. The **utiml** package is a framework for the application of classification algorithms to multi-label data. Like the well known MULAN used with Weka, it provides a set of multi-label procedures such as sampling methods, transformation strategies, threshold functions, pre-processing techniques and evaluation metrics. The package was designed to allow users to easily perform complete multi-label classification experiments in the R environment. This paper describes the **utiml** API and illustrates its use in different multi-label classification scenarios.

Introduction

Multi-label classification (MLC) is a classification task where an instance can be simultaneously classified in more than one of the existing classes. Labeled data extracted from several domains, like text, web pages, multimedia (audio, image, videos), and biology are intrinsically multi-labeled. Additionally, the number of application domains with MLC data is growing fast.

Many current, real-world data science applications are MLC by nature. They are problems from very diverse domains, like labeling newspaper articles by subject and classification of proteins according to their functions. MLC algorithms have been successfully used in these and other MLC tasks (Diplaris et al., 2005). In a recent application, MLC algorithms were used to recommend food truck cuisines (Rivolli et al., 2017), assuming that a person can have more than one cuisine preference, and with the same level of preference.

Despite its growing relevance, there is a lack of comprehensive and easy to use tools for the R environment. A tool frequently used in MLC experiments is MULAN (Tsoumakas et al., 2011), which is a Java library built on top of Weka (Hall et al., 2009) to allow Weka users to deal with MLC data. Its popularity in the research community can be attributed to its ease of use, its large number and variation of its functionalities. The MLC alternative to Python users is the *scikit-multilearn* (Szymański, 2017), which provides a set of MLC algorithms and an interface for the MULAN library. Although other simpler tools, like MEKA (Read et al., 2016) and general data mining software (Gibaja and Ventura, 2015) include good functionalities to deal with MLC tasks, they address few MLC features and are not available in R.

It is important to mention that there are packages that offer some level of support for MLC in R. The most complete is the **mldr** package, an exploratory tool for the manipulation and analysis of MLC datasets (Charte and Charte, 2015). Although it does not contain MLC strategies, it supports the ARFF variation for MLC data, largely used for data mining and machine learning (ML) experiments, and has useful features, such as dataset characterization, MLC evaluation measures, and a rich user interface for the data exploration.

Some works use the **mlr** package, which was not specifically designed for MLC. As a result, it provides only a few multi-label strategies (Probst et al., 2017) and does not support the MLC ARFF format. In fact, it is a general purpose package, with an interface to more than one hundred algorithms that supports several ML tasks (Bischl et al., 2016). Another related package, **MLPUGS**, is a simple MLC package that contains only the implementation of the classifier chains (CC) strategy (Read et al., 2009).

Although the previous packages make it easier to perform some procedures related to MLC learning, their adoption in MLC experiments require more efforts from the developer/researcher than MULAN, available for Weka uses, which motivated the authors to design **utiml**, a more comprehensive, specific, easy to use, and extensible solution. The main features of the **utiml** package include:

- *Pre-processing techniques*: a set of techniques for the preparation and pre-possessing of MLC data to be used in experiments. These techniques deal with simple tasks, like removal of predictive attributes, instances and labels, replacement of nominal attribute values by numerical values, and data normalization.
- *Sampling*: a set of methods used to split MLC data through the holdout and k-fold methodologies.

Random or stratified strategies can be used for data partitioning.

- *Classification/Ranking*: the main MLC strategies. The transformation strategies support several base algorithms and the result can be seen as bipartition, probability/score, and ranking.
- *Threshold*: score-based and ranking-based threshold functions to be employed after the label prediction, so that bipartition values can be changed.
- *Evaluation*: traditional MLC evaluation measures and MLC confusion matrix for the summarization of classification result.

This paper describes the main aspects and resources of the **utiml** package. The current version is 0.1.4 and an updated list with all resources available will be maintained in the vignette document and the reference manual. The following section provides a brief review of MLC learning. Next, the package API is detailed, its resources are presented and some illustrative examples are provided. Finally, the main issues regarding the package use are highlighted in the summary section.

Multi-label classification learning

MLC tasks have attracted a growing attention in the ML community (de Carvalho and Freitas, 2009; Tsoumakas et al., 2010; Gibaja and Ventura, 2014). While in multi-class classification only a single class label is predicted, in MLC, more than one class label can be simultaneously predicted. In the same way as multi-class classification tasks can be seen as a generalization of binary classification tasks, which restricts to two the number of classes, MLC can be seen as a generalization of multi-class, which restricts to one the number of predicted classes (de Carvalho and Freitas, 2009). The main MLC approaches are *prediction of multiple labels*, *label ranking*, and *multi-label ranking* (Tsoumakas et al., 2010).

Multi-Label Classification (MLC), the most common task (Tsoumakas et al., 2010), induces a predictive model $h(x) \rightarrow Y$ from a set of training data, which later assigns one or more labels to each new example. This task can be formally defined as: let D be a set of labeled instances E, such that $D = \{E_1, E_2, ..., E_n\}$. Every labeled instance $E_i = (x_i, Y_i)$ is composed of $x_i = (x_{i1}, x_{i2}, ..., x_{id})$, which describes its position in a $\mathbb{R}^d$ input space, and $Y_i \subseteq L \mid L = \{\lambda_1, \lambda_2, ..., \lambda_q\}$, which describes a position in a $\{0, 1\}^q$ output space.

The *Label Ranking* (LR) task can be characterized by a function $f(x, \lambda_i)$, which, for each class label, outputs a score value in the interval $[0.0, 1.0]$, indicating the relevance, confidence, or probability of instance x belonging to the class whose label is λ_i. The higher the score value, the better the ranking position. While MLC predicts bipartitions and LR predicts scores, *Multi-label Ranking* (MLR) generates both. Since MLC can be derived from the LR formulation (Gibaja and Ventura, 2015)[1], if a strategy can be used in the LR task, it can also be used in the two other tasks.

These models can be obtained by two approaches (Tsoumakas et al., 2010), *problem transformation* and *algorithm adaptation*. *Problem transformation* converts the original MLC task into a set of binary or multi-class classification subtasks. Afterwards, any classification algorithm, here called base algorithm, can be used to induce models for the subtasks. In the *algorithm adaptation* approach, the multi-label support is embedded into the algorithm structure. Thus, while transformation fits data to algorithms, adaptation fits algorithms to data (Zhang and Zhou, 2014).

The transformation approach can be performed in three different ways: *binary*, *pairwise*, and *powerset*. *Binary* transformation generates at least one dataset per label, as in the one-versus-all multiclass strategy. *Pairwise* transformation, instead, creates one dataset for each pair of labels, similarly to one-versus-one multiclass strategy. Finally, *powerset* is a multi-class transformation that uses labelsets as classes. The adaptation approach, on the other hand, modifies conventional ML algorithms, like Decision Tree Induction Algorithms (DT), K-Nearest Neighbors (KNN), Random Forest (RF), and Support Vector Machines (SVM).

Other steps required for the application of ML algorithms need to be adapted to deal with MLC tasks. For example, stratified sampling for MLC data must take into account multiple targets and the predictive performance evaluation must consider situations like partially correct results and ranking accuracy. A complete overview of the alternatives to deal with these issues can be seen in Zhang and Zhou (2014) and Gibaja and Ventura (2015).

[1]Where $h(x) = \{\lambda \mid f(x, \lambda) \geq \tau(x), \lambda \in L\}$, where $\tau(x)$ is a threshold function.

The utiml package

Handling multi-label classification data

The predictive performance of MLC tasks can be strongly affected by the use of data pre-processing techniques. For such, **utiml** uses the **mldr** package (Charte and Charte, 2015), which provides the support for data pre-processing. Moreover, when **utiml** is installed/loaded, the **mldr** package is automatically installed/loaded. Specially, it supports the MLC ARFF format, which has an additional XML file describing the label columns[2].

By default, the **mldr** package handles categorical data as "character", instead of "factor", which is not supported by the implementation of some traditional machine learning algorithms available in R, like Random Forest from the **randomForest** package. To address this limitation, the mldata function converts all text columns to factors of an "mldr" dataset. For example, the function mldata should be used to load the 'flags' dataset[3], contains categorical attributes, like

```
> flags <- mldata(mldr("flags"))
```

After a dataset is loaded, pre-processing techniques can be applied to it. Table 1 shows the pre-processing techniques available in the **utiml** package. All these functions receive an "mldr" dataset as argument and return a pre-processed version of this dataset.

Pre-processing function	Description
`fill_sparse_mldata(mdata)`	Exchanges the NA values present in the dataset to 0 or "", according to the attribute type.
`normalize_mldata(mdata)`	Re-scales all numerical attribute values to values between 0 and 1 according to the min-max transformation. The lowest value is modified to 0.0 and the highest value is converted to 1.0.
`remove_attributes(mdata, attributes)`	Removes the specified attributes from the dataset.
`remove_labels(mdata, labels)`	Removes the specified labels from the dataset.
`remove_unique_attributes(mdata)`	Removes from the dataset attributes whose values are the same for all instances.
`remove_unlabeled_instances(mdata)`	Removes from the dataset instances without class labels.
`remove_skewness_labels(mdata, t)`	Removes from the dataset highly infrequent or highly frequent labels, according to a specific threshold value. The threshold t indicates the minimum number of positive and negative instances associated with each label.
`replace_nominal_attributes(mdata)`	Replaces categorical attributes by binary attributes. An attribute with n different values will be mapped to $n-1$ new columns containing binary values.

Table 1: Pre-processing techniques available in the **utiml** package

The **utiml** package also supports the main methodologies for data sampling, as shown in Table 2. The holdout and k-fold sampling can partition a dataset randomly and in a stratified way. They are selected by a parameter named method, which determines the sampling algorithm that creates the partitions. According to Sechidis et al. (2011), the accepted values are "random", "iterative", and "stratified", where the latter two are different stratification options. The "iterative" process stratifies a MLC dataset considering each label independently, while "stratified" is based on the different combinations of labels, also known as labelset.

These techniques were designed to improve the **mldr** package and to simplify the data preparation for the learning step. Concerning the analysis of the MLC data, **utiml** does not provide additional resources in this current version. However, it is possible to use the **mldr** package, which enables the understanding and exploration of several data aspects through an interactive interface (Charte and Charte, 2015).

[2]The complete specification is available at http://mulan.sourceforge.net/format.html.
[3]This dataset, available at http://mulan.sourceforge.net/datasets-mlc.html.

Sampling function	Description
`create_holdout_partition(mdata, partitions, method)`	Splits the data into at least two distinct parts. The second parameter defines the name and size of the partitions and `method` defines the type of sampling.
`create_kfold_partition(mdata, k, method)`	Creates an object that contains the *k* distinct parts of the dataset using the `method` for splitting the folds. It should be used in combination with `partition_fold(object, fold)`, which provides the training and testing data to a specific `fold`. The parameter `object` is the result of `create_kfold_partition`.
`create_random_subset(mdata, instances, attributes, replacement)`	Creates a random subset of the dataset based on the proportion of `instances` and `attributes`. When `replacement=TRUE`, a same instance can appear one more time in the training data.
`create_subset(mdata, rows, cols)`	Creates a specific subset of the dataset based on the instances (`rows`) and attributes (`cols`) specified.

Table 2: Sampling functions available in the **utiml** package

The **utiml** package also provides two MLC datasets: `toyml`, a synthetic dataset generated by the Mldatagen tool (Tomás et al., 2014); and `foodtruck`, a dataset in which several food truck cuisines are mapped as labels (Rivolli et al., 2017).

Multi-label classification strategies

The classification strategies are the heart of the **utiml** package. Table 3 shows the strategies available in the current version of the package. Some of the implemented strategies, such as `brplus`, `ctrl`, `dbr`, `lift`, `prudent` and `rdbr` were not found by the authors in other tools.

Transformation strategies build the multi-label models by using a ML base algorithm. The `base.algorithm` parameter defines the base algorithm employed to create the internal models. Table 4 shows the ML algorithms currently supported by the package. Their use requires additional packages, as indicated in column *R function Called*[4]. For example, `"C5.0"` algorithm requires the **C50** package be installed. Only the `"MAJORITY"` and `"RANDOM"` algorithms require no additional packages.

The arguments of the transformation strategies follow the pattern:

1. `mdata`: an `"mldr"` dataset object.
2. `base.algorithm`: a base algorithm, as listed in Table 4.
3. *additional strategy parameters*: specific parameters for each strategy. While the BR strategy contains no additional parameters, the ensemble ECC receives 4 specific parameters, namely `m`, `subsample`, `attr.space` and `replacement`[5].
4. `...`: extra parameters used by the `base.algorithm` selected. As illustration, if `base.algorithm = "SVM"`, the extra parameters can be those defined in the `svm` function of the **e1071** package, such as `kernel`,`gamma`,`cost`, among others.
5. `cores`: number of cores used for the parallelization of the training phase. Note some classification strategies, as `lp`, ignore the parameter because the tasks can not be parallelized.
6. `seed`: a seed that ensures reproducibility. This is particularly important when the task is parallelized. In other words, if `cores = 1`, the `seed` effect is similar to that of `set.seed(seed)`. However, if the `cores` are higher than 1, the `set.seed(seed)` command will not guarantee the same result can be obtained, since the task will be performed in parallel.

[4]These packages are not installed together with **utiml**.
[4]The `J48` algorithm has no support for task parallelization.
[5]These parameters denote, respectively, number of models in the ensemble, proportion of instances, attributes, and a possible replacement of instances. The specific parameters of each strategy are reported in the reference manual.

Strategy function	Description	Approach[a]	Reference
`baseline`	Baseline	-	Metz et al. (2012)
`br`	Binary Relevance	`BR`	Tsoumakas et al. (2010)
`brplus`	BR+	`BR, STA`	Cherman et al. (2012)
`cc`	Classifier Chains	`BR, CC`	Read et al. (2009)
`clr`	Calibrated Label Ranking	`PW`	Brinker et al. (2006)
`ctrl`	ConTRolled Label correlation	`BR, ENS`	Li and Zhang (2014)
`dbr`	Dependent Binary Relevance	`BR, STA`	Montañes et al. (2014)
`ebr`	Ensemble of Binary Relevance	`BR, ENS`	Read et al. (2009)
`ecc`	Ensemble of Classifier Chains	`BR, CC, ENS`	Read et al. (2009)
`eps`	Ensemble of Pruned Set	`ENS, PS`	Read et al. (2008)
`homer`	Hierarchy Of Multi-label classifiER	`HIE`	Tsoumakas et al. (2008)
`lift`	Learning with Label specIfic FeaTures	`BR, CLU`	Zhang and Wu (2015)
`lp`	Label Powerset	`PS`	Tsoumakas and Katakis (2007)
`mbr`	Meta-BR, 2BR or stacking	`BR, STA`	Tsoumakas et al. (2009)
`mlknn`	Multi-label kNN	`AD`	Zhang and Zhou (2007)
`ns`	Nested Stacking	`BR, CC`	Senge et al. (2013)
`ppt`	Pruned Problem Transformation	`PS`	Read et al. (2008)
`prudent`	PRUned and confiDENT Stacking	`BR, STA`	Alali and Kubat (2015)
`ps`	Pruned Set	`PS`	Read (2008)
`rakel`	Random k-labelsets	`ENS, PS`	Tsoumakas and Vlahavas (2007)
`rdbr`	Recursive Dependent Binary Relevance	`BR, ENS, STA`	Rauber et al. (2014)
`rpc`	Ranking by Pairwise Comparison	`PW`	Hüllermeier et al. (2008)

[a] `AD` = Adaptation; `BR` = Binary transformation; `CC` = Chain of classifiers; `CLU` = Clustering based; `ENS` = Ensemble; `HIE` = Hierarchy; `PS` = Powerset transformation; `PW` = Pairwise transformation; `STA` = Stacking

Table 3: Strategies available in the **utiml** package

`base.algorithm` value	Description	R function Called
`"C5.0"`	C5.0 Decision Trees	`C50::C5.0`
`"CART"`	Classification and regression trees	`rpart::rpart`
`"KNN"`	K Nearest Neighbor	`kknn::kknn`
`"NB"`	Naive Bayes	`e1071::naiveBayes`
`"RF"`	Random Forest	`randomForest::randomForest`
`"SMO"`	Sequential Minimal Optimization	`RWeka::SMO`
`"SVM"`	Support Vector Machine	`e1071::svm`
`"XGB"`	eXtreme Gradient Boosting	`xgboost::xgboost`
`"MAJORITY"`	Majority class prediction	-
`"RANDOM"`	Random prediction	-

Table 4: Base algorithms available in the **utiml** package

After the creation of a MLC model, the model can be applied to new data through the S3 `predict` method. The arguments of `predict` are:

1. `object`: a multi-label classifier.
2. `newdata`: a `"matrix"`, `"data.frame"` or `"mldr"` object, containing the data to be classified.
3. *additional model parameters*: specific parameters for each model. For example, the vote scheme to be used in the ecc prediction function can be defined[6].
4. `probability`: a logical value that indicates if the prediction result should be probability/score or bipartition. If `TRUE`, a probability result is returned; otherwise, the bipartition is obtained. The result can be changed, as observed next.
5. `...`: extra parameters based on the `predict` method related to the `base.algorithm` selected.
6. `cores`: number of cores used for the parallelization of the prediction phase. Some models, like CC, ignore this parameter because the tasks cannot be parallelized.
7. `seed`: a seed that ensures reproducibility. This is particularly important when the task is parallelized and the base algorithm is not deterministic.

The prediction result is an `"mlresult"` type object and can be used directly as a matrix, where each column is a label and each row is an instance. To change the type of result to bipartition, probability/score or a ranking matrix, the functions `as.bipartition`, `as.probability`, and `as.ranking`, respectively, can be used.

[6]The specific parameters of each predict method are reported in the reference manual.

Multi-label post-processing

Threshold functions adjust the bipartition result according to the score/probabilities predicted by the predictive models. In MLC learning, these functions can be score-based or rank-based, depending on the type of data used to define the threshold values. A single threshold value for all labels is named global threshold, the use of one value per label is named label-wise, and the use of one value per instance is named instance-wise (Al-Otaibi et al., 2014).

Table 5 shows the threshold functions available in the **utiml** package. All of them receive a probability/score matrix or an `"mlresult"` object as input and return a new `"mlresult"` object with different bipartitions as output. The only exception is `scut_threshold`, which returns threshold values, instead of bipartitions, and should be combined with the `fixed_threshold` function. Additionally, the `subset_correction` can be used as a threshold function (Senge et al., 2013). It changes the bipartition based on the labelsets present in the training data and outputs only the known labelsets.

Threshold function	Description	Approach
`fixed_threshold(prediction, threshold)`	Applies a fixed global or label-wise threshold.	score-based
`lcard_threshold(prediction, cardinality)`	Applies an instance-wise threshold using the cardinality measure.	rank-based
`mcut_threshold(prediction)`	Applies an instance-wise threshold and selects the subset of labels of the highest interval between two sorted scores.	score-based
`pcut_threshold(prediction, ratio)`	Applies a global or label-wise threshold using the ratio value to define the proportion of instances that will be relevant.	score-based
`rcut_threshold(prediction, k)`	Applies an instance-wise threshold and defines the *k* labels with highest scores as relevant.	rank-based
`scut_threshold(prediction, expected, loss.function)`	Returns a label-wise threshold using a loss function that minimizes the difference between the value predicted and the expected prediction value.	score-based

Table 5: Threshold functions available in the **utiml** package

Finally, **utiml** also supports the evaluation of MLC models. The `multilabel_evaluate` and `multilabel_confusion_matrix` functions can be used during the evaluation. The first calculates the traditional evaluation measures also available in **mldr** and MULAN, whereas the second generates a multi-label confusion matrix (`"mlconfmat"` object) detailing labels and instances.

The `multilabel_evaluate` function receives an `"mlresult"` or an `"mlconfmat"` object and the desired evaluation measures. One or more measures, likewise one or more group of measures can be adopted. Figure 1 shows the `measures` values, currently supported. A complete review of MLC evaluation measures can be found in Zhang and Zhou (2014) and Gibaja and Ventura (2015). Moreover, if the hyperparameter `labels=TRUE`, the return will be a list that contains the multi-label and labels' results detailed.

Default options

The **utiml** package uses the `option` function to customize some default parameters. For example, the default base algorithm for all transformation strategies is `"SVM"`. The `utiml.base.algorithm` option can be used to change this parameter value. Table 6 shows the option's names, a brief description of each option parameter value, and their default value. The following code defines "Random Forest" as the default base algorithm and sets the default number of cores to 8, to illustrate the setting of the options.

```
> options(utiml.base.algorithm = "RF", utiml.cores=8)
```

The `utiml.empty.prediction` option defines whether the MLC strategies can predict no labels for one or more instances. Among the alternatives to avoid an empty prediction (Liu and Chen, 2015), **utiml** outputs the labels with the highest probability/score. It must be observed that this option may

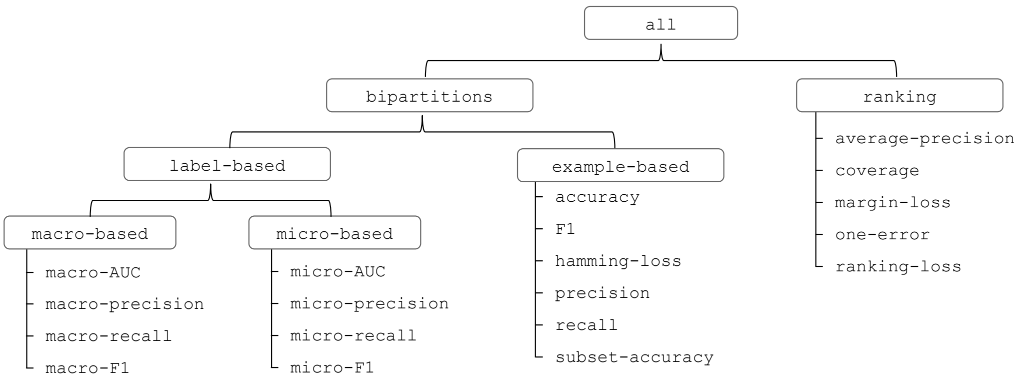


Figure 1: Groups of multi-label evaluation measures

Option parameter values	Description	Default value
`utiml.base.algorithm`	Default base algorithm used in the transformation strategies.	`"SVM"`
`utiml.cores`	Default number of cores used to parallelize the tasks.	`1`
`utiml.seed`	Default seed used by the MLC strategies.	`NA`
`utiml.use.probs`	Default type of the expected prediction results. If `TRUE`, the expected prediction is in the probability/score format, otherwise, a bipartition prediction is expected.	`TRUE`
`utiml.empty.prediction`	Default option concerning empty predictions. If `TRUE`, predictions can contain instances without labels, otherwise, the most important label is considered relevant and no empty predictions are obtained.	`FALSE`

Table 6: Options used by the **utiml** package

directly interfere with the result of the bipartition evaluation measures. Thus, it must be set according to the characteristics of the experiment being carried out.

How to use the package for multi-label classification experiments

The `toyml` dataset was used in the examples illustrated in this section. As `toyml` has two irrelevant attributes (`"iatt8"` and `"iatt9"`) and one redundant (`"ratt10"`) attribute, the pre-processing `remove_attributes` function can be applied to remove them.

```
> new.toyml <- remove_attributes(toyml, c("iatt8", "iatt9", "ratt10"))

> pre.process <- function (mdata) {
+   aux <- remove_skewness_labels(mdata, 5)  # Remove infrequent labels (less than 5)
+   aux <- remove_unlabeled_instances(aux)   # Remove instances without labels
+   aux <- remove_unique_attributes(aux)     # Remove constant attributes
+   return(mdata)
+ }
```

As `toyml` is already normalized and the dataset has a small number of instances, no other pre-processing technique is required. Thus, the `pre.process` function has no effect if applied in this case. For other datasets, the same procedure can be useful for their preparation for MLC experiments. Two scenarios that illustrate the use of **utiml** for the development of MLC experiments are presented next. Finally, a simple experimental analysis is performed using the `foodtruck` dataset, illustrating the use of the package in a more realist scenario.

Training and test experiment example using holdout

This example shows a MLC experiment using holdout, in which 70% of the dataset instances are used for training and 30% for test. A BR model that uses "Random Forest" as a base algorithm is induced and applied to the test instances. Next, predictions are assessed using MLC evaluation measures.

```
> set.seed(123)
> ds <- create_holdout_partition(new.toyml, c(train=0.7, test=0.3))
> model <- br(ds$train, "RF")
> predictions <- predict(model, ds$test)

> results <- multilabel_evaluate(ds$test, predictions, c("example-based", "macro-F1"))
> round(results, 4)
## accuracy      F1  hamming-loss  macro-F1  precision  recall  subset-accuracy
##   0.6444  0.7411        0.1933    0.4015     0.7833  0.7722           0.3000
```

A MLC baseline can be included among the strategies being experimentally compared. In the following code, the general baseline (Metz et al., 2012) is induced. A subtle difference is observed in `"hamming-loss"` and `"F1"` measures in favor of the BR model. The small number of labels are due to very common combinations of them found in `toyml` favors the general baseline.

```
> base.preds <- predict(baseline(ds$train, "general"), ds$test)
> base.res <- multilabel_evaluate(ds$test, base.preds, c("hamming-loss", "F1"))

> round(base.res, 4)
##     F1  hamming-loss
## 0.7311        0.2000
```

In both examples, the test set predictions compose an `"mlresult"` object and, by default, show the score/probability produced by the base algorithm. For example:

```
> head(predictions)
##       y1    y2    y3    y4    y5
## 23 0.336 0.640 0.178 0.922 0.122
## 19 0.106 0.860 0.366 0.670 0.280
## 62 0.094 0.776 0.526 0.688 0.090
## 1  0.196 0.618 0.204 0.758 0.162
## 67 0.090 0.964 0.210 0.700 0.234
## 92 0.060 0.856 0.162 0.598 0.454
```

The `as.bipartition` and `as.ranking` functions can be used to change, respectively, the probability/score to a bipartition matrix or the raking values, as illustrated next. Optionally, a threshold function can be applied to change the bipartitions.

```
> head(as.bipartition(predictions))
##    y1 y2 y3 y4 y5
## 23  0  1  0  1  0
## 19  0  1  0  1  0
## 62  0  1  1  1  0
## 1   0  1  0  1  0
## 67  0  1  0  1  0
## 92  0  1  0  1  0

> head(as.ranking(predictions))
##    y1 y2 y3 y4 y5
## 23  3  2  4  1  5
## 19  5  1  3  2  4
## 62  4  1  3  2  5
## 1   4  2  3  1  5
## 67  5  1  4  2  3
## 92  5  1  4  2  3
```

```
> head(mcut_threshold(predictions))
##    y1 y2 y3 y4 y5
## 23  0  1  0  1  0
## 19  0  1  0  1  0
## 62  0  1  1  1  0
## 1   0  1  0  1  0
## 67  0  1  0  1  0
## 92  0  1  0  1  1
```

Three different ECC models are created in the following code to illustrate the use of different parameters and base algorithms. Each model uses a specific base algorithm and configuration, which, consequently, results in different models for the same data and MLC strategy.

```
> # Using KNN with k = 5 and changing ECC parameters
> model1 <- ecc(ds$train, "KNN", m=7, subsample=0.8, k=5)

> # Using C5.0 and changing ECC parameters
> model2 <- ecc(ds$train, "C5.0", subsample=0.6, attr.space=1)

> # Using SVM with cost = 10 and gamma = 0.5 and default ECC parameters
> model3 <- ecc(ds$train, "SVM", cost=10, gamma=0.5)
```

By default, the `create_holdout_partition` function creates two random partitions (train and test) with 70% and 30% of the dataset instances, respectively. The number of partitions, sizes and sampling method can be modified. The following code shows how to create three, label-stratified partitions, named `"train"`, `"test"`, and `"val"` with 70%, 20%, and 10% of the instances, respectively. The `"val"` partition can be used in a validation step for model selection or hyperparameter tuning.

```
> partitions <- c(train=0.7, test=0.2, val=0.1)
> strat <- create_holdout_partition(new.toyml, partitions, "iterative")
```

Training and test example experiment using k-fold cross validation

This section shows some examples of how to perform cross-validation MLC experiments. The `cv` method can be used to encapsulate the whole procedure, which simplifies the respective task, such that a 10-fold stratified cross-validation can be performed with few lines of code. For instance, the RAkEL strategy using the `"SVM"` base algorithm can be evaluated in the following way:

```
# Defining the evaluation measures
> measures <- c("hamming-loss", "subset-accuracy", "one-error")

# Running 10-fold cross validation
> results <- cv(new.toyml, method="rakel", base.algorith="SVM", cv.folds=10,
+               cv.sampling="stratified", cv.measures=measures, cv.seed=123)

> round(results, 4)
##   hamming-loss       one-error subset-accuracy
##          0.212           0.160           0.240
```

To obtain detailed results by folds and/or labels, the hyperparameter `cv.results=TRUE` can be set. In this case, a list is returned where the multi-label and labels' results can be obtained as illustrated in the next example.

```
> results <- cv(new.toyml, method="rakel", base.algorith="SVM", cv.results=TRUE,
+               cv.sampling="stratified", cv.measures=measures, cv.seed=123)

> t(results$multilabel)
##                 [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## hamming-loss     0.2 0.18 0.18  0.2 0.22 0.22 0.22  0.2 0.24  0.26
## one-error        0.3 0.20 0.20  0.2 0.10 0.00 0.00  0.2 0.20  0.20
## subset-accuracy  0.3 0.30 0.30  0.3 0.20 0.20 0.20  0.2 0.20  0.20

> round(sapply(results$labels, colMeans), 4)
##            y1   y2   y3     y4   y5
## accuracy 0.83 0.78 0.81 0.6900 0.83
## balacc   0.50 0.50 0.50 0.5054 0.50
## TP       0.00 7.80 0.00 6.8000 0.00
## TN       8.30 0.00 8.10 0.1000 8.30
## FP       0.00 2.20 0.00 3.0000 0.00
## FN       1.70 0.00 1.90 0.1000 1.70
```

Any MLC strategy can be used in the cv method, as well as specific hyperparameters for them. Additionally, the procedure can be parallelized, using `cv.cores`. The next example shows the ECC algorithm with specific hyperparameters being executed using 5 folds and parallelized in 4 cores.

```
> results <- cv(new.toyml, method="ecc", base.algorith="RF", subsample=0.9,
+               attr.space=0.9, cv.folds=5, cv.cores=4)
```

Finally, to perform a cross-validation procedure manually, the methods `create_kfold_partition` and `partition_fold`, can be used to create the folds and obtain the train and test dataset for each one of them, respectively. A good example, is the code used in the cv method, such that

```
...
> cvdata <- create_kfold_partition(mdata, cv.folds, cv.sampling)
> results <- parallel::mclapply(seq(cv.folds), function (k){
+    ds <- partition_fold(cvdata, k)
+    model <- do.call(method, c(list(mdata=ds$train), ...))
+    pred <- predict(model, ds$test, ...)
+    multilabel_evaluate(ds$test, pred, cv.measures, labels=TRUE)
+ }, mc.cores=cv.cores)
...
```

Experiments with the food truck dataset

In order to show how the package can be used in a real world problem, this section illustrates the use of the **utiml** to perform an exploratory analysis of the food truck dataset (Rivolli et al., 2017). First, the `br` strategy is evaluated with different ML base algorithms ("C5.0", "RF", "SVM" and "XGB") to identify the algorithm that produces the best macro and micro-F1 results.

```
> measures <- c("macro-F1", "micro-F1")
> algorithms <- c("C5.0", "RF", "SVM", "XGB")

> res <- sapply(algorithms, function(alg) {
+   cv(foodtruck, "br", base.algorithm=alg, cv.measures=measures, cv.seed=1)
+ })

> round(res, 4)
##            C5.0     RF    SVM    XGB
## macro-F1 0.1764 0.1824 0.1188 0.1827
## micro-F1 0.4856 0.5340 0.4835 0.5130
```

Regarding the macro-F1 measure, XGB presented the best result, however the difference observed between RF and C5.0 was small. For micro-F1, RF obtained the best result, followed closely by XGB. The differences observed between the macro and micro measures, independently of the base algorithm, may indicate that some infrequent labels had a poor F1 performance. To analyze this hypothesis, `br` was run again with RF, which obtained the best performance in the previous cross-validation procedure, with a new data subset. In the confusion matrix for the induced model, some patterns can be observed.

The following code shows that six labels (`mexican_food`, `chinese_food`, `japanese_food`, `arabic_food`, `healthy_food` and `fitness_food`) had no True Positive (TP) and False Positive (FP) predictions. Thus, for these labels, all instances were predicted as negative. This explains the difference observed between the macro and micro-F1 result, since the macro-F1 is the average labels' F1, which is 0 for these labels.

```
> set.seed(1)
> ds <- create_holdout_partition(foodtruck, method="iterative")

> model <- br(ds$train, "RF")
> pred <- predict(model, ds$test)

> cm <- multilabel_confusion_matrix(ds$test, pred)
> as.matrix(cm)
##                 TP  TN FP FN
## street_food     88   5 32  0
## gourmet         13  81  8 23
## italian_food     1 113  0 11
## brazilian_food   3 104  0 18
## mexican_food     0 113  0 12
## chinese_food     0 121  0  4
## japanese_food    0 115  0 10
## arabic_food      0 118  0  7
## snacks           7 103  2 13
## healthy_food     0 116  0  9
## fitness_food     0 116  0  9
## sweets_desserts 11  66 13 35
```

It must be observed that the cm object is a list containing several information about the prediction, like the confusion matrix values summarized by instances and labels. Any evaluation measure can be computed using only the information provided by this object. As an example, the next code summarizes the proportion of instances and the number of labels correctly predicted in the previous example. The results show that the BR model was not able to predict a correct label for almost 20% of the test instances; around 65% of the instances were correctly predicted with a single label; 12% were correctly predicted with 2 labels; and 3% were correctly predicted with 3 labels.

```
> prop.table(table(cm$TPi))
##     0     1     2     3
## 0.200 0.648 0.120 0.032
```

These results show a researcher can simulate new scenarios and explore different solutions in order to improve the predictive performance in a MLC task. The **utiml** package offers several resources that simplify the most basic and recurrent procedures adopted in the MLC domain.

Summary

Data classification is one of the main ML tasks. Although ML classification algorithms are usually designed and employed for single label classification tasks, in several application domains, an instance can have more than one class label. This paper introduced the **utiml** package, which provides several functions for MLC experiments in R. Similarly to MULAN, one of the most popular MLC tools, **utiml** offers a wide set of functionalities. The provided functions implement procedures that cover several MLC-related tasks, which include data pre-processing, data sampling model induction, optimization and evaluation of MLC models. The package **utiml** also supports the intrinsic parallelization of tasks and allows the reproducibility of MLC experiments.

To the best of the authors knowledge, some of the features present in **utiml** are not available in any other R tool, such as the implementation of MLC stratification (Sechidis et al., 2011), baselines (Metz et al., 2012), thresholds (Al-Otaibi et al., 2014) and an option that allow the users to avoid the empty prediction problem (Liu and Chen, 2015). Moreover, as in MULAN which enables users to take advantage of the resources available in the Weka environment, **utiml** users can benefit from the several libraries available in R.

The most important limitation of this package is that some common MLC procedures, like feature selection, imbalanced data, and classification strategies based on the algorithm adaptation approach are not available yet. They will be implemented in the future as a natural progression of this work and will be included in the next versions of the **utiml** package. The authors encourage other developers to integrate their own algorithms in the **utiml** package[7], so that it becomes a more robust and complete MLC package.

Bibliography

R. Al-Otaibi, P. Flach, and M. Kull. Multi-Label Classification: A Comparative Study on Threshold Selection Methods. In *First International Workshop on Learning over Multiple Contexts (LMCE) at ECML-PKDD 2014*, Nancy, France, 2014. [p29, 34]

[7]The source and project details are available on the https://github.com/rivolli/utiml page

A. Alali and M. Kubat. PruDent: A Pruned and Confident Stacking Approach for Multi-Label Classification. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2480–2493, 2015. URL https://doi.org/10.1109/tkde.2015.2416731. [p28]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine Learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL http://jmlr.org/papers/v17/15-066.html. [p24]

K. Brinker, J. Fürnkranz, and E. Hüllermeier. A unified model for multilabel classification and ranking. In *Proceedings of the 2006 Conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva Del Garda, Italy*, pages 489–493, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press. ISBN 1-58603-642-4. URL http://dl.acm.org/citation.cfm?id=1567016.1567123. [p28]

F. Charte and F. D. Charte. Working with Multilabel Datasets in R: The mldr Package. *The R Journal*, 7 (2):149–162, 2015. [p24, 26]

E. A. Cherman, J. Metz, and M. C. Monard. Incorporating Label Dependency into the Binary Relevance Framework for Multi-Label Classification. *Expert Systems with Applications*, 39(2):1647–1655, 2012. URL https://doi.org/10.1016/j.eswa.2011.06.056. [p28]

A. C. P. L. F. de Carvalho and A. A. Freitas. *A Tutorial on Multi-Label Classification Techniques*, chapter 8, pages 177–195. Springer-Verlag, Berlin, Heidelberg, 2009. URL https://doi.org/10.1007/978-3-642-01536-6_8. [p25]

S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas. Protein Classification with Multiple Algorithms. In *Proceedings of the 10th Panhellenic Conference on Informatics*, pages 448–456, Volos, Greece, 2005. [p24]

E. Gibaja and S. Ventura. Multi-Label Learning: a Review of the State of the Art and Ongoing Research. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(6):411–444, 2014. URL https://doi.org/10.1002/widm.1139. [p25]

E. Gibaja and S. Ventura. A Tutorial on Multilabel Learning. *ACM Computing Surveys*, 47(3):1–38, 2015. URL https://doi.org/10.1145/2716262. [p24, 25, 29]

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009. URL https://doi.org/10.1145/1656274.1656278. [p24]

E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker. Label Ranking by Learning Pairwise Preferences. *Artificial Intelligence*, 172(16-17):1897–1916, 2008. URL https://doi.org/10.1016/j.artint.2008.08.002. [p28]

Y. Li and M. Zhang. Enhancing Binary Relevance for Multi-Label Learning with Controlled Label Correlations Exploitation. In *Proceedings of the 13th Pacific Rim International Conference on Artificial Intelligence*, pages 91–103, Gold Coast, Australia, 2014. Springer-Verlag. URL https://doi.org/10.1007/978-3-319-13560-1_8. [p28]

S. M. Liu and J.-H. Chen. An Empirical Study of Empty Prediction of Multi-Label Classification. *Expert Systems with Applications*, 42(13):5567–5579, 2015. URL https://doi.org/10.1016/j.eswa.2015.01.024. [p29, 34]

J. Metz, L. F. de Abreu, E. A. Cherman, and M. C. Monard. On the Estimation of Predictive Evaluation Measure Baselines for Multi-Label Learning. In *13th Ibero-American Conference on AI*, pages 189–198, Cartagena de Indias, Colombia, 2012. URL https://doi.org/10.1007/978-3-642-34654-5_20. [p28, 31, 34]

E. Montañes, R. Senge, J. Barranquero, J. R. Quevedo, J. J. del Coz, and E. Hüllermeier. Dependent Binary Relevance Models for Multi-Label Classification. *Pattern Recognition*, 47(3):1494–1508, 2014. URL https://doi.org/10.1016/j.patcog.2013.09.029. [p28]

P. Probst, Q. Au, G. Casalicchio, C. Stachl, and B. Bischl. Multilabel Classification with R Package mlr. *The R Journal*, 9(1):352–369, 2017. [p24]

T. W. Rauber, L. H. Mello, V. F. Rocha, D. Luchi, and F. M. Varejão. Recursive Dependent Binary Relevance Model for Multi-Label Classification. In *Proceedings of the 14th Ibero-American Conference on AI*, pages 206–217, Santiago de Chile, Chile, 2014. Springer-Verlag. URL https://doi.org/10.1007/978-3-319-12027-0. [p28]

J. Read. A Pruned Problem Transformation Method for Multi-Label Classification. In *Proceedings of the New Zealand Computer Science Research Student Conference*, pages 143–150, 2008. [p28]

J. Read, B. Pfahringer, and G. Holmes. Multi-Label Classification Using Ensembles of Pruned Sets. In *Proceedings of the IEEE International Conference on Data Mining*, pages 995–1000, Pisa, Italy, 2008. IEEE. URL `https://doi.org/10.1109/icdm.2008.74`. [p28]

J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier Chains for Multi-Label Classification. In *Proceedings of the European Conference*, volume 5782, pages 254–269, Bled, Slovenia, 2009. URL `https://doi.org/10.1007/978-3-642-04174-7_17`. [p24, 28]

J. Read, P. Reutemann, B. Pfahringer, and G. Holmes. MEKA: A Multi-Label / Multi-Target Extension to WEKA. *Journal of Machine Learning Reasearch*, 17:1–5, 2016. URL `http://jmlr.org/papers/v17/12-164.html`. [p24]

A. Rivolli, L. C. Parker, and A. C. P. L. F. de Carvalho. Food truck recommendation using multi-label classification. In E. Oliveira, J. Gama, Z. Vale, and H. Lopes Cardoso, editors, *Progress in Artificial Intelligence*, pages 585–596. Springer-Verlag, 2017. [p24, 27, 33]

K. Sechidis, G. Tsoumakas, and I. Vlahavas. On the Stratification of Multi-Label Data. In *Lecture Notes in Computer Science*, volume 6913 LNAI, pages 145–158, 2011. URL `https://doi.org/10.1007/978-3-642-23808-6_10`. [p26, 34]

R. Senge, J. J. del Coz, and E. Hüllermeier. Rectifying Classifier Chains for Multi-Label Classification. In *Workshop of Lernen, Wissen & Adaptivität (LWA 2013)*, pages 162–169, Bamberg, Germany, 2013. [p28, 29]

P. Szymański. A scikit-based python environment for performing multi-label classification. *arXiv preprint arXiv:1702.01460*, 2017. [p24]

J. T. Tomás, N. Spolaôr, E. A. Cherman, and M. C. Monard. A Framework to Generate Synthetic Multi-Label Datasets. In *Proceedings of the XXXIX Latin American Computing Conference (CLEI 2013)*, volume 302, pages 155–176, Naiguata, Venezuela, 2014. URL `https://doi.org/10.1016/j.entcs.2014.01.025`. [p27]

G. Tsoumakas and I. Katakis. Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007. URL `https://doi.org/10.4018/jdwm.2007070101`. [p28]

G. Tsoumakas and I. Vlahavas. Random k -Labelsets: An Ensemble Method for Multilabel Classification. In *Proceedings of the European Conference on Machine Learning*, pages 406–417, Warsaw, Poland, 2007. URL `https://doi.org/10.1007/978-3-540-74958-5_38`. [p28]

G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and Efficient Multilabel Classification in Domains with Large Number of Labels. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery, Workshop on Mining Multidimensional Data*, pages 30–44, Antwerp, Belgium, 2008. [p28]

G. Tsoumakas, E. Loza Mencía, I. Katakis, S.-H. Park, and J. Fürnkranz. On the Combination of Two Decompositive Multi-Label Classification Methods. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery, Workshop on Preference Learning*, pages 114–129, Bled, Slovenia, 2009. [p28]

G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining Multi-Label Data. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, chapter 34, pages 667–685. Springer-Verlag, 2 edition, 2010. ISBN 0387244352. URL `https://doi.org/10.1007/978-0-387-09823-4_34`. [p25, 28]

G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. MULAN: A Java Library for Multi-Label Learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011. [p24]

M.-L. Zhang and L. Wu. Lift: Multi-Label Learning with Label-Specific Features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(1):107–120, 2015. URL `https://doi.org/10.1109/tpami.2014.2339815`. [p28]

M.-L. Zhang and Z.-H. Zhou. A Review on Multi-Label Learning Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, 2014. ISSN 1041-4347. URL `https://doi.org/10.1109/tkde.2013.39`. [p25, 29]

M.-L. L. Zhang and Z.-H. H. Zhou. ML-KNN: A Lazy Learning Approach to Multi-Label Learning. *Pattern Recognition*, 40(7):2038–2048, 2007. URL https://doi.org/10.1016/j.patcog.2006.12.019. [p28]

Adriano Rivolli
Federal University of Technology - Parana (UTFPR)
Cornélio Procópio - PR
Brazil
rivolli@utfpr.edu.br

Andre C. P. L. F. de Carvalho
Institute of Mathematical and Computer Sciences (ICMC)
University of São Paulo (USP)
São Carlos - SP
Brazil
andre@icmc.usp.br

rcss: R package for optimal convex stochastic switching

by Juri Hinz and Jeremy Yee

Abstract The R package **rcss** provides users with a tool to approximate the value functions in the Bellman recursion under certain assumptions that guarantee desirable convergence properties. This R package represents the first software implementation of these methods using matrices and nearest neighbors. This package also employs a pathwise dynamic method to gauge the quality of these value function approximations. Statistical analysis can be performed on the results to obtain other useful practical insights. This paper describes **rcss** version 1.6.

Motivation

Sequential decision making is usually addressed under the framework of discrete-time stochastic control. The theory of *Markov Decision Processes/Dynamic Programming* provides a variety of methods to deal with such questions. In generic situations, approaching analytical solutions for even some of the simplest decision processes may be a cumbersome process (Powell (2007), Bauerle and Rieder (2011), Pham (2009)). The use of numerical approximations may often be far more practical given the rapid improvements in available computational power. The ability to gauge the quality of these approximations is also of significant importance. This paper will describe the implementation of fast and accurate algorithms to address problems with a finite time setting, finite action set, and convex reward functions whose state processes follow linear dynamics.

Although these assumptions seem restrictive, a large class of practically important control problems can be solved using our methods. For instance, approximating a compact action set by a sufficiently dense finite grid of representative actions, an approximate solution with excellent precision and numerical performance can be obtained. This technique is illustrated by battery storage control (Hinz and Yee (2017a) in the present contribution. Furthermore, various extension to non-linear state dynamics are possible. By using appropriate transformations to linearize the state dynamics, our methodology becomes applicable to control partially observable systems (Hinz and Yee (2017b)).

This R package implements algorithms to approximate the value functions in finite horizon Markov decision processes and also to gauge the quality of these approximations. Under the conditions discussed in Hinz (2014), these value functions converge to the true unknown value functions *uniformly on compact sets*. Defined by the maxima half-norms over compact sets, the corresponding topology ensures that the decision rules, inferred from value function approximations, converge pointwise to the optimal decision policy.

This paper points any interested readers to the technical details contained in Hinz (2014). The package **rcss** represents the first software implementation of these algorithms and has already been used to address problems such as pricing financial options (Hinz and Yap (2015)), natural resource extraction (Hinz et al. (In Press)), battery management (Hinz and Yee (2017a)), and optimal asset allocation under hidden state dynamics (Hinz and Yee (2017b)). One of the major benefits of implementing these methods within R (R Core Team (2013)) is that the results can be analysed using the vast number of statistical tools available in this language.

This R package focuses on linear state dynamics for the continuous state component. While non-linear dynamics have been covered in Yee (Preprint), linear dynamics allow much of the computational effort to be reduced to a series of multiplications and additions of matrices. This is an attractive feature given the availability of well developed linear algebra libraries. Linear state dynamics also allow for a standardized problem formulation which is desirable for a software package. This R package also focuses on problems with finite action sets which occur frequently in many real world problems. While this may seem restrictive for other problems, note that action sets are often discretized in practice to allow for numerical tractability.

Problem setting

Suppose that state space $\mathbf{X} = \mathbf{P} \times \mathbf{Z}$ is the product of a finite set $\mathbf{P}$ and an open convex set $\mathbf{Z} \subseteq \mathbb{R}^d$. Furthermore, assume that a finite set $\mathbf{A}$ represents all possible actions. Given a finite time horizon $\{0, 1, \ldots, T\} \subset \mathbb{N}$, consider *a fully observable* controlled Markovian process $(X_t)_{t=0}^T := (P_t, Z_t)_{t=0}^T$ which consists of two parts. The discrete component $(P_t)_{t=0}^T$ describes the evolution of a finite-state controlled Markov chain which takes values in a finite set $\mathbf{P}$. Further assume that at any time

$t = 0, \ldots, T-1$ the controller takes an action $a \in \mathbf{A}$ from a finite set $\mathbf{A}$ of all admissible actions in order to cause the one-step transition from the mode $p \in \mathbf{P}$ to the mode $p' \in \mathbf{P}$ with a probability $\alpha_{p,p'}(a)$, where $(\alpha_{p,p'}(a))_{p,p' \in \mathbf{P}}$ are pre-specified stochastic matrices for all $a \in \mathbf{A}$.

Let us now turn to the evolution of the other component $(Z_t)_{t=0}^T$ of the state process $(X_t)_{t=0}^T$. Here, we assume that it follows an uncontrolled evolution in the space $\mathbf{Z}$ driven as

$$Z_{t+1} = W_{t+1} Z_t, \qquad t = 0, \ldots, T-1$$

by independent *disturbance matrices* $(W_t)_{t=1}^T$. That is, the transition kernels $\mathcal{K}_t^a$ governing the evolution of our controlled Markov process $(X_t)_{t=0}^T := (P_t, Z_t)_{t=0}^T$ from time t to $t+1$ are given for each $a \in \mathbf{A}$ by

$$\mathcal{K}_t^a v(p,z) = \sum_{p' \in \mathbf{P}} \alpha_{p,p'}(a) \mathbb{E}(v(p', W_{t+1} z)), \quad p \in \mathbf{P},\, z \in \mathbf{Z},\, t = 0, \ldots, T-1$$

which acts on each function $v : \mathbf{P} \times \mathbf{Z} \to \mathbb{R}$ where the above expectations are well-defined.

If the system is in the state (p,z), the costs of applying action $a \in \mathbf{A}$ at time $t = 0, \ldots, T-1$ are expressed through $r_t(p,z,a)$. Having arrived at time $t = T$ in the state (p,z), a final *scrap value* $r_T(p,z)$ is collected. Thereby the reward and scrap functions

$$r_t : \mathbf{P} \times \mathbf{Z} \times \mathbf{A} \to \mathbb{R}, \quad r_T : \mathbf{P} \times \mathbf{Z} \to \mathbb{R}$$

are exogenously given for $t = 0, \ldots, T-1$. At each time $t = 0, \ldots, T$, the *decision rule* π_t is given by a mapping $\pi_t : \mathbf{P} \times \mathbf{Z} \to \mathbf{A}$, prescribing at time t an action $\pi_t(p,z) \in \mathbf{A}$ for a given state $(p,z) \in \mathbf{P} \times \mathbf{Z}$. Note that each decision rule refers to the recent state of the system, representing a *feedback control*. A sequence $\pi = (\pi_t)_{t=0}^{T-1}$ of decision rules is called a *policy*. For each policy $\pi = (\pi_t)_{t=0}^{T-1}$, the so-called policy value $v_0^\pi(p_0, z_0)$ is defined as the total expected reward

$$v_0^\pi(p_0, z_0) = \mathbb{E}^{(p_0,z_0),\pi} \left[\sum_{t=0}^{T} r_t(P_t, Z_t, \pi_t(P_t, Z_t)) + r_t(P_t, Z_t) \right].$$

In this formula $\mathbb{E}^{(p_0,z_0),\pi}$ stands for the expectation with respect to the probability distribution of $(X_t)_{t=0}^T := (P_t, Z_t)_{t=0}^T$ defined by Markov transitions from (P_t, Z_t) to (P_{t+1}, Z_{t+1}), which are induced by the kernels $\mathcal{K}_t^{\pi_t(P_t,Z_t)}$ for $t = 0, \ldots, T-1$, started at the initial point $(P_0, Z_0) = (p_0, z_0)$.

Now we turn to the optimization goal. A policy $\pi^* = (\pi_t^*)_{t=0}^{T-1}$ is called optimal if it maximizes the total expected reward over all policies $\pi \mapsto v_0^\pi(p,z)$. To obtain such policy, one introduces for $t = 0, \ldots, T-1$ the so-called *Bellman operator*

$$\mathcal{T}_t v(p,z) = \max_{a \in \mathbf{A}} \left[r_t(p,z,a) + \sum_{p' \in \mathbf{P}} \alpha_{p,p'}(a) \mathbb{E}[v(p', W_{t+1} z)] \right] \tag{1}$$

for $(p,z) \in \mathbf{P} \times \mathbf{Z}$, acting on all functions v where the stochastic kernel is defined. Consider the *Bellman recursion*, also referred to as backward induction:

$$v_T = r_T, \quad v_t = \mathcal{T}_t v_{t+1} \qquad \text{for } t = T-1, \ldots, 0. \tag{2}$$

Having assumed that the reward functions are convex and globally Lipschitz and the disturbances W_{t+1} are integrable, there exists a unique recursive solution $(v_t^*)_{t=0}^T$ to the Bellman recursion. These functions $(v_t^*)_{t=0}^T$ are called *value functions* and they determine an optimal policy (possibly not unique) $\pi^* = (\pi_t^*)_{t=0}^T$ via

$$\pi_t^*(p,z) = \arg\max_{a \in \mathbf{A}} \left[r_t(p,z,a) + \sum_{p' \in \mathbf{P}} \alpha_{p,p'}(a) \mathbb{E}[v_{t+1}^*(p', W_{t+1} z)] \right], \tag{3}$$

for $t = T-1, \ldots, 0$.

The aim of the package **rcss** is to approximate the true value functions $(v_t^*)_{t=0}^{T-1}$ and the corresponding optimal policies $\pi^* = (\pi_t^*)_{t=0}^{T-1}$. Given these approximations, our package also determines their distance to optimality which will allow the user to decide whether it is within some acceptable margin or whether further fine tuning is needed to obtain even more accurate results.

Numerical approach

Since the reward and scrap functions are convex in the continuous variable, the value functions are also convex and can be approximated by piecewise linear and convex functions. For this, introduce the so-called subgradient envelope $\mathcal{S}_{\mathbf{G}^m} f$ of a convex function $f : \mathbf{Z} \to \mathbb{R}$ on a grid $\mathbf{G}^m \subset \mathbf{Z}$ with m points:

$$(\mathcal{S}_{\mathbf{G}^m} f)(z) = \max_{g \in \mathbf{G}^m} (\triangledown_g f)(z),$$

for $z \in \mathbf{Z}$ where $\triangledown_g f$ is the tangent of f at grid point $g \in \mathbf{G}^m$. Using the subgradient envelope operator, define the double-modified Bellman operator as

$$\mathcal{T}_t^{m,n} v(p,z) = \mathcal{S}_{\mathbf{G}^m} \max_{a \in \mathbf{A}} \left(r_t(p,z,a) + \sum_{p' \in \mathbf{P}} \alpha^a_{p,p'} \sum_{k=1}^{n} \nu^{(k)}_{t+1} v(p', W^{(k)}_{t+1} z) \right),$$

where the probability weights $(\nu^{(k)}_{t+1})^n_{k=1}$ correspond to the distribution sampling $(W^{(k)}_{t+1})^n_{k=1}$ of each disturbance W_{t+1}. The corresponding backward induction

$$v_T^{m,n}(p,z) \quad = \quad \mathcal{S}_{\mathbf{G}^m} r_T(p,z), \qquad (4)$$
$$v_t^{m,n}(p,z) \quad = \quad \mathcal{T}_t^{m,n} v_{t+1}^{m,n}(p,z), \qquad t = T-1, \ldots 0, \qquad (5)$$

yields the so-called double-modified value functions $(v_t^{m,n})^T_{t=0}$. Under appropriate assumptions regarding grid density and the disturbance sampling, the double-modified value functions converge uniformly to the true value functions on compact sets (see Hinz (2014)). To gauge the quality of the above approximations, we construct two random variables whose expectations bound the true value function i.e.

$$\mathbb{E}(\underline{v}_0(p,z_0)) \le v_0(p,z_0) \le \mathbb{E}(\overline{v}_0(p,z_0)), \qquad p_0 \in \mathbf{P}, \quad z_0 \in \mathbf{Z}. \qquad (6)$$

We refer any interested readers to Hinz and Yap (2015) for the technical details. This process exhibits a helpful *self-tuning* property. The closer the value function approximations resemble their true unknown counterparts, the tighter the bounds in Equation 6 and the lower the standard errors of the bound estimates. The R package **rcss** represents these piece-wise linear functions as matrices and utilises nearest neighbor algorithms (from Mangenat and Jefferies (2018)) to reduce the computational effort. Most of the computational work is done in C++ via **Rcpp** (Eddelbuettel and Francois (2011)) and is parallelized using **OpenMp** (Dagum and Menon (1998)). The following sections give some code demonstrations.

Example: Bermuda put

Stochastic control problems of optimal switching type naturally arise in the framework of valuation of financial contracts. A simple example is given by the Bermudan put option. This option gives its owner the right but not an obligation to choose a time to exercise the option in a finite number of time points before the maturity of the option in order to receive a payment which depends on the price of the underlying asset at the exercise time. The so-called fair price of the Bermudan option is related to the solution of an optimal stopping problem (see Glasserman (2003)). Here, the risk-neutral undiscounted asset price process $(\tilde{Z}_t)^T_{t=0}$ at time steps $0, \ldots, T$ is modeled as a sampled geometric Brownian motion

$$\tilde{Z}_{t+1} = \epsilon_{t+1} \tilde{Z}_t, \quad t = 0, \ldots, T-1, \; Z_0 \in \mathbb{R}_+, \qquad (7)$$

where $(\varepsilon_t)^T_{t=1}$ are independent identically distributed random variables following a log-normal distribution, usually modeled in terms of

$$\varepsilon_t = \exp(((\mu - \frac{\sigma^2}{2})\Delta + \sigma\sqrt{\Delta} N_t),$$

with independent, identically standard normally distributed random variables $(N_t)^T_{t=1}$ where the parameters $\sigma > 0$ and $\mu \in \mathbb{R}_+$ represent the stock price volatility and the continuously compounded interest rate respectively , measured on yearly scale. The tenor Δ describes the length (measured in years) of the equidistant time interval between exercise times. The fair price of such option with strike price K, interest rate parameter $\rho = \mu\Delta$, and maturity date ΔT, is given by the solution to the optimal stopping problem

$$\sup\{\mathbb{E}(e^{-\rho\tau} \max((K - \tilde{Z}_\tau), 0)) : \;\; \tau \text{ is } \{0, 1, \ldots, T\}\text{-valued stopping time}\}.$$

A transformation of the state space is required to represent the reward functions in a convenient way. Thus, we introduce an augmentation with 1 via

$$Z_t = \left[\begin{array}{c} 1 \\ \tilde{Z}_t \end{array} \right], \qquad t = 0, \ldots, T.$$

then it becomes possible to represent the evolution as the linear state dynamics

$$Z_{t+1} = W_{t+1} Z_t, \qquad t = 0, \ldots, T-1$$

with independent and identically distributed matrix-valued random variables $(W_t)_{t=1}^T$ given by

$$W_{t+1} = \left[\begin{array}{cc} 1 & 0 \\ 0 & \epsilon_{t+1} \end{array} \right], \quad t = 0, ..., T-1. \tag{8}$$

This switching system is defined by two positions $\mathbf{P} = \{1, 2\}$ and two actions $\mathbf{A} = \{1, 2\}$. Here, the positions "exercised" and "not exercised" are represented by $p = 1$ and $p = 2$ respectively, and the actions "don't exercise" and "exercise" are denoted by $a = 1$ and $a = 2$ respectively. With this interpretation, the position change is given by deterministic transitions to specified states:

$$\alpha^a_{p,p'} = \begin{cases} 1 & \text{if } p' = \alpha(p, a) \\ 0 & \text{else;} \end{cases} \tag{9}$$

deterministically determined by the target positions

$$(\alpha(p,a))^2_{p,a=1} \sim \left[\begin{array}{cc} \alpha(1,1) & \alpha(1,2) \\ \alpha(2,1) & \alpha(2,2) \end{array} \right] . = \left[\begin{array}{cc} 1 & 1 \\ 2 & 1 \end{array} \right], \tag{10}$$

While the rewards at time $t = 0, \ldots, T$, are defined as

$$r_t(p, (z^{(1)}, z^{(2)}), a) = e^{-\rho t} \max(K - z^{(2)}, 0)(p - \alpha(p, a)), \tag{11}$$

$$r_T(p, (z^{(1)}, z^{(2)})) = e^{-\rho T} \max(K - z^{(2)}, 0)(p - \alpha(p, 2)), \tag{12}$$

for all $p \in \mathbf{P}$, $a \in \mathbf{A}$, $z \in \mathbb{R}_+$.

Code example

As a demonstration, let us consider a Bermuda put option with strike price 40 that expires in 1 year. The put option is exercisable at 51 evenly spaced time points in the year, which includes the start and end of the year. The following code approximates the value functions in the Bellman recursion.

```
library(rcss)
rate <- 0.06                 ## Interest rate
step <- 0.02                 ## Time step between decision epochs
vol <- 0.2                   ## Volatility of stock price process
n_dec <- 51                  ## Number of decision epochs
strike <- 40                 ## Strike price
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)  ## Control
grid <- as.matrix(cbind(rep(1, 301), seq(30, 60, length = 301)))  ## Grid
## Disturbance sampling
u <- (rate - 0.5 * vol^2) * step
sigma <- vol * sqrt(step)
condExpected <- function(a, b) {
    aa <- (log(a) - (u + sigma^2)) / sigma
    bb <- (log(b) - (u + sigma^2)) / sigma
    return(exp(u + sigma^2 / 2) * (pnorm(bb) - pnorm(aa)))
}
weight <- rep(1 / 1000, 1000)
disturb <- array(0, dim = c(2, 2, 1000))
disturb[1,1,] <- 1
part <- qlnorm(seq(0, 1, length = 1000 + 1), u, sigma)
for (i in 1:1000) {
    disturb[2,2,i] <- condExpected(part[i], part[i+1]) / (plnorm(part[i+1], u, sigma) -
            plnorm(part[i], u, sigma))
}
## Subgradient representation of reward
```

```
in_money <- grid[,2] <= strike
reward <- array(0, dim = c(301, 2, 2, 2, n_dec - 1))
reward[in_money,1,2,2,] <- strike
reward[in_money,2,2,2,] <- -1
for (tt in 1:n_dec - 1){
    reward[,,,,tt] <- exp(-rate * step * (tt - 1)) * reward[,,,,tt]
}
## Subgrad representation of scrap
scrap <- array(data = 0, dim = c(301, 2, 2))
scrap[in_money,1,2] <- strike
scrap[in_money,2,2] <- -1
scrap <- exp(-rate * step * (n_dec - 1)) * scrap
## Bellman
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
```

The matrix `grid` represents the grid points where each row represents a point. The 3-dimensional array `disturb` represents the sampling of the disturbances where `disturb[,,i]` gives the i-th sample. Here, we use local averages on a 1000 component partition of the disturbance space. The 5-dimensional array `reward` represents the subgradient approximation with `reward[,,a,p,t]` representing $\mathcal{S}_{\mathbf{G}^m} r_t(p,.,a)$. The object `bellman` is a list containing the approximations of the value functions and expected value functions for all positions and decision epochs. Please refer to the package manual for the format of the inputs and outputs. To visualise the value function of the Bermuda put option, simply run the following plot command:

```
plot(grid[,2], rowSums(bellman$value[,,2,1] * grid), type = "l", xlab = "Stock Price",
     ylab = "Option Value")
```

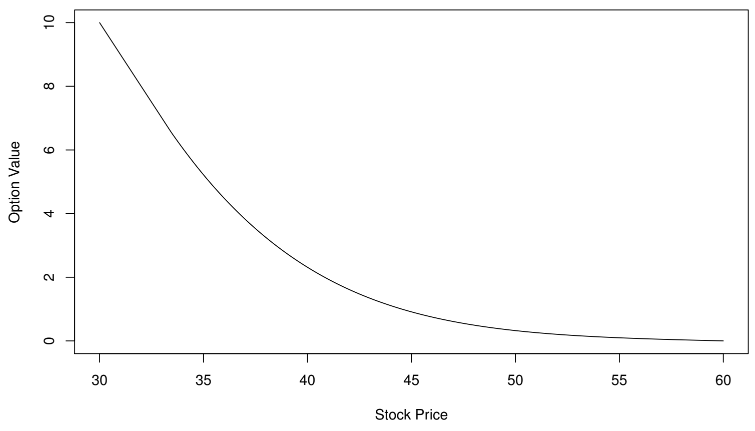


Figure 1: Bermuda put value function.

The following code then performs the solution diagnostics by computing the lower and upper bound estimates for the value of the option when $Z_0 = 36$.

```
## Reward function
RewardFunc <- function(state, time) {
  output <- array(data = 0, dim = c(nrow(state), 2, 2))
  output[,2,2] <- exp(-rate * step * (time - 1)) * pmax(40 - state[,2], 0)
  return(output)
}
## Scrap function
ScrapFunc <- function(state) {
  output <- array(data = 0, dim = c(nrow(state), 2))
  output[,2] <- exp(-rate * step * (n_dec - 1)) * pmax(40 - state[,2], 0)
  return(output)
}
## Get primal-dual bounds
start <- c(1, 36)
## Path disturbances
```

```
set.seed(12345)
n_path <- 500
path_disturb <- array(0, dim = c(2, 2, n_path, n_dec - 1))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm(n_path * (n_dec - 1) / 2)
rand1 <- as.vector(rbind(rand1, -rand1))
path_disturb[2, 2,,] <- exp((rate - 0.5 * vol^2) * step + vol * sqrt(step) * rand1)
path <- PathDisturb(start, path_disturb)
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Subsim disturbances
n_subsim <- 500
subsim <- array(0, dim = c(2, 2, n_subsim, n_path, (n_dec - 1)))
subsim[1,1,,,] <- 1
rand2 <- rnorm(n_subsim * n_path * (n_dec - 1) / 2)
rand2 <- as.vector(rbind(rand2, -rand2))
subsim[2,2,,,] <- exp((rate - 0.5 * vol^2) * step + vol * sqrt(step) * rand2)
subsim_weight <- rep(1 / n_subsim, n_subsim)
mart <- FastAddDual(path, subsim, subsim_weight, grid, bellman$value, ScrapFunc)
bounds <- AddDualBounds(path, control, RewardFunc, ScrapFunc, mart, policy)
```

The above code takes the exact reward and scrap functions as inputs. The function `FastPathPolicy` computes the candidate optimal policy. The object `bounds` is a list containing the primals and duals for each sample path i and each position p at each decision time t. Again, please refer to the package manual for the format of the inputs and outputs. If the price of the underlying asset is 36, the 99% confidence interval for the option price is given by the following.

```
> print(GetBounds(bounds, 0.01, 2))
[1] 4.475802 4.480533
```

The package **rcss** also allows the user to test the prescribed policy from the Bellman recursion on any supplied set of sample paths. The resulting output can then be further studied with time series analysis or other statistical work. In the following code, we will use the previously generated 500 sample paths to backtest our policy and generate histograms.

```
test <- FullTestPolicy(2, path, control, RewardFunc, ScrapFunc, policy)
## Histogram of cumulated rewards
hist(test$value, xlab = "Cumulated Rewards", main = "")
## Exercise times
ex <- apply(test$position == 1, 1, function(x) min(which(x)))
ex[ex == Inf] <- 51
ex <- ex - 1
hist(ex, xlab = "Exercise Times", main = "")
```

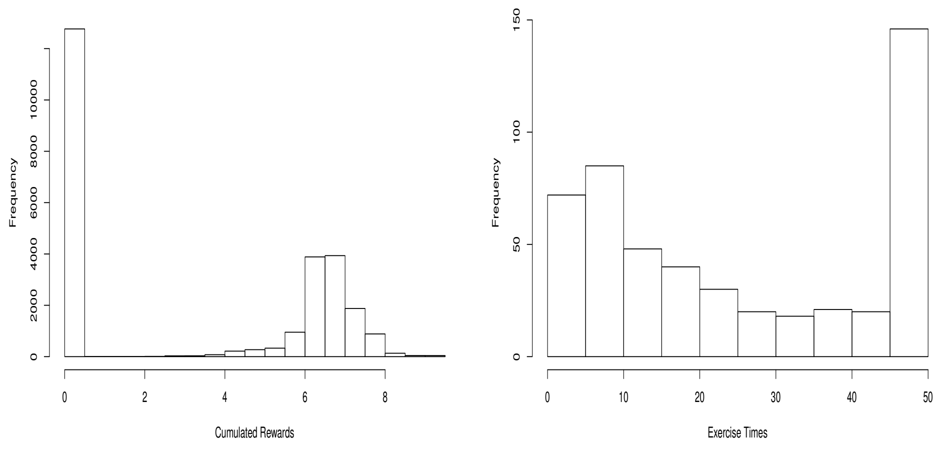


Figure 2: Distribution of cumulated rewards and exercise times.

Figure 2 contains the histograms for the cumulated rewards and exercise times. These can be useful to a practitioner (e.g., anticipating when the holder will exercise their option). Let us emphasise the usefulness of such scenario generation. Given an approximately optimal policy and backtesting, one can perform statistical analysis on the backtested values to obtain practical insights such as for risk analysis purposes.

Example: swing option

Let us now consider the swing option which is a financial contract popular in the energy business. In the simplest form, it gives the owner the right to obtain a certain commodity (such as gas or electricity) at a pre-specified price and volume at a number of exercise times which can be freely chosen by the contract owner. Let us consider a specific case of such a contract, referred to as a *unit-time refraction period* swing option. In this contract, there is a limit to exercise only one right at any time. Given the discounted commodity price $(S_t)_{t=0}^T$, the so-called fair price of a swing option with N rights is given by the supremum,

$$\sup_{0\leq\tau_1<\cdots<\tau_N\leq T} \mathbb{E}\Big[\sum_{n=1}^{N}(S_{\tau_n} - Ke^{-\rho\tau_n})^+\Big],$$

over all stopping times, $\tau_1, \dots, \tau_N$, with values in $\{0, \dots, T\}$. In order to represent this control problem as a switching system, we use the position set $\mathbf{P} = \{1, \dots, N+1\}$ to describe the number of exercise rights remaining. That is, $p \in \mathbf{P}$ stands for the situation when there are $p-1$ rights remaining to be exercised. The action set $\mathbf{A} = \{1, 2\}$ represents the choice between exercising ($a = 1$) or not exercising ($a = 2$). The control matrices $(\alpha^a_{p,p'})$ are given for exercise action $a = 1$

$$\alpha^1_{p,p'} = \begin{cases} 1 & \text{if } p' = \max\{1, (p-1)\} \\ 0 & \text{else,} \end{cases}$$

and for not-exercise action $a = 2$ as

$$\alpha^2_{p,p'} = \begin{cases} 1 & \text{if } p' = p \\ 0 & \text{else} \end{cases}$$

for all $p, p' \in \mathbf{P}$. In the case of the swing option, the transition between p and p' occurs deterministically, since once the controller decides to exercise the right, the number of rights remaining is diminished by one. The deterministic control of the discrete component is easier to describe in terms of the matrix $(\alpha(p,a))_{p\in\mathbf{P},a\in\mathbf{A}}$ where $p' = \alpha(p,a) \in \mathbf{P}$ stands for the discrete component which is reached from $p \in \mathbf{P}$ by the action $a \in \mathbf{A}$. For the case of the swing option this matrix is

$$(\alpha(p,a))_{p\in\mathbf{P},a\in\mathbf{A}} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 2 & 3 \\ \dots & \dots \\ N & N+1 \end{bmatrix}. \tag{13}$$

Having modelled the discounted commodity price process as an exponential mean-reverting process with a reversion parameter $\kappa \in [0, 1[$, long run mean $\mu > 0$, and volatility $\sigma > 0$, we obtain the logarithm of the discounted price process as

$$\tilde{Z}_{t+1} = (1-\kappa)(\tilde{Z}_t - \mu) + \mu + \sigma\epsilon_{t+1}, \quad \tilde{Z}_0 = \ln(S_0). \tag{14}$$

A further transformation of the state space is required before linear state dynamics can be achieved. If we introduce an augmentation with 1 via

$$Z_t = \begin{bmatrix} 1 \\ \tilde{Z}_t \end{bmatrix}, \qquad t = 0, \dots, T,$$

then it becomes possible to represent the evolution as the linear state dynamics

$$Z_{t+1} = W_{t+1}Z_t, \qquad t = 0, \dots, T-1,$$

with independent and identically distributed matrix-valued random variables $(W_t)_{t=1}^T$ given by

$$W_{t+1} = \begin{bmatrix} 1 & 0 \\ \kappa\mu + \sigma\epsilon_{t+1} & (1-\kappa) \end{bmatrix}, \quad t = 0, ..., T-1.$$

The reward and scrap values are given by

$$r_t(p, (z^{(1)}, z^{(2)}), a) = (e^{z^{(2)}} - Ke^{-\rho t})^+(p - \alpha(p,a)) \tag{15}$$

for $t = 0, \dots, T-1$ and

$$r_T(p,(z^{(1)},z^{(2)})) = (e^{z^{(2)}} - Ke^{-\rho T})^{+}(p - \alpha(p,1)) \qquad (16)$$

respectively for all $p \in \mathbf{P}$ and $a \in \mathbf{A}$.

Code example

In this example, consider a swing option with 5 rights exercisable on 101 time points. As before, we begin by performing the value function approximation.

```
## Parameters
rho <- 0
kappa <- 0.9
mu <- 0
sigma <- 0.5
K <- 0
n_dec <- 101                  ## number of time epochs
N <- 5                        ## number of rights
n_pos <- N + 1                ## number of positions
grid <- cbind(rep(1, 101), seq(-2, 2, length = 101))  ## Grid
## Control matrix
control <- cbind(c(1, 1:N), 1:(N + 1))
## Reward subgradient representation
reward <- array(0, dim = c(101, 2, 2, nrow(control), n_dec - 1))
slope <- exp(grid[, 2])
for (tt in 1:(n_dec - 1)) {
    discount <- exp(-rho * (tt - 1))
    for (pp in 2:n_pos) {
        intercept <- (exp(grid[,2]) - K * discount) - slope * grid[, 2]
        reward[, 1, 1, pp, tt] <- intercept
        reward[, 2, 1, pp, tt] <- slope
    }
}
## Scrap subgradient representation
scrap <- array(0, dim = c(101, 2, nrow(control)))
discount <- exp(-rho * (n_dec - 1))
for (pp in 2:n_pos) {
    intercept <- (exp(grid[,2]) - K * discount) - slope * grid[, 2]
    scrap[, 1, pp] <- intercept
    scrap[, 2, pp] <- slope
}
## Disturbance sampling
weight <- rep(1/1000, 1000)
disturb <- array(0, dim = c(2, 2, 1000))
disturb[1, 1,] <- 1
disturb[2, 2,] <- 1 - kappa
CondExpected <- function(a, b){
    return(1/sqrt(2 * pi) * (exp(-a^2/2)- exp(-b^2/2)))
}
part <- qnorm(seq(0, 1, length = 1000 + 1))
for (i in 1:1000) {
  disturb[2,1,i] <- kappa * mu + sigma *
    (CondExpected(part[i], part[i+1]) / (pnorm(part[i+1]) - pnorm(part[i])))
}
## Bellman recursion
r_index <- matrix(c(2, 1), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
```

After obtaining these function approximations, the following code computes the 99% confidence intervals for the value of a swing option with 5 remaining rights.

```
## Exact reward function
RewardFunc <- function(state, time) {
    output <- array(0, dim = c(nrow(state), 2, nrow(control)))
    discount <- exp(-rho * (time - 1))
    for (i in 2:nrow(control)) {
        output[, 1, i] <- pmax(exp(state[, 2]) - K * discount, 0)
    }
    return(output)
}
## Exact scrap function
ScrapFunc <- function(state) {
    output <- array(0, dim = c(nrow(state), nrow(control)))
    discount <- exp(-rho * (n_dec - 1))
    for (i in 2:nrow(control)) {
        output[, i] <- pmax(exp(state[, 2]) - K * discount, 0)
    }
    return(output)
}
## Generate paths
set.seed(12345)
n_path <- 500
path_disturb <- array(0, dim = c(2, 2, n_path, n_dec - 1))
path_disturb[1, 1,,] <- 1
path_disturb[2, 2,,] <- 1 - kappa
rand1 <- rnorm(n_path * (n_dec - 1) / 2)
rand1 <- as.vector(rbind(rand1, -rand1))
path_disturb[2, 1,,] <- kappa * mu + sigma * rand1
start <- c(1, 0)
path <- PathDisturb(start, path_disturb)
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Set subsimulation disturbances
n_subsim <- 500
subsim <- array(0, dim = c(2, 2, n_subsim, n_path, n_dec - 1))
subsim[1, 1,,,] <- 1
subsim[2, 2,,,] <- 1 - kappa
rand2 <- rnorm(n_subsim * n_path * (n_dec - 1) / 2)
rand2 <- as.vector(rbind(rand2, -rand2))
subsim[2, 1,,,] <- kappa * mu + sigma * rand2
subsim_weight <- rep(1 / n_subsim, n_subsim)
## Primal-dual
mart <- FastAddDual(path, subsim, subsim_weight, grid, bellman$value, ScrapFunc)
bounds <- AddDualBounds(path, control, RewardFunc, ScrapFunc, mart, policy)
```

```
> print(GetBounds(bounds, 0.01, 6))
[1] 13.42159 13.44162
```

The tight 99% confidence interval for the price of the swing option above seems to indicate that the function approximations are of a good quality. With this in mind, we now proceed to backtesting the associated policy. We reuse the $1,000$ sample path generated for the solution diagnostics.

```
test <- FullTestPolicy(6, path, control, RewardFunc, ScrapFunc, policy)
## Histogram of cumulated rewards
hist(test$value, xlab = "Cumulated Rewards", main = "")
## Exercise times
ex <- rep(0, n_dec)
for (i in 1:n_path) {
  for (t in 1:(n_dec - 1)) {
    if ((test$position[i, t] != 1) && (test$action[i, t] == 1)) {
      ex[t] <- ex[t] + 1
    }
  }
}
plot(0:(n_dec-1), ex, xlab="Time", ylab="Number of Exercises", type="b", main="")
```

Figure 3 show the histogram for the cumulated rewards and the number of exercises at each time point over the $1,000$ sample paths. As mentioned before, more sophisticated statistical analysis can be performed on these path values to gain practical insights.

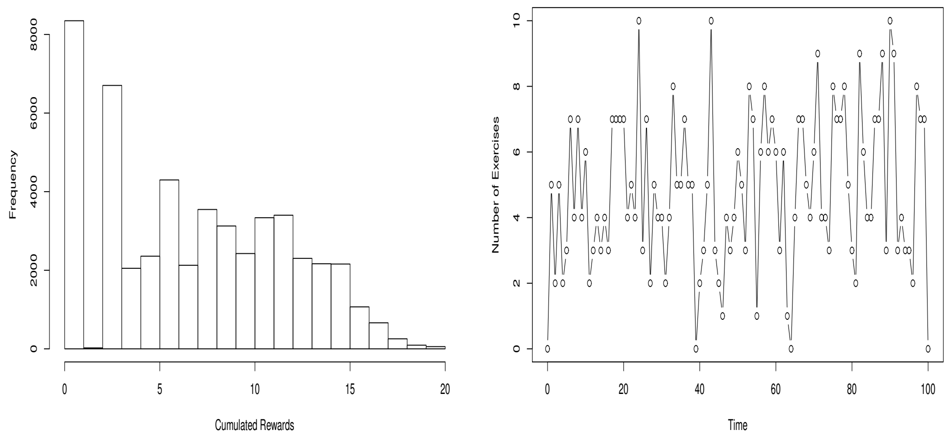


Figure 3: Distribution of cumulated rewards and number of exercises.

Example: optimal energy storage

This section examines optimal energy storage for an energy retailer in the presence of a battery. The model is beyond the scope of this paper and this paper points any interested readers to Hinz and Yee (2017a) for more details.

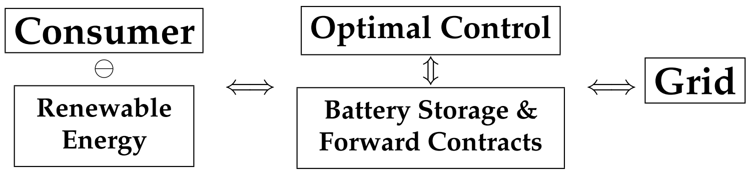


Figure 4: Energy dispatch in the presence of renewable energy and battery storage.

The model is represented by the above figure and the basic gist is as follows:

- At every decision epoch, the energy retailer faces an energy imbalance determined by the net energy demand and the retailer's forward position in the electricity market.
- Any real time imbalance must be rectified by buying or selling at real time grid prices.
- When the electricity retailer has access to a battery, any imbalances can be offset (at least partially) by stored energy. This battery has fixed capacity.
- The retailer's actions is given by the size of the retailer's forward positions in the electricity market. These actions determine the expected amount of energy stored in the battery at each time.
- The goal of the retailer is to reduce the amount of balancing costs and forward trading costs.

The code below begins by specifying the grid, the battery specifications, the action set, and the effect of the action on the battery level. The grid will be used to represent the continuous state component that drives the forward price in the electricity market. Here, the battery has capacity 100 and discretized into steps of 5 from 0. The action set represents the safety margin (the amount of energy bought in the day ahead market that exceeds the predicted net demand) that the retailer may choose. The control array below captures the transition probabilities of the expected change in the battery level based on the current battery level and the retailer's safety margin.

```
## Grid
n_grid <- 501                 ## number of grid points
grid_start <- -15             ## lowest state
grid_end <- 15  ## highest state
grid <- cbind(rep(1, n_grid), seq(grid_start, grid_end, length = n_grid))
## Battery specification
step <- 5                     ## step between battery levels
```

```
pos_start <- 0
pos_end <- 100
position <- seq(pos_start, pos_end, by = step)  ## battery levels
n_pos <- length(position)
## Standard deviation for the consumer demand
std <- 10
## Actions
n_action <- 11                ## number of safety margins
safety_start <- 0
safety_end <- 50
safety <- seq(safety_start, safety_end, length = n_action)  ## safety margins
## Control array
control <- array(data = 0, dim = c(n_pos, n_action, n_pos))
for (p in 1:n_pos) {
    for (a in 1:n_action) {
        temp <- position[p] + safety[a]  ## center of normal distribution
        control[p,a,1] <- pnorm(pos_start + step/2, temp, std)
        control[p,a,n_pos] <- 1 - pnorm(pos_end - step/2, temp, std)
        for (pp in 2:(n_pos-1)) {
            control[p,a,pp] <- pnorm(position[pp] + step/2, temp, std) -
                pnorm(position[pp] - step/2, temp, std)
        }
    }
}
```

Next the code computes the expected excess and shortages of electricity faced by the retailer at each time period.

```
## Functions to calculate expected excess and shortage energy demand
erf <- function(x){  ## error function
    return(2 * pnorm(x * sqrt(2)) - 1)
}
Excess <- function(pos, act) {
    temp1 <- pos_end + step/2
    temp2 <- pos + act
    result <- std/sqrt(2*pi) * exp(-(temp1-temp2)^2/(2*std^2)) +
        (temp2 - pos_end)/2 * (1 - erf(1/sqrt(2*std^2) * (temp1 - temp2)))
    return(result)
}
Shortage <- function(pos, act) {
    temp1 <- pos_start - step/2
    temp2 <- pos + act
    result <- std/sqrt(2*pi) * exp(-(temp1-temp2)^2/(2*std^2)) +
        (pos_start - temp2)/2 * (erf(1/sqrt(2*std^2) * (temp1 - temp2)) + 1)
    return(result)
}
## Expected excess and shortage energy demand
excess <- matrix(data = NA, nrow = n_pos, ncol = n_action)
shortage <- matrix(data = NA, nrow = n_pos, ncol = n_action)
for (p in 1:n_pos) {
    for (a in 1:n_action) {
        excess[p,a] <- Excess(position[p], safety[a])
        shortage[p,a] <- Shortage(position[p], safety[a])
    }
}
```

The subgradient envelope of the reward and scrap functions are then specified below. Essentially, the reward is given by the cost of the forward trading and the cost of any expected shortages or excess energy that needs to be balanced. At the final time, the retailer sells all stored energy in the battery at the forward price. This is captured by the scrap reward function.

```
## Subgradient representation of reward functions
n_dec <- 48 * 7 ## number of decision epochs
u_t <- 10 + cos((0:(n_dec-1)) * 2*pi/48 + 3*pi/2)
v_t <- 1 + (sin((0:(n_dec-1)) * 2*pi/48 + 3*pi/2))/2
```

```
buy <- 20  ## price to buy from grid
sell <- 0  ## price to sell to grid
reward <- array(0, dim = c(n_grid, 2, n_action, n_pos, n_dec - 1))
for (p in 1:n_pos) {
    for (a in 1:n_action) {
        for (t in 1:(n_dec-1)) {
            reward[,1,a,p,t] <- -safety[a] * u_t[t] - shortage[p, a] * buy + excess[p, a] * sell
            reward[,2,a,p,t] <- -safety[a] * v_t[t]
        }
    }
}
scrap <- array(0, dim = c(n_grid, 2, n_pos))
for (p in 1:n_pos) {
    scrap[,1,p] <- position[p] * u_t[n_dec]
    scrap[,2,p] <- position[p] * v_t[n_dec]
}
```

Finally, the disturbance sampling and the resulting double modified value functions are computed using the subgradient approach. Here, we assume that the continuous state (Z_t) driving the forward price is an autoregression of order one. For the disturbance sample, we partition the disturbance space into 1000 components of equal probability measure and take the local average on each component.

```
## Parameters for AR(1) process (Z_t)
mu <- 0
sigma <- 0.5
phi <- 0.9
## Disturbance sampling
n_disturb <- 1000  ## size of sampling
disturb_weight <- rep(1/n_disturb, n_disturb)  ## probability weights
disturb <- array(matrix(c(1, 0, 0, phi), ncol = 2, byrow = TRUE), dim = c(2, 2, n_disturb))
CondExpected <- function(a, b){
  return(1/sqrt(2 * pi) * (exp(-a^2/2)- exp(-b^2/2)))
}
part <- qnorm(seq(0, 1, length = n_disturb + 1))
for (i in 1:n_disturb) {
  disturb[2,1,i] <- mu + sigma *
    (CondExpected(part[i], part[i+1]) / (pnorm(part[i+1]) - pnorm(part[i])))
}
r_index <- matrix(c(2, 1), ncol = 2)  ## randomness index
## Fast bellman recursion
bellman <- FastBellman(grid, reward, scrap, control, disturb, disturb_weight, r_index)
```

The following code example performs the solution diagnostics, computes the bound estimates, and then outputs the results (shown in Table 1). This gives the expected cumulative cost (and its 99% confidence interval) faced by the retailer depending on the current amount of stored energy in the battery. If interested, we point the reader to Hinz and Yee (2017a) for more details regarding this interesting problem.

```
## Exact reward function
RewardFunc <- function(state, time) {
  output <- array(0, dim = c(nrow(state), n_action, n_pos))
  for (p in 1:n_pos) {
    for (a in 1:n_action) {
      output[,a,p] <- -safety[a] * (u_t[time] + v_t[time] * state[,2]) -
        shortage[p,a] * buy + excess[p,a] * sell
    }
  }
  return(output)
}
## Scrap function
ScrapFunc <- function(state) {
  output <- array(0, dim = c(nrow(state), n_pos))
  for (p in 1:n_pos) {
    output[,p] <- position[p] * (u_t[n_dec] + v_t[n_dec] * state[,2])
  }
  return(output)
}
## Generate sample path disturbances
set.seed(12345)
n_path <- 100
path_disturb <- array(matrix(c(1, 0, 0, phi), ncol = 2, byrow = TRUE),
                      dim = c(2, 2, n_path, n_dec - 1))
rand <- rnorm(n_path * (n_dec - 1) / 2)
rand <- as.vector(rbind(rand, -rand))
path_disturb[2,1,,] <- mu + sigma * rand
start <- c(1, 0)  ## z_0
path <- PathDisturb(start, path_disturb)
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Specifying subsimulation disturbance matrices
n_subsim <- 100
subsim_weight <- rep(1/n_subsim, n_subsim)
subsim <- array(matrix(c(1, 0, 0, phi), ncol = 2, byrow = TRUE),
                dim = c(2, 2, n_subsim, n_path, n_dec - 1))
rand <- rnorm(n_subsim * n_path * (n_dec - 1)/2)
rand <- as.vector(rbind(rand, -rand))
subsim[2,1,,,] <- mu + sigma * rand
## Compute primal and dual values
mart <- FastAddDual(path, subsim, subsim_weight, grid, bellman$value, ScrapFunc)
bounds <- AddDualBounds(path, control, RewardFunc, ScrapFunc, mart, policy)
## Storing the results
results <- matrix(data = NA, nrow = n_pos, ncol = 3)
alpha <- 0.01
for (p in 1:n_pos) {
  results[p,1] <- sum(bellman$value[251,,p,1] * grid[251,])
  results[p,2:3] <- GetBounds(bounds, alpha, p)
}
print(results)
```

Start Level (MWh)	Subgradient Estimate	99% CI
0	-1679.859	(-1679.862, -1679.612)
5	-1629.859	(-1629.862, -1629.612)
10	-1579.859	(-1579.862, -1579.612)
15	-1529.859	(-1529.862, -1529.612)
20	-1480.168	(-1480.171, -1479.922)
25	-1433.574	(-1433.577, -1433.328)
30	-1389.685	(-1389.688, -1389.441)
35	-1348.509	(-1348.512, -1348.266)
40	-1310.129	(-1310.131, -1309.887)
45	-1274.601	(-1274.604, -1274.362)
50	-1241.952	(-1241.955, -1241.714)
55	-1212.186	(-1212.188, -1211.949)
60	-1185.294	(-1185.297, -1185.059)
65	-1161.261	(-1161.263, -1161.027)
70	-1140.063	(-1140.066, -1139.831)
75	-1121.677	(-1121.680, -1121.446)
80	-1106.080	(-1106.083, -1105.850)
85	-1093.250	(-1093.253, -1093.021)
90	-1083.161	(-1083.164, -1082.932)
95	-1075.727	(-1075.730, -1075.499)
100	-1070.728	(-1070.731, -1070.500)

Table 1: Subgradient and duality approach using $Z_0^{(2)} = 0$.

Natural resource extraction

As a final example, let us consider the case of optimal resource extraction in Hinz et al. (In Press). Suppose we have a hypothetical copper mine represented in Table 2.

Output rate: 10 million pounds/year	Inventory level: 150 million pounds
Costs of production: $0.50/pound	Opening/closing costs: $200,000
Maintenance costs: $500,000/year	Inflation rate: 8%/year
Convenience yield: 1%/year	Price variance: 8%/year
Real estate tax: 2%/year	Income tax: 50%
Royalty tax: 0%	Interest rate: 10%/year

Table 2: Hypothetical copper mine.

The following code approximates the value function of the above mine using the following setting:

- The mining lease is 30 years. After 30 years, the owner relinquishes ownership of the mine to the government.
- Time is discretized into quarter years.
- At the start of each time epoch, the owner decides whether the mine will be open (i.e. extracting ore) or closed (not extracting) for the next quarter. The owner can also relinquish ownership of the mine early at any of these time points.
- Switching between open and closed states incurs a switching cost.
- An open mine produces income but incurs higher operating costs than a closed mine.
- The amount of ore in the mine is finite and decreases whenever the mine is open.
- The owner wishes to maximise the expected total profits which gives the true value of the mine.

In the code below, the grid will be used to represent the price of the ore which is assumed to follow geometric Brownian motion. The control matrix will be used to reflect changes in the amount of unextracted ore based on the owner's dynamic decision to open or close. For the disturbance sampling, the disturbance space is partitioned into 1000 components of equal probability measure and the local averages on each partition component is taken.

```
## Parameters
n_grid <- 501
grid <- as.matrix(cbind(rep(1, n_grid), seq(0, 10, length = n_grid)))
rate <- 0.1
delta <- 0.01
vol <- sqrt(0.08)
step <- 0.25
## Disturbance sampling
n_disturb <- 1000
disturb_weight <- rep(1/n_disturb, n_disturb)
disturb <- array(0, dim = c(2, 2, n_disturb))
disturb[1, 1,] <- 1
quantile <- rep(NA, n_disturb)
u <- (rate - delta - 0.5 * vol^2) * step
sigma <- vol * sqrt(step)
part <- qlnorm(seq(0, 1, length = n_disturb + 1), u, sigma)
condExpected <- function(a, b){  ##[a,b]
  aa <- (log(a) - (u+sigma^2))/sigma
  bb <- (log(b) - (u+sigma^2))/sigma
  vv <- exp(u + sigma^2/2) * (pnorm(bb) - pnorm(aa))
  return(vv)
}
for (i in 1:n_disturb) {
  quantile[i] <- condExpected(part[i], part[i+1]) /
         (plnorm(part[i+1], u, sigma) - plnorm(part[i], u, sigma))
}
disturb[2, 2,] <- quantile
r_index <- matrix(c(2, 2), ncol = 2)
## Control matrix, a = 1 (close), 2 (abandon), 3 (open)
## p = 1 (exhausted), levels + 1 (full open), 2 * levels + 1 (full closed)
levels <- 15 / step
control <- matrix(data = 1, nrow = (2 * levels + 1), ncol = 3)
control[2:(2 * levels + 1), 1] <- (levels + 2):(2 * levels + 1)
control[2:(2 * levels + 1), 3] <- 1:levels
```

Next, we define the subgradient representation of the reward and scrap functions. When the mine is open, the reward is essentially the income from selling the extracted ore minus the cost of extracting that ore. When the mine is closed, the reward is the cost of maintenance. If the owner switches between open and closed states, there is a switching cost. When the owner relinquishes ownership of the mine, there is no reward or cost. The scrap reward function is set to zero in the below code. These objects are then passed into the FastBellman function which computes the value function approximations using nearest neighbors.

```
## Subgrad rep of rewards
H1 <- 5 * step
H2 <- -2.5 * step
maint <- 0.5 * step
switch <- 0.2
inflation <- 0.08
tax_property <- 0.02
n_dec <- 1 + 30 / step + 1
discount <- exp(-(rate + tax_property) * step)
n_pos <- nrow(control)
n_action <- ncol(control)
n_dim <- ncol(grid)
reward <- array(data = 0, dim = c(n_grid, n_dim, n_action, n_pos, n_dec - 1))
for (p in 2:n_pos) {
  for (t in 1:(n_dec - 1)) {
    pi <- exp(inflation * (t-1) * step)
    adjust <- discount ^ (t-1)
    ## Close the asset
    if (p > (levels + 1)) {  ## Closed
      ## Close
      reward[, 1, 1, p, t] <- adjust * -maint * pi
      ## Open
      reward[, 1, 3, p, t] <- (H2 - switch) * pi * adjust
      reward[, 2, 3, p, t] <- H1 * adjust
    } else if (p <= (levels + 1)) {  ## Opened
      ## Close
      reward[, 1, 1, p, t] <- -(maint + switch) * pi * adjust
      ## Open
      reward[, 1, 3, p, t] <- H2 * pi * adjust
      reward[, 2, 3, p, t] <- H1 * adjust
    }
  }
}
## Subgrad rep of scrap
scrap <- array(data = 0, dim = c(n_grid, n_dim, n_pos))
## Performing fast bellman recursion
bellman <- FastBellman(grid, reward, scrap, control, disturb, disturb_weight, r_index)
```

Finally, running the following code gives an illustration of the value of an open mine based on the current ore price.

```
plot(grid[,2], rowSums(bellman$value[,,levels,1] * grid), type = "l",
        xlab = "Current Ore Price ($)", ylab = "Mine Value ($ millions)")
```

For more details regarding this problem, we point any interested readers to Hinz et al. (In Press).

Conclusion

This paper gives a demonstration of our R package, **rcss** in solving optimal switching problems. Our package represents the first software implementation of this class of algorithms. The problem setting

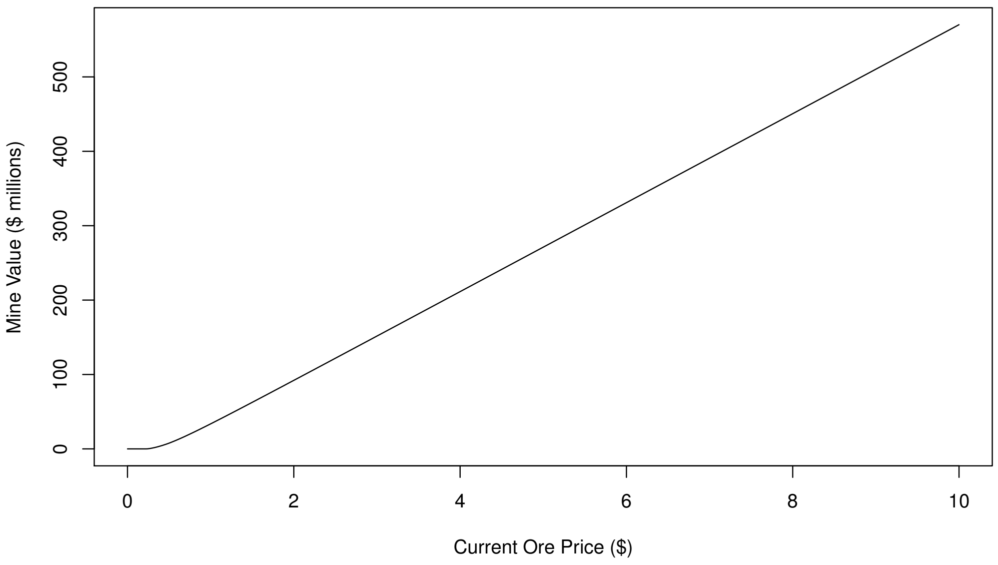


Figure 5: Mine valuation.

discussed in this paper is broad and can be used to model a wide range of problems. Using nearest neighbor algorithms, the package **rcss** is able to solve some real world problems in an accurate and quick manner. It is also worth mentioning that this R package can be used to address Markov decision processes when there is a hidden Markov model as shown in Hinz and Yee (2017b). However, this is not demonstrated here in order to preserve the brevity of this paper.

Bibliography

N. Bauerle and U. Rieder. *Markov Decision Processes with Applications to Finance*. Springer-Verlag, 2011. URL `https://doi.org/10.1007/978-3-642-18324-9`. [p38]

L. Dagum and R. Menon. Openmp: An industry standard api for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998. [p40]

D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL `https://doi.org/10.18637/jss.v040.i08`. [p40]

P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, 2003. [p40]

J. Hinz. Optimal stochastic switching under convexity assumptions. *SIAM Journal on Control and Optimization*, 52(1):164–188, 2014. URL `https://doi.org/10.1137/13091333x`. [p38, 40]

J. Hinz and N. Yap. Algorithms for optimal control of stochastic switching systems. *Theory of Probability and its Applications*, 60(4):770–800, 2015. [p38, 40]

J. Hinz and J. Yee. Optimal forward trading and battery control under renewable electricity generation. *Journal of Banking & Finance*, InPress, 2017a. ISSN 0378-4266. URL `https://doi.org/10.1016/j.jbankfin.2017.06.006`. [p38, 47, 49]

J. Hinz and J. Yee. Stochastic switching for partially observable dynamics and optimal asset allocation. *International Journal of Control*, 90(3):553–565, 2017b. [p38, 53]

J. Hinz, T. Tarnopolskaya, and J. Yee. Efficient algorithms of pathwise dynamic programming for decision optimization in mining operationsh. *Annals of Operations Research*, In Press. [p38, 51, 52]

S. Mangenat and G. Jefferies. *Nabor: Wraps 'Libnabo', a Fast K Nearest Neighbour Library for Low Dimensions*, 2018. URL `https://CRAN.R-project.org/package=nabor`. R package version 0.5. [p40]

H. Pham. *Continuous-Time Stochastic Control and Optimization with Financial Applications*, volume 61. Springer-Verlag, 2009. [p38]

W. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons, 2007. URL https://doi.org/10.1002/9781118029176. [p38]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2013. URL http://www.R-project.org/. ISBN 3-900051-07-0. [p38]

J. Yee. Convex function approximations for Markov decision processes. *arXiv:1712.00970*, Preprint. [p38]

Juri Hinz
University of Technology Sydney
Juri.Hinz@uts.edu.au

*Jeremy Yee**
**Corresponding Author*
University of Technology Sydney
jeremyyee@outlook.com.au

nsROC: An R package for Non-Standard ROC Curve Analysis

by Sonia Pérez-Fernández, Pablo Martínez-Camblor, Peter Filzmoser and Norberto Corral

Abstract The receiver operating characteristic (ROC) curve is a graphical method which has become standard in the analysis of diagnostic markers, that is, in the study of the classification ability of a numerical variable. Most of the commercial statistical software provide routines for the standard ROC curve analysis. Of course, there are also many R packages dealing with the ROC estimation as well as other related problems. In this work we introduce the **nsROC** package which incorporates some new ROC curve procedures. Particularly: ROC curve comparison based on general distances among functions for both paired and unpaired designs; efficient confidence bands construction; a generalization of the curve considering different classification subsets than the one involved in the classical definition of the ROC curve; a procedure to deal with censored data in cumulative-dynamic ROC curve estimation for time-to-event outcomes; and a non-parametric ROC curve method for meta-analysis. This is the only R package which implements these particular procedures.

Introduction

Given a continuous variable, *(bio)marker*, we are frequently interested in performing a binary classification according to its value. This binary classification can be regarded as the presence or not of a certain characteristic of interest in the population (for instance, one disease). On the basis of data containing the real diagnosis, subjects are called *positive* when they have the characteristic and *negative* otherwise.

The receiver operating characteristic (ROC) curve assumes that higher values of the marker are associated with a higher probability of having the characteristic. Therefore, a subject whose marker value is below a fixed point (usually called *threshold* or *cut-off point*) is classified as negative (without the characteristic) while a subject with a marker value above the threshold is classified as positive (with the characteristic). Under this proviso, it displays the ability of the marker to correctly classify a positive subject as positive, or *true-positive rate* (TPR), versus the inability to correctly classify a negative subject as negative, or *false-positive rate* (FPR), for each cut-off point along all the possible values of the marker. That is, the *sensitivity* (TPR) versus the complementary of the *specificity* (FPR) for each possible threshold. In addition, the *area under the ROC curve*, AUC, is frequently used as an index of the global diagnostic capacity (Fluss et al., 2005). It ranges between $1/2$, when the marker does not contribute to a correct classification, and 1, if the marker may classify subjects properly. If AUC is less than $1/2$ it means that the direction of the classification should be the opposite (see comments about side of the ROC curve discussed below).

Mathematically, let χ and ξ be two continuous random variables representing the marker values for negative and positive subjects, respectively. For a fixed value $t \in [0,1]$, the usual ROC curve (*right-sided*) can be defined as follows in terms of the distribution function of negative (F_χ) and positive (F_ξ) group:

$$\mathcal{R}(t) = 1 - F_\xi\left(F_\chi^{-1}(1-t)\right) = F_{1-F_\chi(\xi)}(t)$$

leading the following area under the curve:

$$\mathcal{A} = \int_0^1 \mathcal{R}(t)\,dt = \mathcal{P}\left(\chi < \xi\right).$$

Of course there exists a wide literature dealing with both theoretical and practical aspects of the ROC curve and other related problems. The interested reader can consult the monographs of Zhou et al. (2002) and Pepe (2003) for an extensive review of the topic. There are also a number of papers dealing with some problems related to ROC curve such as the usual ROC curve point estimation (see Gonçalvez et al. (2014) for a recent overview) from both parametric and non-parametric approaches, even considering Bayesian methods as an alternative to the maximum likelihood principle; or the curve interval estimation (confidence bands construction) also using both parametric (Demidenko, 2012) and non-parametric techniques (Jensen et al. (2000), Horváth et al. (2008) and Martínez-Camblor et al. (2018)).

Furthermore, the ROC curve procedure has been extended to other situations where the outcome is not binary. For instance, Mossman (1999) extends ROC concepts to diagnostic tests with trichotomous outcomes; whereas Heagerty and Zheng (2005) deal with time-dependent responses, whose most direct extension is by means of the cumulative/dynamic approach (Heagerty et al., 2000), but it involves a new problem: handling censored data. Additionally, Martínez-Camblor et al. (2017) proposed a ROC curve generalization for non-monotone relationships between the marker and the response, particularly convenient for situations in which both lower and higher marker values are associated with higher probabilities of having the studied characteristic. Some other scenarios where the information is not provided as standard may lead us to conduct a meta-analysis of ROC curves (see Martínez-Camblor (2017) and references therein) or fit a regression model for these curves (Cai (2004) and Rodríguez-Álvarez et al. (2011)).

On the other hand, the ROC curve comparison is one of the issues which has been more treated in literature. Usually ROC curves are compared from their respective AUCs, but in some situations these hypothesis tests are not the most appropriate (further discussed in the **Comparison** section). The similarity between two ROC curves have been traditionally discussed by Venkatraman and Begg (1996) for both paired and unpaired designs (Venkatraman, 2000). On the other hand, the comparison of the curves as functions is not different from the cumulative distribution function comparison problem, and this analogy was used by Martínez-Camblor et al. (2011), and subsequently extended to paired structures (Martínez-Camblor et al., 2013).

Some of the previous approaches have already been implemented in several software packages, including R packages such as **pROC** (Robin et al., 2018) and **ROCR** (Sing et al., 2015) which include different procedures to estimate the usual ROC curve (incorporating smoothing techniques), as well as confidence intervals computation for different parameters of the curve (sensitivity, specificity, AUC) and comparison of areas under two curves. There exist also more specific packages to deal with different particular topics and approaches of the ROC curve. For instance **plotROC** (Sachs, 2018) displays sophisticated plots of these curves; **fbroc** (Peter, 2016) focuses on a fast implementation of bootstrap techniques; **OptimalCutpoints** (Lopez-Raton and Rodriguez-Alvarez, 2014) includes several methods to select optimal cut-off points of the marker; **timeROC** (Blanche, 2015) and **survivalROC** (Heagerty and packaging by Paramita Saha-Chaudhuri, 2013) estimate time-dependent ROC curves and deal with some related analyses; and **HSROC** (Schiller and Dendukuri, 2015) implements a model for joint meta-analysis of sensitivity and specificity of the diagnostic test under evaluation.

ROC curves research is in fact a growing field in statistics. The aforementioned R packages are some of the most relevant ones in this topic but there are also more implementations covering certain algorithms. However, some non-standard ROC curve analyses exist which were not available to the scientific community in a practical software and this is the main reason why the new package presented in this paper has been created. The **nsROC** package (Pérez-Fernández, 2018) is a compilation of different analyses not computed to date which attempts to boost awareness of new techniques that have already been published but not implemented in a user-friendly software widely available. Furthermore, it incorporates several studies and techniques (from comparison of ROC curves to time-dependent estimation and meta-analysis), making it more manageable since all of them are included in the same package.

The rest of the paper is organized as follows: in the next two sections, **Estimation** and **Comparison**, some basic information about the statistical techniques included in the **nsROC** package, as well as some remarkable technical issues about its main functions, are provided. Particularly, the **Estimation** section incorporates several aforementioned situations: the ROC curve generalization for non-monotone relationships, confidence bands construction, censored data treatment for time-dependent outcomes, and meta-analysis involving ROC curves. In turn, the **Comparison** section includes three different methods of comparison (based on AUC, diagnostic capacity of the marker, or ROC curve definition in terms of CDF) to deal with both paired and unpaired data scenarios. Subsequently, in the **Examples** section, a complete analysis with different datasets is carried out to illustrate certain applications of the submitted package; and finally a **Summary** of the utility of the package is reported.

Estimation

Non-standard ROC curve estimation

As mentioned previously, an ROC curve is a graphical method which displays the sensitivity (*Se*) versus the complementary of the specificity (1-*Sp*) for all possible thresholds of the considered marker.

Although different parametric and semi-parametric estimators for the ROC curve have been studied, in our package the empirical estimator, based on replacing the involved unknown distribution functions with their respective empirical cumulative distribution functions, $\hat{F}$, has been considered. Hence, the implemented ROC curve estimator is

$$\widehat{\mathcal{R}}(t) = \hat{F}_{1-\hat{F}_{\chi}(\xi)}(t)\,.$$

This is the usual definition when higher values of the marker are considered to be associated with a higher probability of existence of the characteristic under study. It can be also called *right-sided* ROC curve.

However, sometimes it can be supposed the opposite, i.e. that higher values of the marker are associated with a lower probability of the existence of the characteristic. In this context, the definitions should be adapted and the resulting ROC curve (usually called *left-sided* curve) estimator is

$$\widehat{\mathcal{R}}(t) = \hat{F}_{\hat{F}_\chi(\xi)}(t)\,.$$

There exist several R packages also incorporating the non-parametric estimation, for instance the **pROC** package includes smoothed estimates. However, they suppose one of the assumptions aforementioned (right-sided or left-sided curve), considering a single threshold of the marker in order to classify, since the standard ROC curve definition is associated with this particular type of classification subsets.

Nevertheless, an extension of those classification subsets has been studied by Martínez-Camblor et al. (2017), dealing with situations in which not only higher or lower values of the marker are associated with a higher probability of existence of the studied characteristic, but both may be related. Under this assumption, not only one cut-off point is considered, but two x_l and x_u corresponding to the extremes of a marker interval are regarded, i.e those subjects with a marker value within the interval (x_l, x_u) are classified as negative and those with a marker value below x_l or greater that x_u are supposed to be positive. In this context, the sensitivity and specificity definitions are the following ones:

$$Se(x_l, x_u) = P\left(\xi \leq x_l \cup \xi \geq x_u\right) = F_\xi(x_l) + 1 - F_\xi(x_u)$$
$$Sp(x_l, x_u) = P\left(x_l < \chi < x_u\right) = F_\chi(x_u) - F_\chi(x_l).$$

At this juncture, it is important to note that there may be different couples (x_l, x_u) reporting the same specificity but different sensitivity, so the generalized ROC curve is defined by the supreme of them:

$$\mathcal{R}_g(t) = \sup_{(x_l,x_u)\in\mathcal{F}_t} \left\{F_\xi(x_l) + 1 - F_\xi(x_u)\right\}$$

where $(x_l, x_u) \in \mathcal{F}_t$ iff $x_l \leq x_u$ and $Sp(x_l, x_u) \geq 1 - t$. It is clear that $(x_l, x_u) \in \mathcal{F}_t$ can also be written as $x_l = F_\chi^{-1}(\gamma t)$ and $x_u = F_\chi^{-1}(1 - [1-\gamma]\,t)$ for some $\gamma \in [0, 1]$, therefore

$$\mathcal{R}_g(t) = \sup_{\gamma\in[0,1]} \left\{F_\xi\left(F_\chi^{-1}(\gamma t)\right) + 1 - F_\xi\left(F_\chi^{-1}(1 - [1-\gamma]\,t)\right)\right\}.$$

Using the aforementioned notation, the implemented *general ROC curve* estimator is

$$\widehat{\mathcal{R}}_g(t) = \sup_{\gamma\in[0,1]} \left\{1 - \hat{F}_{1-\hat{F}_\chi(\xi)}(1 - \gamma t) + \hat{F}_{1-\hat{F}_\chi(\xi)}([1-\gamma]\,t)\right\}.$$

Different parametric models have been considered in order to estimate the ROC curve. Among them, the binormal model is one of the most used, according to which the usual and general ROC curves, respectively, are the following:

$$\mathcal{R}(t) = \Phi\left(a + b \cdot \Phi^{-1}(t)\right)$$

$$\mathcal{R}_g(t) = \sup_{\gamma\in[0,1]} \left\{\Phi\left(a + b \cdot \Phi^{-1}([1-\gamma]\cdot t)\right) + 1 - \Phi\left(a + b \cdot \Phi^{-1}(1 - \gamma \cdot t)\right)\right\}$$

where $a = \left(\mu_\xi - \mu_\chi\right)/\sigma_\xi$, $b = \sigma_\chi/\sigma_\xi$ and Φ is the cumulative distribution function of a standard normal. Therefore, the parametric ROC curve estimation gets boiled down to estimate the parameters involved.

While the usual AUC has a direct probabilistic interpretation: "given two randomly and independently selected subjects, one negative and one positive, the AUC is the probability that the marker value in the positive subject is greater than in the negative subject", this reading is not directly related to the classification subsets involved in the definition of the usual ROC curve. However, it is possible to enunciate this relationship in terms of the diagnostic rule involved (citing Martínez-Camblor and Pardo-Fernández (2017)) and following the same idea the authors also proved the interpretation of the generalized AUC in terms of the probability of belonging to the corresponding classification subsets, under a condition about the continuity of $R_g(\cdot)$ and self-contained subsets as specificity increases.

In the **nsROC** package the point non-parametric ROC curve estimation can be computed by the `gROC` function. Some computational details must be mentioned: if `Ni` is `NULL` a fast algorithm is used to estimate the ROC curve for the considered sample; otherwise, if `Ni` is a number, thresholds considered are the marker values collected (adding $-\infty$ and ∞) and the specificities, t, used to estimate the ROC curve are those resulting from dividing the unit interval in `Ni` subintervals with the same length. This latter case is slower because the vector of γ-values taken into account in order to estimate the general ROC curve is the result of dividing the unit interval in subintervals with length 0.001. The area under the curve is computed by the trapezoidal rule.

Input parameters	
`X`	Vector of marker values.
`D`	Vector of response values. Two levels; if more, the two first ones are used.
`side`	Type of ROC curve. One of `"right"` (right-sided), `"left"` (left-sided), `"auto"` (right or left-sided is automatically chosen so that AUC will be greater than 0.5) or `"both"` (general). Default: `"right"`.
`Ni`	Number of subintervals of the unit interval considered to compute the curve. Default: NULL (which will use the fast algorithm considering as many subintervals as number of positive subjects).
`pval.auc`	If TRUE, a permutation test to test $H_1 : AUC \neq 0$ is performed.
`B`	Number of permutations used for testing. Default: 500.
Output parameters	
`controls,cases`	Marker values of negative and positive subjects, respectively.
`points.coordinates`	Matrix whose second and third columns correspond to coordinates where the ROC curve has a step in case of right or left-sided ROC curves. In the first column there are the marker thresholds considered reporting these coordinates.
`pairpoints.coordinates`	Matrix whose third and fourth columns correspond to coordinates where the ROC curve has a step in case of general ROC curves. The first and second columns are the marker thresholds considered, x_l and x_u, respectively, reporting these coordinates.
`roc`	Vector of values of the ROC curve for each `t` considered.
`auc`	Area under the curve estimate.
`pval.auc,Paucs`	p-value and different permutation AUCs if the hypothesis test is performed.
Additional functions to be passed	
`plot`	Plot the ROC curve estimate.
`print`	Print some relevant information.

Table 1: The most relevant input and output parameters of the `gROC` function.

The point estimation of the curve is essential, but it is also important to have an idea of how relevant the underlying sample is in this estimation, i.e. the interval estimation: how to build confidence bands of the ROC curve. This problem has been addressed from different points of view, most of them based on point-wise confidence intervals for sensitivity and/or specificity instead of focusing on the curve as a function.

There are some R packages providing some kind of confidence regions: **fbroc** includes a function which computes regions for the right-sided curve but no information about the method used to build them is provided; **plotROC** displays 'rectangular confidence regions for the ROC curve'; and **pROC** computes square pointwise confidence bands of the AUC, thresholds, specificity, sensitivity and/or coordinates of an ROC curve.

A review of the performance of these methods has already been carried out by Macskassy et al. (2005) who pointed out the difficulty of translating methods for building pointwise confidence intervals into methods to obtain confidence bands. However, when the focus is the whole ROC curve, one should construct confidence bands, and just considering the 'band' obtained joining the pointwise confidence intervals does not provide a real confidence band with the desired confidence level, because the probability that one point of the curve will be outside this 'band' is higher.

In this package three different techniques dealing with the ROC curve itself have been computed. Namely, one parametric assuming the binormal model (Demidenko (2012)) and two non-parametric have been included (Jensen et al. (2000) and Martínez-Camblor et al. (2018)):

- Demidenko (2012) adapted the Working-Hotelling type confidence bands used in linear regression and proposed a method called ellipse-envelope. It should be noted that the ROC curve estimated by this method is not empirical, but the binormal one.
- Jensen et al. (2000) approach is based on the asymptotic distribution of the ROC curve in terms of Brownian bridges, developing symmetrical non-parametric confidence bands for the curve, even on a particular region. The main drawback is the need to estimate density functions from smooth procedures involving a scale parameter (not chosen by the user) which can strongly

affect the resulting ROC curve estimate. In terms of computational aspects it must be pointed out that the `BBridge` function in the **sde** (Iacus, 2016) package has been used to simulate the Brownian bridges involved. In addition, the extremes of the interval in $(0, 1)$ in which the user wants to compute the regional confidence bands must be set. The bootstrap method has been applied and the confidence bands are truncated making the lower-band being inside the $(0, 0.95)$ interval and the upper-band within $(0.05, 1)$.

- Martínez-Camblor et al. (2018) method approximates the distribution of the following pivotal function by a smoothed bootstrap method:

$$\sqrt{n} \cdot \sigma_n^{-1}(t) \cdot \left[\widehat{\mathcal{R}}(t) - \mathcal{R}(t)\right]$$

where n is the number of positive subjects and $\sigma_n(t)$ is the standard deviation estimate of $\sqrt{n}\left[\widehat{\mathcal{R}}(t) - \mathcal{R}(t)\right]$. Computational issues which should be taken into account are the following: confidence bands are truncated as in the previous method and the scale parameter, s, used to compute the smoothed kernel distribution functions (with bandwidth $h = s \cdot \hat{\sigma} \cdot \min\{n_+, n_-\}$) must be set by the user. Furthermore, there exists the option of selecting a parameter, α_1, affecting the width between lower (and consequently upper) band and ROC curve point estimate. If α_1 is not specified by the user, the one minimizing the theoretical area between the bands is automatically considered. It should be remarked that this is the only method designed to estimate ROC curve confidence bands for the general ROC curve.

	Input parameters
`groc`	Output of the `gROC` function. `Ni` is the number of subintervals used for estimation.
`method`	Method used. One of `"PSN"` (Martínez-Camblor et al., 2018), `"JMS"` (Jensen et al., 2000) or `"DEK"` (Demidenko, 2012).
`conf.level`	Confidence level considered. Default: 0.95.
`B`	Number of bootstrap replicates. Default: 500.
`alpha1,s`	Parameters to pass to `"PSN"` method. Default: `s`= 1.
`a.J,b.J`	Extremes of interval to pass to `"JMS"` method. Default: `a.J`= 1/`Ni`, `b.J`= 1 − 1/`Ni`.
`plot.var`	If `TRUE`, variance estimate along t resulting from `"PSN"` or `"JMS"` method is displayed.
	Output parameters
`L,U`	Lower and upper bands respectively for each $t \in \{0, 1/\texttt{Ni}, 2/\texttt{Ni}, \ldots, 1\}$.
`practical.area`	Estimated area between lower and upper band.
`alpha1,alpha2`	α_1 and α_2 used in `"PSN"` method.
	Additional functions to be passed
`plot`	Plot the confidence bands of the ROC curve.
`print`	Print some relevant information.

Table 2: The most relevant input and output parameters of the `ROCbands` function.

Time-dependent ROC curve

Sometimes the response variable is not binary but time-dependent. In this case the resulting curve is called time-dependent ROC curve. Although there exist different approaches of this kind of curves depending on the association between the referred time-dependent outcome and the binary classification (for instance, Heagerty and Zheng (2005) considered the *incident* sensitivity defined as $Se^I(x) = P(X > x | T = t)$ to build the *incident/dynamic* ROC curve), the most direct one is the *cumulative/dynamic* approach, which classifies as positive a subject in which the event happens before a fixed point of time t and negative otherwise. In other words, the *cumulative sensitivity* and the *dynamic specificity* are defined as follows: $Se^C(x) = P(X > x | T \leq t)$ and $Sp^D(x) = P(X \leq x | T > t)$.

However, the time-dependent problem involves a new issue to be addressed: how to deal with subjects censored before t. There are some R packages which incorporate time-dependent ROC curve estimation procedures in the presence of censored data. Some good examples are **timeROC**, which also performs some estimations about different concepts related to time-dependent ROC curve and compare time-dependent AUCs (see Blanche et al. (2013) for a complete overview of the implemented methods); **survivalROC** which computes time-dependent ROC curves from censored survival data using the Kaplan-Meier (KM) or Nearest Neighbor Estimation (NNE) method by Heagerty et al. (2000); and **tdROC** (Li et al., 2016), based on the Li et al. (2018) method mentioned below.

In order to deal with time-dependent outcomes, the **nsROC** package has used the *cumulative/-dynamic* approach. A different solution for the censoring problem has been proposed by Martínez-Camblor et al. (2016), considering a time-dependent ROC curve estimator based on assigning a probability to be negative (consequently positive) to those censored subjects. Particularly, two different statistics have been suggested in order to estimate the probability of surviving beyond *t*: a semiparametric one, using a proportional hazard Cox regression model considering the marker as the covariate; and a non-parametric one, using directly the Kaplan-Meier estimator. There exists a subsequent paper based on the same idea (Li et al., 2018) but using the kernel-weighted Kaplan-Meier estimator instead of the naive one. This last method is also included in **nsROC** package, allowing the user to choose the kernel and bandwidth to be considered in the kernel-weighted statistic.

In terms of computational aspects it should be noted that the **survival** (Therneau, 2018) package has been used. In particular, the `survfit` and `Surv` functions are required to estimate survival functions, and the `coxph` function is used to fit the Cox proportional hazard regression model involved in the semiparametric approach aforementioned.

	Input parameters
`stime`	Vector of observed times.
`status`	Vector of status (0 if the subject is censored and 1 otherwise).
`marker`	Vector of marker values.
`predict.time`	Time point *t* considered.
`method`	Method to estimate the probability aforementioned. One of `"Cox"`, `"KM"` or `"wKM"`.
`kernel`	Procedure used to calculate kernel function if `method` is `"wKM"`. One of `"normal"`, `"Epanechnikov"` or `"other"` (if the user defines a different one).
`h,kernel.fun`	Bandwidth and kernel function used if `method` is `"wKM"` and `kernel` is `"other"`.
`boot.n`	Number of bootstrap samples considered. Default: 100.
	Output parameters
`TPR,TNR`	Vector of sensitivities and specificities estimates, respectively.
`cutPoints`	Vector of marker thresholds considered.
`auc`	Area under the time-dependent ROC curve estimate.
	Additional functions to be passed
`plot`	Plot the time-dependent ROC curve estimate.
`print`	Print some relevant information.

Table 3: The most relevant input and output parameters of the `cdROC` function.

Meta-analysis

Meta-analysis is a popular statistical methodology for combining the results from multiple independent studies about the same topic. It allows us to know the state of the art, strengths and weaknesses of one considered topic, combining estimation effects from different independent comparable studies (Riley et al., 2010). However, the main particularity of meta-analysis is that only limited information is available from each study considered. There exist two different meta-analysis models depending on the consideration (or not) of the variability between studies: the *fixed-effects* model just considers the within-study variability whereas the *random-effects* model also takes into account the variability between studies (DerSimonian and Laird, 1986).

In the case that the target is the ROC curve, the goal of meta-analysis is combining the results from several independent studies performed by the same marker and characteristic of interest in a single outcome. Different methods to compute summary ROC curves have been introduced in order to determine the global diagnostic accuracy for both fixed-effects (Moses et al. (1993)) and random-effects model (Hamza et al. (2008), among others). Besides, the **HSROC** package implements the procedure of Rutter and Gatsonis (2001). However, most of those approaches are parametric and consider that only one estimated pair of sensitivity and specificity from each paper exist and they are supposed to be independently selected in each study, but often the reported points are the best ones in the Youden index sense. Nevertheless, some new techniques have been developed taking into account all the pairs of points reported; Hoyer and Kuss (2018) and Steinhauser et al. (2016) are good examples. Martínez-Camblor (2017) includes a different view focusing on the direct ROC curve estimation from a non-parametric approach, using weighted means of each individual ROC curve, taking all pairs of points (Se, Sp) reported in each study, and performing a simple linear interpolation between them. Moreover, both the fixed and random-effects model are covered.

The `metaROC` function in the **nsROC** package implements this last approach reporting a fully non-parametric ROC curve estimate from a data frame including the number of true positive and negative (`TP` and `TN`) subjects, false positive and negative (`FP` and `FN`) subjects and a identifier of the study they come from. It displays in a plot the non-parametric summary ROC (nPSROC) curve estimate, and the user has the possibility of including all ROC curve interpolations in the same graphic, as well as a confidence band estimate. In the random-effects model there is also the option of plotting the inter-study variability estimate along the different specificities on the unit interval.

Input parameters	
`data`	A data frame containing the variables: `"Author"`, `"TP"`, `"TN"`, `"FP"` and `"FN"`.
`model`	Meta-analysis model considered. One of `"fixed-effects"` or `"random-effects"`.
`Ni`	Number of subintervals of the unit interval considered to compute the curve. Default: 1000.
`plot.Author`	If `TRUE`, a plot including ROC curve estimates (by linear interpolation) for each study under consideration is displayed.
`plot.bands`	If `TRUE`, confidence interval estimate for the ROC curve is added.
`plot.inter.var`	If `TRUE`, a plot reflecting inter-study variability estimate is displayed on an additional window.
Output parameters	
`sRA`	nPSROC curve estimate resulting from the `model` considered with a slight modification to ensure the monotonicity along the points on the unit interval considered.
`se.RA`	Standard-error of nPSROC curve estimate.
`area`	Area under the curve estimate.
`youden.index`	Optimal specificity and sensitivity in the Youden index sense for nPSROC curve.
`roc.j`	A matrix whose columns contain the ROC curve estimate (by linear interpolation) of each study.
`w.j,w.j.rem`	A matrix whose columns contain the weights in fixed or random-effects model, respectively, of each study.

Table 4: The most relevant input and output parameters of the `metaROC` function.

Comparison

An important role of diagnostic medicine research is the comparison of the accuracy of diagnostic tests. With the goal of comparing their global accuracy, the comparison of AUCs is the most usual method (DeLong et al., 1988). However, when there is no uniform dominance between the involved curves (i.e. the sensitivities associated with each specificity along the unit interval are not always higher in one curve than in the other), they can differ having the same AUC. In these situations, these tests are not valid to compare the equality among the ROC curves, and some other approaches could be considered to compare the equality of all the curves, such as Martínez-Camblor et al. (2013) and Martínez-Camblor et al. (2011) mentioned below, which deal with the ROC curve by its definition as a cumulative distribution function. On the other hand, Venkatraman and Begg (1996) and Venkatraman (2000) propose the use of a non-parametric permutation test to compare the equality of two diagnostic criteria. Both paired and unpaired designs have been treated, i.e. when different markers for detecting the existence of one characteristic are compared in the same sample (just one positive-negative sample) or when the same marker is compared along different and independent samples (as many positive-negative samples as groups to compare), respectively.

In the first case (paired design), different non-parametric tests have been implemented to perform the comparison:

- The procedure of Martínez-Camblor et al. (2013) takes into account the expression of the ROC curve in terms of the distribution function shown in the **Estimation** section and extends classical tests for comparing the cumulative distribution functions to this context. Four of these tests have been included in the `compareROCdep` function but any other can be defined by the `FUN.dist` input parameter. Those included are the following: Kolmogorov-Smirnov, the two ones based on the L_1 or L_2 measure and Cramér von-Mises. It is important to highlight that the user can set any other criteria to perform the test.

  Two different methods could also be considered in order to approximate the distribution function of the selected statistic under the null hypothesis: the procedure of Venkatraman and Begg (1996) or the one of Martínez-Camblor and Corral (2012) based on permutated and bootstrap samples, respectively. This last one (gBA) is a novel bootstrap procedure which allows us to deal with complex structures.

- Venkatraman and Begg (1996) method tests the hypothesis that two curves are identical for all cut-off points. It should be noted that the permutation procedure covered in this paper requires the *exchangeability* assumption.

  Some technical issues should be also indicated: if the comparison involves more than two ROC curves, the value of the statistic is the sum of the corresponding values of each pair without repetition. In addition, the Venkatraman estimator has been developed just for comparing right-sided ROC curves.

- One test based on the comparison of the areas under the curve has also been included; in particular, the one proposed by DeLong et al. (1988). It should be noted that two different ROC curves can have the same AUC as it has been mentioned above. In computational terms this procedure takes longer because the statistic involved requires *positive sample size* $\times$ *negative sample size* comparisons.

Input parameters	
`X`	A matrix whose columns are the vectors of each marker-values sample.
`D`	Vector of response values.
`side`	Type of ROC curve. One of `"right"` or `"left"`.
`statistic`	Statistic used to compare the curves. One of `"KS"`, `"L1"`, `"L2"`, `"CR"`, `"other"` (if the user defines a different one using other input parameters) or `"VK"`.
`FUN.dist`	The distance considered as a function of one variable. Example: `FUN.dist = function(g){max(abs(g))}` defines the Kolmogorov-Smirnov statistic.
`method`	Method used to approximate the statistic distribution under the null. One of `"general.bootstrap"`, `"permutation"` or `"auc"`.
`B,perm`	Number of bootstrap or permutation samples considered, respectively. Default: 500.
`plot.roc`	If `TRUE`, a plot including the ROC curve estimates for each sample and their mean is displayed.
Output parameters	
`statistic`	The value of the test statistic.
`p.value`	The p-value for the test.

Table 5: The most relevant input and output parameters of the `compareROCdep` function.

In the second case (unpaired design), different non-parametric tests have also been implemented to perform the comparison. They are similar to the previous ones:

- The comparisons of Martínez-Camblor et al. (2011) are inspired by the usual distances between cumulative distribution functions. Three of those distances have been included in the `compareROCindep` function (particularly, the two ones based on L_1 and L_2 measures and the Cramér von-Mises criterion), but it should be highlighted that the user has the possibility to define any other distance by the `FUN.stat.dist` and `FUN.stat.cons` input parameters, described in more detail below.

  Furthermore, the permutation method proposed by Venkatraman (2000) is used to approximate the distribution function of the selected statistic under the null. Related to this method, both raw or ranked data (including a method for breaking ties) could be considered.

- The procedure of Venkatraman (2000) is based on the idea that two ROC curves are identical if and only if for every cut-off point from one marker there is an equivalent one from the other with the same probabilities of failure, i.e the same sensitivity and specificity. Technical issues which are worth noting are the same as those aforementioned in the Venkatraman method for paired samples.

  A straightforward k-sample non-parametric test for the AUC statistic computing the differences with respect to the mean can also be considered. It should be remembered the consideration mentioned above about comparing areas under the curve.

Examples

Some examples analysing real data-sets are shown in this section in order to illustrate the application of the different functions included in the **nsROC** package. Namely, the *Breast Cancer dataset* is used to show the different estimation of the ROC curve considering the usual definition versus the generalization (`gROC` function) as well as confidence bands estimation reported by different procedures, particularly `PSN`, `JMS` and `DEK` (`ROCbands` function). Furthermore, a comparison of the ROC curve reported by two different markers is performed by the `compareROCdep` function and also the diagnostic capacity of one marker in three different groups is studied by the `compareROCindep` function. The intended goal of the *Primary Biliary Cirrhosis dataset* example is considered to show time-dependent ROC curve estimation in the presence of censored data at a specific time using different procedures implemented in `cdROC` function: `Cox`, `KM` and `wKM` with different kernels. Finally, the *Interleukin 6 dataset* includes the results regarding diagnostic ability of a marker over the same characteristic reported by different research papers and the goal is to perform a meta-analysis over them in order to unify the studies in one unique response (`metaROC` function).

Input parameters	
`X`	Vector of marker values.
`G`	Vector of group identifier values (with as many levels as independent samples to compare).
`D`	Vector of response values.
`side`	Type of ROC curve. One of `"right"` or `"left"`.
`statistic`	Statistic used to compare the curves. One of `"L1"`, `"L2"`, `"CR"`, `"other"` (if the user defines a different one using other input parameters), `"VK"` or `"AUC"`.
`FUN.stat.int`	A function of two variables, `roc.i` and `roc` standing for ROC curve estimate for i-th sample and mean ROC curve estimate along k samples, respectively. Example: `FUN.stat.int = function(roc.i,roc){mean(abs(roc.i -roc))}` defines the L_1-measure statistic.
`raw`	If TRUE, raw data is considered; if FALSE (default) data is ranked and a method to break ties in permutations is performed.
`perm`	Number of permutation samples considered. Default: 500.
`plot.roc`	If `TRUE`, a plot of ROC curve estimates for each sample and their mean is displayed.
Output parameters	
`statistic`	The value of the test statistic.
`p.value`	The p-value for the test.

Table 6: The most relevant input and output parameters of the `compareROCindep` function.

Breast cancer dataset

The Breast Cancer dataset consists of several features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass describing the characteristics of the cell nuclei present in a 3D image. This dataset, freely available at https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data , includes a *diagnosis* variable ("malignant" vs "benign") and ten real-valued features about each cell nucleus (*radius*, *texture*, *perimeter*, *area*, *smoothness*, *compactness*, *concavity*, *concave points*, *symmetry* and *fractal dimension*) collected from 569 patients in Wisconsin. The *mean*, *standard error* and *worst* (defined as the mean of the three largest values) of these features were computed for each image, resulting in 30 variables. The reader is referred to Bennett and Mangasarian (1992) for a complete information about the dataset.

There exists a variable, the *fractal dimension (mean)*, which does not seem to correctly distinguish between "malignant" and "benign" cases, reporting an usual ROC curve (left-sided) crossing the diagonal with an AUC of 0.513. Looking at the density function estimates displayed in Figure 1 (top), it can be seen that although the vast majority of the fractal dimension values are in the interval $(0.055, 0.075)$ in both groups (so this marker is not a good one to perform the classification), lower and higher values are likely to be "malignant" cases (positive subjects). Thus, it makes sense to compute the general ROC curve estimate proposed in this package, which reports an AUC of 0.633, higher than the usual one, and of course the curve is above the diagonal by definition (see graph bottom-right in Figure 1).

Figure 1 and information about ROC curve estimates have been reported using the `gROC` function:

```
library(data.table)
data <- fread('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-
cancer-wisconsin/wdbc.data')
names(data) <- c("id", "diagnosis", "radius_mean", "texture_mean", "perimeter_mean",
                 "area_mean", "smoothness_mean", "compactness_mean", "concavity_mean",
                 "concave.points_mean", "symmetry_mean", "fractal_dimension_mean",
                 "radius_se", "texture_se", "perimeter_se", "area_se", "smoothness_se",
                 "compactness_se", "concavity_se", "concave.points_se", "symmetry_se",
                 "fractal_dimension_se", "radius_worst", "texture_worst",
                 "perimeter_worst", "area_worst", "smoothness_worst",
                 "compactness_worst", "concavity_worst", "concave.points_worst",
                 "symmetry_worst", "fractal_dimension_worst")
attach(data)

library(nsROC)

roc <- gROC(fractal_dimension_mean, diagnosis, side = "auto", plot.density = TRUE)
generalroc <- gROC(fractal_dimension_mean, diagnosis, side = "both")

print(roc)
#> Data was encoded with B (controls) and M (cases).
#> Wilcoxon rank sum test:
#>          alternative hypothesis: median(cases) < median(controls); p-value = 0.7316
#> It is assumed that lower values of the marker indicate larger confidence that a
#> given subject is a case.
```

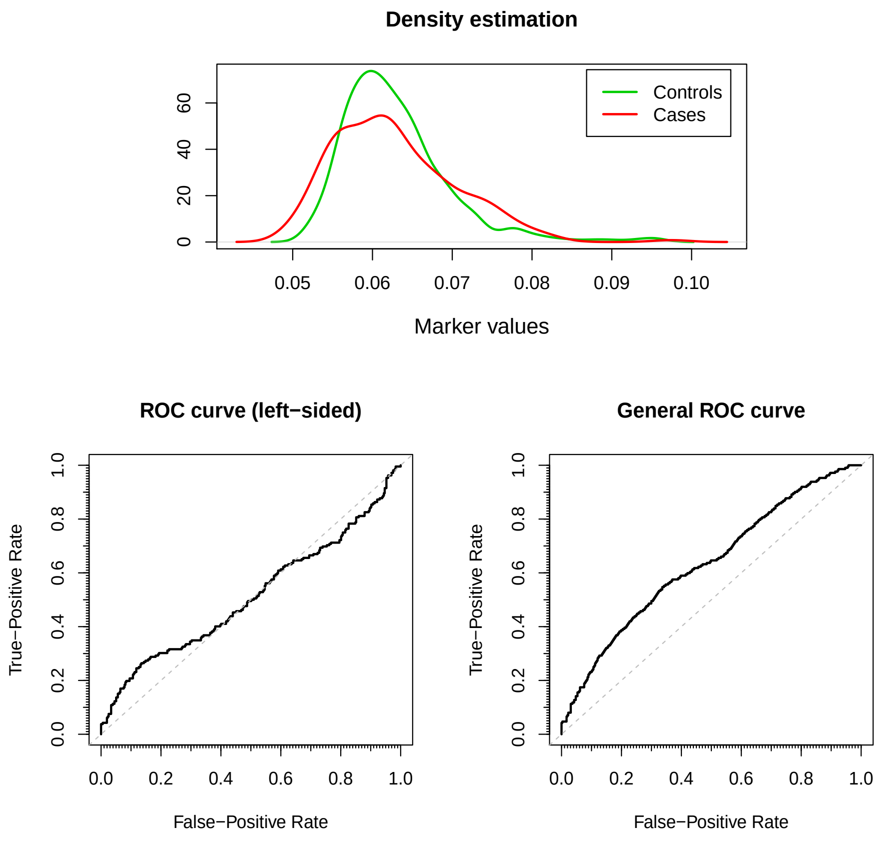


Figure 1: Top, density function estimates of fractal dimension mean variable for both “malignant” and “benign” subjects. Bottom, left-sided and generalized ROC curve estimates, respectively.

```
#> There are 357 controls and 212 cases.
#> The area under the ROC curve (AUC) is 0.513.

print(generalroc)
#> Data was encoded with B (controls) and M (cases).
#> It is assumed that both lower and larges values of the marker indicate larger
#> confidence that a given subject is a case.
#> There are 357 controls and 212 cases.
#> The area under the ROC curve (AUC) is 0.633.

plot(roc, main = "ROC curve (left-sided)")
plot(generalroc, main = "General ROC curve")
```

In order to illustrate the confidence bands construction reported by each method implemented (`"PSN"` (Martínez-Camblor et al., 2018), `"JMS"` (Jensen et al., 2000) and `"DEK"` (Demidenko, 2012)), a marker with a better global diagnostic accuracy (in terms of AUC) than the previous one has been considered: the *texture (mean)*.

In Figure 2 it can be seen that not only the bands are different but also the ROC curve point estimates. This is because each method uses a different way to compute it: "PSN" considers the same as the one computed in the `gROC` function, "JMS" performs a similar one with smoothed estimators and "DEK" computes a parametric estimate based on the assumption of the binormal model. This last one displays the narrowest confidence bands as it was expected (with an area between the CI bands equals to 0.069).

The `ROCbands` function has been used for this purpose:

```
roc <- gROC(texture_mean, diagnosis)        # right-sided in this case

rocbands_psn <- ROCbands(roc, method = "PSN")
rocbands_psn_mod <- ROCbands(roc, method = "PSN", alpha1 = 0.025)
rocbands_jms <- ROCbands(roc, method = "JMS")
rocbands_dek <- ROCbands(roc, method = "DEK")
```

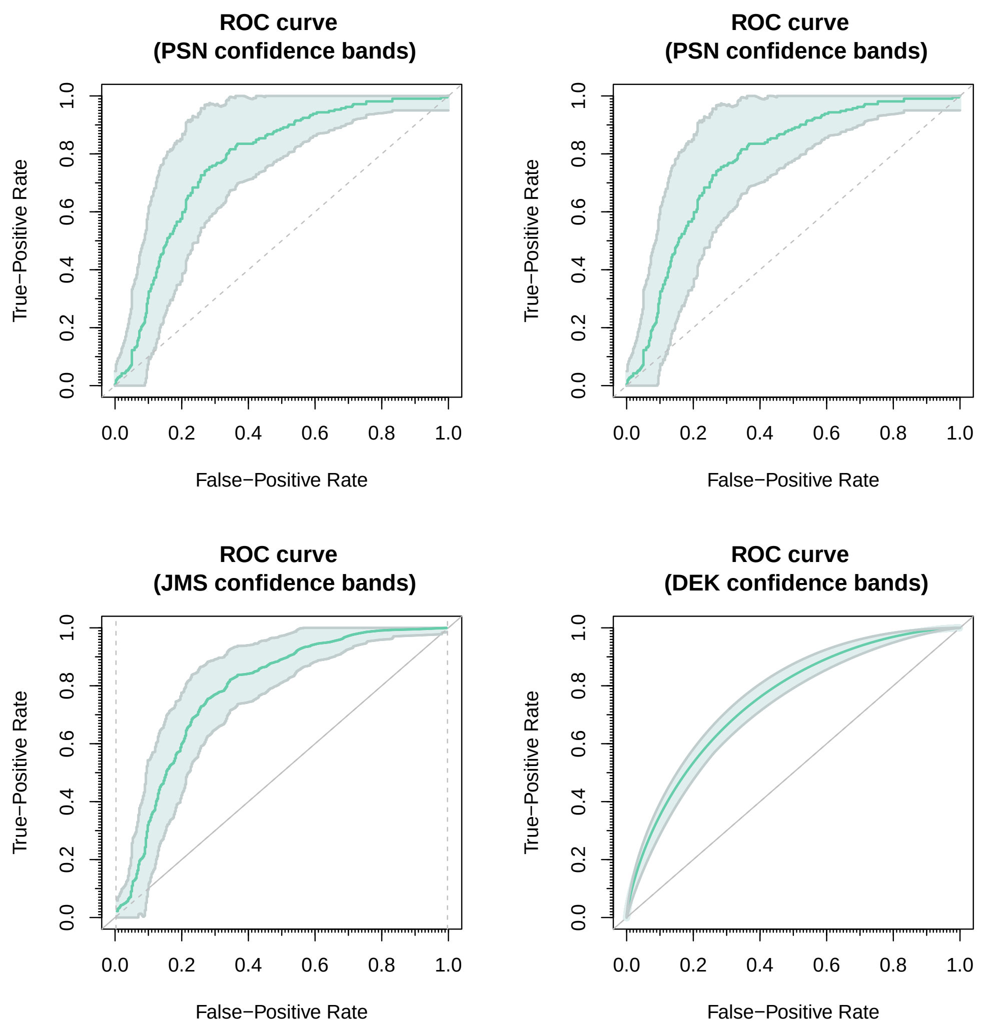
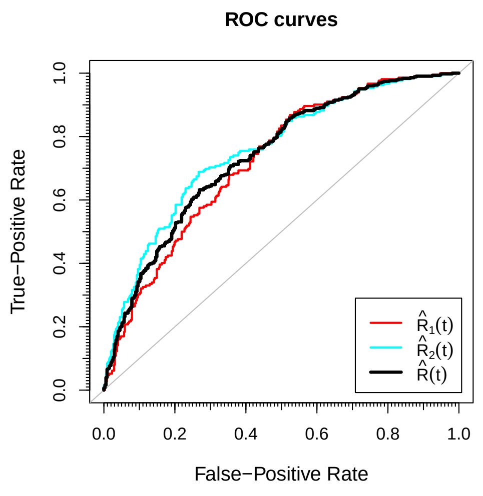


Figure 2: Top, confidence bands for the ROC curve using the "PSN" procedure for optimal α_1 and fixed $\alpha_1 = \alpha/2$, respectively. Bottom, confidence bands for the ROC curve constructed by the "JMS" and "DEK" method, respectively. Confidence level: $1 - \alpha = 0.95$.

The computations performed to get the graphics and some useful information about the confidence bands construction are detailed below:

```
print(rocbands_psn)
#> The method considered to build confidence bands is the one proposed in
#> Martinez-Camblor et al. (2016).
#> Confidence level (1-alpha): 0.95.
#> Bootstrap replications: 500.
#> Scale parameter (bandwidth construction): 1.
#> The optimal confidence band is reached for alpha1 = 0.035 and alpha2 = 0.015.
#> The area between the confidence bands is 0.2294 (theoretically 0.2453).

print(rocbands_psn_mod)
#> The method considered to build confidence bands is the one proposed in
#> Martinez-Camblor et al. (2016).
#> Confidence level (1-alpha): 0.95.
#> Bootstrap replications: 500.
#> Scale parameter (bandwidth construction): 1.
#> alpha1: 0.025.
#> The area between the confidence bands is 0.2368 (theoretically 0.2539).

print(rocbands_jms)
#> The method considered to build confidence bands is the one proposed in
#> Jensen et al. (2000).
#> Confidence level (1-alpha): 0.95.
#> Bootstrap replications: 500.
#> Interval in which compute the regional confidence bands: (0.00280112,0.9971989).
#> K.alpha: 3.163202.
#> The area between the confidence bands is 0.1668.

print(rocbands_dek)
#> The method considered to build confidence bands is the one proposed in
#> Demidenko (2012).
#> Confidence level (1-alpha): 0.95.
#> The area between the confidence bands is 0.0694.

plot(rocbands_psn)
plot(rocbands_psn_mod)
plot(rocbands_jms)
plot(rocbands_dek)
```

Figure 3 presents the comparison of two dependent ROC curves describing the ability of the markers *mean* and the *worst smoothness* to make an accurate diagnosis. Different procedures dealing with different estimators and ways to approximate their distribution under the null hypothesis ($H_0 : R_1(t) = R_2(t) \;\; \forall t \in (0,1)$) have been considered.



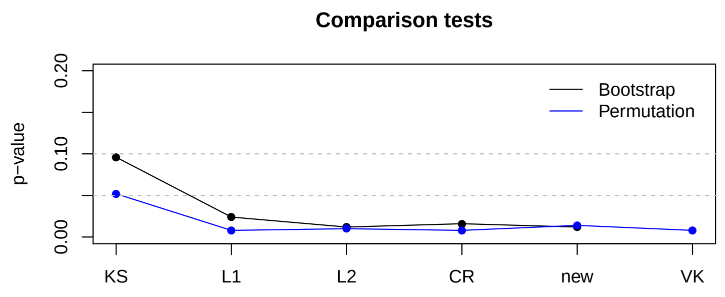


Figure 3: Top, ROC curve estimates for *mean* ($\widehat{R}_1(t)$) and *worst* ($\widehat{R}_2(t)$) *smoothness*, in black the mean ROC curve estimate ($\widehat{R}(t)$). Bottom, p-values of previous tests by bootstrap (black line) and permutated (blue line) iterations based on the Kolmogorov-Smirnov test, L_1 and L_2 measures, Cramér von-Mises criterion, a new one whose statistic value is defined as $\frac{1}{2}\sum_{i=1}^{2}\int_0^1 n^2 \left(\widehat{R}_i(t) - \widehat{R}(t)\right)^4 dt$, and the Venkatraman approach.

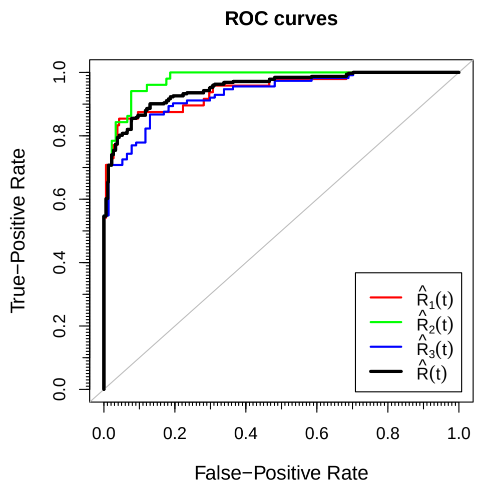
The p-values reported by every method are below 0.05 except for Kolmogorov-Smirnov. It should be noted that the p-values returned by the Venkatraman permutation method are slightly lower than those ones obtained by the general bootstrap technique.

The `compareROCdep` function has been used with this objective:

```
depmarker <- cbind(smoothness_mean,smoothness_worst)
```

```
out.KS <- compareROCdep(depmarker, diagnosis)
out.L1 <- compareROCdep(depmarker, diagnosis, statistic = "L1")
out.L2 <- compareROCdep(depmarker, diagnosis, statistic = "L2")
out.CR <- compareROCdep(depmarker, diagnosis, statistic = "CR")
out.new <- compareROCdep(depmarker, diagnosis, statistic = "other",
                         FUN.dist = function(g){mean(g^4)})

out.perm.KS <- compareROCdep(depmarker, diagnosis, method = "perm")
out.perm.L1 <- compareROCdep(depmarker, diagnosis, statistic = "L1", method = "perm")
out.perm.L2 <- compareROCdep(depmarker, diagnosis, statistic = "L2", method = "perm")
out.perm.CR <- compareROCdep(depmarker, diagnosis, statistic = "CR", method = "perm")
out.VK <- compareROCdep(depmarker, diagnosis, statistic="VK")
out.perm.new <- compareROCdep(depmarker, diagnosis, statistic = "other", method = "perm",
                              FUN.dist = function(g){mean(g^4)})
```

On the other hand, Figure 4 reflects the comparison of three independent ROC curves performed to analyze the diagnostic accuracy of the *mean radius* variable in each group defined by *symmetry* values: *group 1* if *symmetry_mean* < 0.18 and *symmetry_worst* < 0.29, *group 3* if *symmetry_mean* > 0.18 and *symmetry_worst* > 0.29, and *group 2* otherwise. The five estimators computed in the `compareROCindep` function have been used.



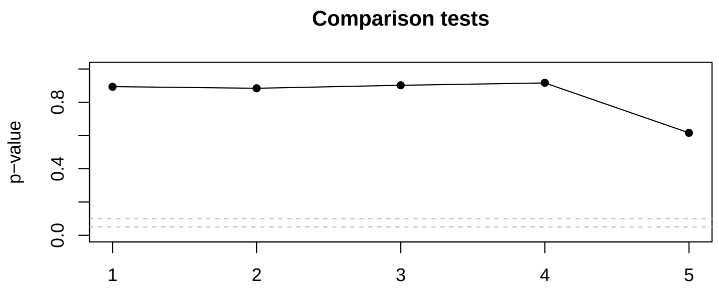


Figure 4: Top, ROC curve estimates for *radius mean* variable in each group ($\widehat{R}_i(t)$) and their mean ROC curve estimate ($\widehat{R}(t)$). Bottom, p-values of previous tests based on L_1 and L_2 measures, Cramér von-Mises criterion, Venkatraman approach and AUC comparison test.

The p-value is greater than 0.1 for all tests considered, being the one reported by the AUC approach the lowest one. Therefore it might be concluded that there is no statistically significant evidence to state that these three ROC curves differ.

The commands used to build Figure 4 are the following, using the `compareROCindep` function:

```
type <- as.numeric(symmetry_mean > 0.18) + as.numeric(symmetry_worst > 0.29) + 1
table(type,diagnosis)
#>     diagnosis
#> type   B   M
#>    1 189  48
#>    2  91  51
#>    3  77 113

output.L1 <- compareROCindep(radius_mean, type, diagnosis, statistic = "L1")
output.L2 <- compareROCindep(radius_mean, type, diagnosis, statistic = "L2")
output.CR <- compareROCindep(radius_mean, type, diagnosis, statistic = "CR")
output.VK <- compareROCindep(radius_mean, type, diagnosis, statistic = "VK")
output.AUC <- compareROCindep(radius_mean, type, diagnosis, statistic = "AUC")
```

Primary Biliary Cirrhosis (PBC) Data

The Primary Biliary Cirrhosis (PBC) dataset contains the results of a trial in PBC of the liver conducted between 1974 and 1984 referred to Mayo Clinic. A total of 424 PBC patients met eligibility criteria for the randomized placebo controlled trial of the drug D-penicillamine; among them, the 393 non-transplanted ones have been considered for this analysis. This dataset is freely available within the R package **survival** by the name pbc. The reader is referred to Therneau and Grambsch (2000) for a complete information about the study.

In order to analyze how good the marker *serum bilirubin (mg/dl)* is to detect those patients who died or survived by 4000 days from their registration in the study, the ROC curve has been estimated. However, there are some patients censored before the regarded time, and two different approaches have been considered in order to estimate the survival probability of those patients censored before the time considered: Figure 5 at top-left, a semi-parametric one based on Cox regression model; at top-right and bottom, a non-parametric one based on naive and smoothed Kaplan-Meier estimators, respectively.

As shown in Figure 5, the different approaches considered report similar ROC curves but it should be noted that the area under the curve reported by the weighted Kaplan-Meier method with normal kernel is slightly higher (AUC= 0.809) because the sensitivities related to specificity values close to one are the highest.

The cdROC function has been used for this purpose:

```
library(survival)
data <- subset(pbc, status!=1)
attach(data)
status <- status/2

out1 <- cdROC(time, status, bili, 4000)
out2 <- cdROC(time, status, bili, 4000, method = "KM")
out3 <- cdROC(time, status, bili, 4000, method = "wKM")
out4 <- cdROC(time, status, bili, 4000, method = "wKM", kernel = "other",
              kernel.fun = function(x,xi,h){u <- (x-xi)/h; 1/(2*h)*(abs(u) <= 1)}, h = 0.5)

plot(out1, main="ROC curve at time 4000 (Cox method)")
text(0.8, 0.1, paste("AUC =", round(out1$auc,3)))
plot(out2, main="ROC curve at time 4000 (KM method)")
text(0.8, 0.1, paste("AUC =", round(out2$auc,3)))
plot(out3, main="ROC curve at time 4000 \n (Weighted KM method with normal kernel)")
text(0.8, 0.1, paste("AUC =", round(out3$auc,3)))
plot(out4, main="ROC curve at time 4000 \n (Weighted KM method with uniform kernel)")
text(0.8, 0.1, paste("AUC =", round(out4$auc,3)))
```

Interleukin 6 (IL6) Data

The Interleukin 6 (IL6) dataset includes the results of 9 papers which study the use of the IL6 as a marker for the early detection of neonatal sepsis. An analysis of this dataset, freely available within the **nsROC** package by the name interleukin6, can be found in Martínez-Camblor (2017). Particularly it includes true-positive (TP), false-positive (FP), true-negative (TN) and false-negative (FN) sizes for all cut-off points reported in each paper, resulting in 19 entries.

Figure 6 shows the summary ROC curve estimate from the 9 papers included, considering either a fixed-effects or a random-effects meta-analysis model (up and down, respectively). The optimal point of the curve in the Youden index sense is displayed, as well as the area under the curve. In this case, the curve does not vary much when the variability between studies is taken into account, reporting similar AUC (0.772 and 0.788, respectively) as a consequence. In addition, both estimates seem to be below most of the interpolated curves they come from; that is because the weights for the

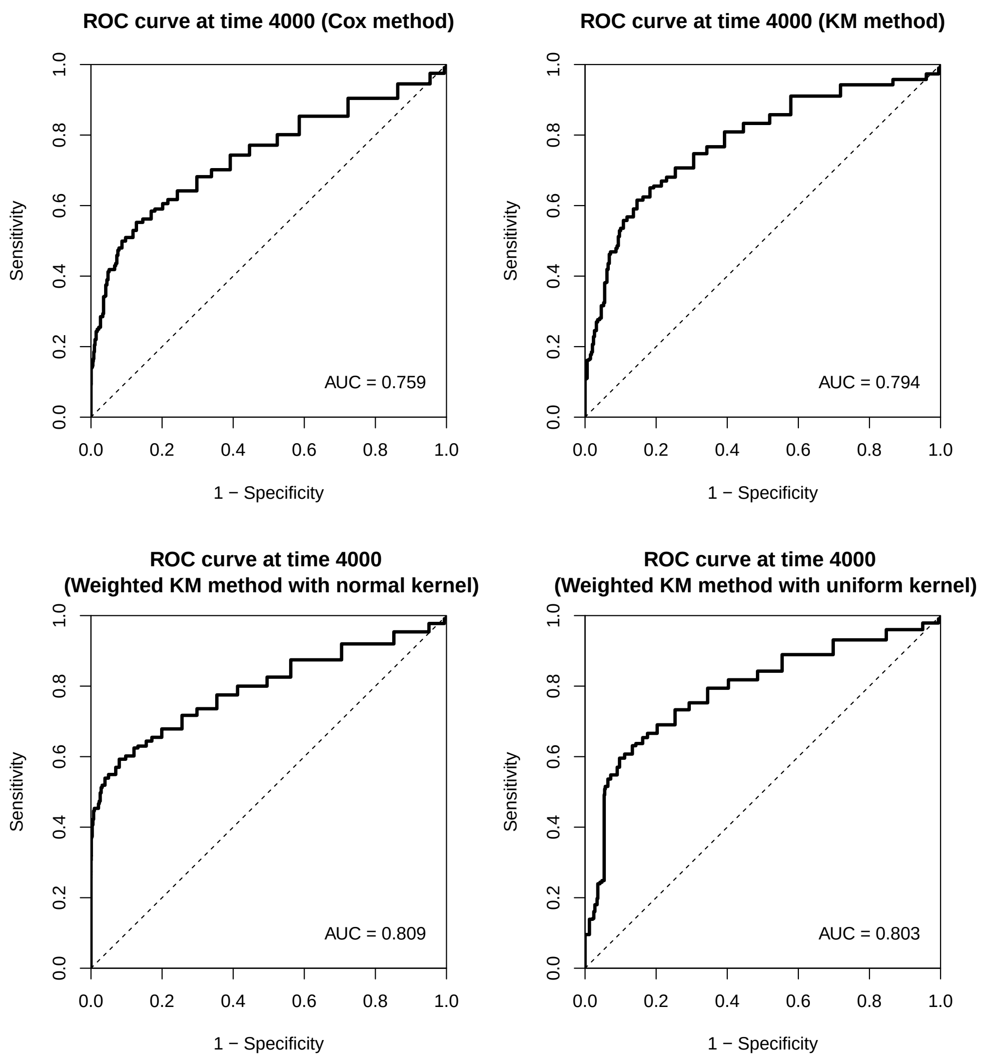


Figure 5: Time-dependent ROC curve estimate using `"Cox"`, `"KM"` (top) and `"wKM"` method with normal kernel and bandwidth $h = 1$ and with uniform kernel and $h = 0.5$ (bottom), respectively.

study number 9 (with an interpolate ROC curve close to diagonal) are the largest ones in the interval $(0, 0.5)$. As it can be seen in the bottom-right plot of Figure 6, the FPR interval with higher inter-study variability is $(0, 0.2)$.

The code computed, using the `metaROC` function, is listed below:

```
data(interleukin6)

output1 <- metaROC(interleukin6, plot.Author = TRUE)
#> Number of papers included in meta-analysis: 9
#> Model considered: fixed-effects
#> The area under the summary ROC curve (AUC) is 0.772.
#> The optimal specificity and sensitivity (in the Youden index sense) for summary
#> ROC curve are 0.7 and 0.76, respectively.
points(1-output1$youden.index[1], output1$youden.index[2], pch = 16, col = 'blue')

output2 <- metaROC(interleukin6, model = "random-effects", plot.Author = TRUE,
                   plot.inter.var = TRUE)
#> Number of papers included in meta-analysis: 9
#> Model considered: random-effects
#> The area under the summary ROC curve (AUC) is 0.788.
#> The optimal specificity and sensitivity (in the Youden index sense) for summary
#> ROC curve are 0.701 and 0.763, respectively.
points(1-output2$youden.index[1], output2$youden.index[2], pch = 16, col = 'blue')
```

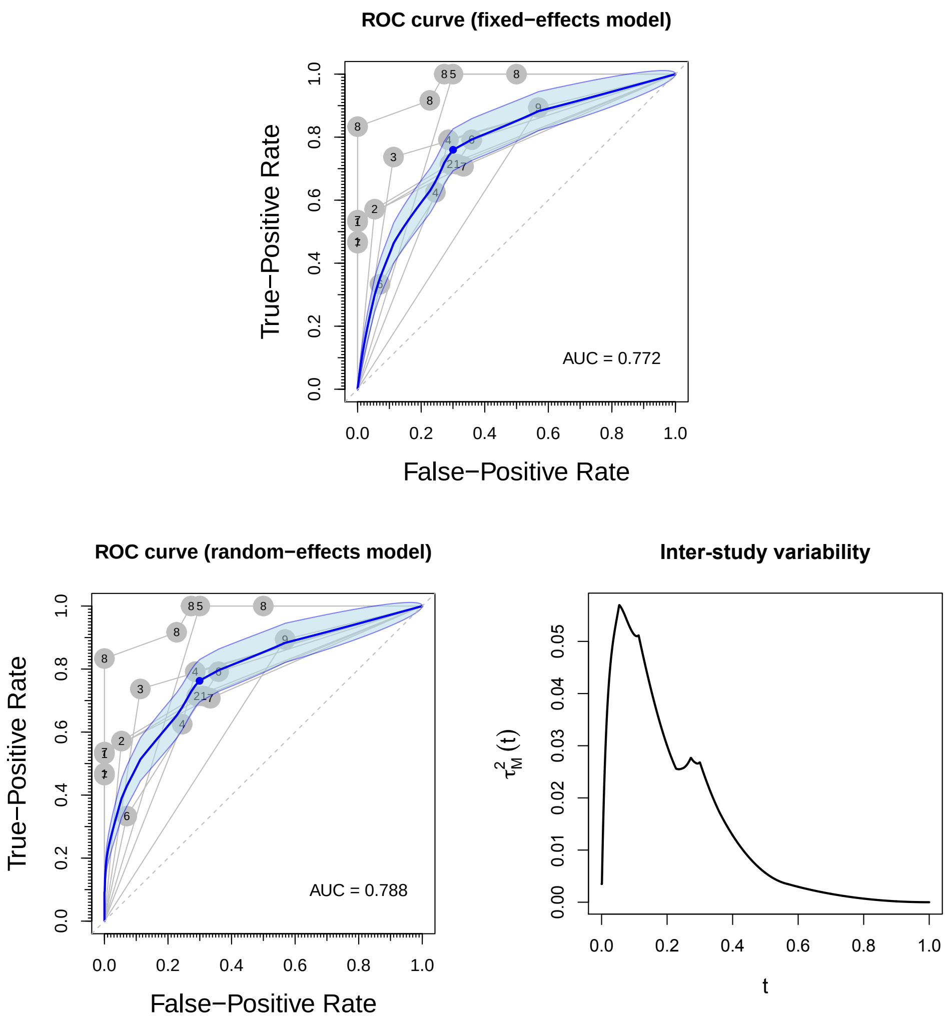


Figure 6: Summary ROC curve estimate considering a fixed-effects (top) and a random-effects meta-analysis model (bottom), respectively. Bottom-right, inter-study variability estimate of summary ROC curve reported by a random-effects model.

Summary

This article introduces the usage of the R package **nsROC** for analyzing ROC curves. In particular, the package contains the following new techniques:

- point ROC curve non-standard estimation implementing the generalization proposed by Martínez-Camblor et al. (2017) [`gROC` function];
- confidence bands construction by three different methods: two of them are non-parametric (Jensen et al. (2000) and Martínez-Camblor et al. (2018)) and the other one is based on the binormal model (Demidenko (2012)) [`ROCbands` function];
- time-dependent ROC curve estimation, dealing with the presence of censored data respect to the time-dependent response variable following Martínez-Camblor et al. (2016) procedure [`cdROC` function];
- meta-analysis, implementing the methods proposed by Martínez-Camblor (2017), covering both fixed and random-effects model considering all the points of the curve reported in each study [`metaROC` function];
- comparison of several ROC curves using different procedures, among which the ones based on usual tests to compare distribution functions proposed by Martínez-Camblor et al. (2011) and Martínez-Camblor et al. (2013) stand out. Not only the usual tests can be performed, but the user can define any other by the input parameters in the `compareROCdep` and `compareROCindep` functions.

In spite of the popularity of R packages about ROC curves dealing with some of the most important analyses related to this tool, **nsROC** includes some algorithms which had not been computed to date in order to address some of those standard analyses (such as time-dependent ROC curve estimation and comparison between curves) and others totally new such as the generalized ROC curve estimation and non-parametric procedure for meta-analysis. Any of these particular techniques had been addressed earlier, excluding the usual estimation of the curve, the weighted Kaplan-Meier method to deal with the presence of censored data in time-dependent ROC curves estimation, and the Venkatraman and DeLong approaches to compare diagnostic accuracies of two tests.

The following table indicates which functions in the package can be used for different options of side of the ROC curve. In addition to this, it should be mentioned that `cdROC` function estimates a time-dependent ROC based on cumulative sensitivity and dynamic specificity definitions, which are ultimately related to right-sided ROC curve. On the other hand, `metaROC` function includes directly the TP, FP, TN and FN as input parameters, and those may have been generated by any ROC curve approach, but it should be the same for all studies considered.

Right	Left	Both
`gROC`	`gROC`	`gROC`
`ROCbands`	`ROCbands(method=="PSN")`	`ROCbands(method=="PSN")`
`compareROCdep`	`compareROCdep(method!="VK")`	
`compareROCindep`	`compareROCindep(statistic!="VK")`	

Table 7: Options of side of the ROC curve for different functions of the **nsROC** package

Acknowledgements

The authors acknowledge support by the Grants MTM2015-63971-P and MTM2014-55966-P from the Spanish Ministerio of Economía y Competitividad and by FC-15-GRUPIN14-101 from the Principado de Asturias.

Bibliography

K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1(1):23–34, 1992. URL `https://doi.org/10.1080/10556789208805504`. [p63]

P. Blanche. *timeROC: Time-Dependent ROC Curve and AUC for Censored Survival Data*, 2015. URL `https://CRAN.R-project.org/package=timeROC`. R package version 0.3. [p56]

P. Blanche, J.-F. Dartigues, and H. Jacqmin-Gadda. Estimating and comparing time-dependent areas under receiver operating characteristic curves for censored event times with competing risks. *Statistics in Medicine*, 32(30):5381–5397, 2013. URL `https://doi.org/10.1002/sim.5958`. [p59]

T. Cai. Semi-parametric ROC regression analysis with placement values. *Biostatistics*, 5(1):45–60, 2004. URL `https://doi.org/10.1093/biostatistics/5.1.45`. [p56]

E. R. DeLong, D. M. DeLong, and D. L. Clarke-Pearson. Comparing the Areas under Two or More Correlated Receiver Operating Characteristic Curves: A Nonparametric Approach. *Biometrics*, 44(3): 837–845, 1988. URL `https://doi.org/10.2307/2531595`. [p61, 62]

E. Demidenko. Confidence intervals and bands for the binormal ROC curve revisited. *Journal of Applied Statistics*, 39(1):67–79, 2012. URL `https://doi.org/10.1080/02664763.2011.578616`. [p55, 58, 59, 64, 70]

R. DerSimonian and N. Laird. Meta-analysis in clinical trials. *Controlled Clinical Trials*, 7(3):177–188, 1986. URL `https://doi.org/10.1016/0197-2456(86)90046-2`. [p60]

R. Fluss, D. Faraggi, and B. Reiser. Estimation of the Youden Index and its Associated Cutoff Point. *Biometrical Journal*, 47(4):458–472, 2005. URL `https://doi.org/10.1002/bimj.200410135`. [p55]

L. Gonçalvez, A. Subtil, M. R. Oliveira, and P. de Zea Bermudez. ROC curve estimation: An overview. *REVSTAT Statistical Journal*, 12(1):1–20, 2014. [p55]

T. H. Hamza, J. B. Reitsma, and T. Stijnen. Meta-Analysis of Diagnostic Studies: A Comparison of Random Intercept, Normal-Normal, and Binomial-Normal Bivariate Summary ROC Approaches. *Medical Decision Making*, 28(5):639–649, 2008. URL https://doi.org/10.1177/0272989X08323917. [p60]

P. Heagerty and Y. Zheng. Survival model predictive accuracy and ROC curves. *Biometrics*, 61(1): 92–105, 2005. URL https://doi.org/10.1111/j.0006-341X.2005.030814.x. [p55, 59]

P. Heagerty, T. Lumley, and M. S. Pepe. Time-dependent ROC curves for censored survival data and a diagnostic marker. *Biometrics*, 56(2):337–344, 2000. URL https://doi.org/10.1111/j.0006-341X.2000.00337.x. [p55, 59]

P. J. Heagerty and packaging by Paramita Saha-Chaudhuri. *survivalROC: Time-dependent ROC curve estimation from censored survival data*, 2013. URL https://CRAN.R-project.org/package=survivalROC. R package version 1.0.3. [p56]

L. Horváth, Z. Horváth, and W. Zhou. Confidence bands for ROC curves. *Journal of Statistical Planning and Inference*, 138(6):1894–1904, 2008. URL https://doi.org/10.1016/j.jspi.2007.07.009. [p55]

A. Hoyer and O. Kuss. Meta-analysis for the comparison of two diagnostic tests to a common gold standard: A generalized linear mixed model approach. *Statistical Methods in Medical Research*, 27(5): 1410–1421, 2018. URL https://doi.org/10.1177/0962280216661587. [p60]

S. M. Iacus. *sde: Simulation and Inference for Stochastic Differential Equations*, 2016. URL https://CRAN.R-project.org/package=sde. R package version 2.0.15. [p59]

K. Jensen, H.-H. Müller, and H. Schäfer. Regional confidence bands for ROC curves. *Statistics in Medicine*, 19(4):493–509, 2000. URL https://doi.org/10.1002/(SICI)1097-0258(20000229)19:4<493::AID-SIM352>3.0.CO;2-W. [p55, 58, 59, 64, 70]

L. Li, C. W. Department of Biostatistics, and The University of Texas MD Anderson Cancer Center. *tdROC: Nonparametric Estimation of Time-Dependent ROC Curve from Right Censored Survival Data*, 2016. URL https://CRAN.R-project.org/package=tdROC. R package version 1.0. [p59]

L. Li, T. Greene, and B. Hu. A simple method to estimate the time-dependent receiver operating characteristic curve and the area under the curve with right censored data. *Statistical Methods in Medical Research*, 27(8):2264–2278, 2018. URL https://doi.org/10.1177/0962280216680239. [p59, 60]

M. Lopez-Raton and M. X. Rodriguez-Alvarez. *OptimalCutpoints: Computing optimal cutpoints in diagnostic tests*, 2014. URL https://CRAN.R-project.org/package=OptimalCutpoints. R package version 1.1-3. [p56]

S. Macskassy, F. Provost, and S. Rosset. ROC Confidence Bands: An Empirical Evaluation. *Proceedings of the 22nd international conference on machine learning*, pages 537–544, 2005. URL https://doi.org/10.1145/1102351.1102419. [p58]

P. Martínez-Camblor. Fully non-parametric receiver operating characteristic curve estimation for random-effects meta-analysis. *Statistical Methods in Medical Research*, 26(1):5–20, 2017. URL https://doi.org/10.1177/0962280214537047. [p56, 60, 68, 70]

P. Martínez-Camblor and N. Corral. A general bootstrap algorithm for hypothesis testing. *Journal of Statistical Planning and Inference*, 142(2):589–600, 2012. URL https://doi.org/10.1016/j.jspi.2011.09.003. [p61]

P. Martínez-Camblor and J. C. Pardo-Fernández. Parametric estimates for the receiver operating characteristic curve generalization for non-monotone relationships. *Statistical Methods in Medical Research*, (Published online), 2017. URL https://doi.org/10.1177/0962280217747009. [p57]

P. Martínez-Camblor, C. Carleos, and N. Corral. Powerful nonparametric statistics to compare k independent ROC curves. *Journal of Applied Statistics*, 38(7):1317–1332, 2011. URL https://doi.org/10.1080/02664763.2010.498504. [p56, 61, 62, 70]

P. Martínez-Camblor, C. Carleos, and N. Corral. General nonparametric ROC curve comparison. *Journal of the Korean Statistical Society*, 42(1):71–81, 2013. URL https://doi.org/10.1016/j.jkss.2012.05.002. [p56, 61, 70]

P. Martínez-Camblor, G. F-Bayón, and S. Pérez-Fernández. Cumulative/dynamic ROC curve estimation. *Journal of Statistical Computation and Simulation*, 86(17):3582–3594, 2016. URL https://doi.org/10.1080/00949655.2016.1175442. [p59, 70]

P. Martínez-Camblor, N. Corral, C. Rey, J. Pascual, and E. Cernuda-Morollón. Receiver operating characteristic curve generalization for non-monotone relationships. *Statistical Methods in Medical Research*, 26(1):113–123, 2017. URL https://doi.org/10.1177/0962280214541095. [p56, 57, 70]

P. Martínez-Camblor, S. Pérez-Fernández, and N. Corral. Efficient nonparametric confidence bands for receiver operating-characteristic curves. *Statistical Methods in Medical Research*, 27(6):1892–1908, 2018. URL https://doi.org/10.1177/0962280216672490. [p55, 58, 59, 64, 70]

L. E. Moses, D. Shapiro, and B. Littenberg. Combining independent studies of a diagnostic test into a summary ROC curve: Data-analytic approaches and some additional considerations. *Statistics in Medicine*, 12(14):1293–1316, 1993. URL https://doi.org/10.1002/sim.4780121403. [p60]

D. Mossman. Three-way ROCs. *Medical Decision Making*, 19(1):78–89, 1999. URL https://doi.org/10.1177/0272989X9901900110. [p55]

M. S. Pepe. *The Statistical Evaluation of Medical Tests for Classification and Prediction*. Oxford: Oxford University Press, 2003. ISBN 9780198509844. [p55]

S. Pérez-Fernández. *nsROC: Non-Standard ROC Curve Analysis*, 2018. URL https://CRAN.R-project.org/package=nsROC. R package version 1.1. [p56]

E. Peter. *fbroc: Fast Algorithms to Bootstrap Receiver Operating Characteristics Curves*, 2016. URL https://CRAN.R-project.org/package=fbroc. R package version 0.4.0. [p56]

R. D. Riley, P. C. Lambert, and G. Abo-Zhaid. Meta-analysis of individual participant data: rationale, conduct, and reporting. *British Medical Journal*, 340, 2010. URL https://doi.org/10.1136/bmj.c221. [p60]

X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J.-C. Sanchez, M. Müller, and S. Siegert. *pROC: Display and Analyze ROC Curves*, 2018. URL https://CRAN.R-project.org/package=pROC. R package version 1.13.0. [p56]

M. X. Rodríguez-Álvarez, P. G. Tahoces, C. Cadarso-Suárez, and M. J. Lado. Comparative study of ROC regression techniques - Applications for the computer-aided diagnostic system in breast cancer detection. *Computational Statistics and Data Analysis*, 55(1):888–902, 2011. URL https://doi.org/10.1016/j.csda.2010.07.018. [p56]

C. M. Rutter and C. A. Gatsonis. A hierarchical regression approach to meta-analysis of diagnostic test accuracy evaluations. *Statistics in Medicine*, 20(19):2865–2884, 2001. URL https://doi.org/10.1002/sim.942. [p60]

M. C. Sachs. *plotROC: Generate Useful ROC Curve Charts for Print and Interactive Use*, 2018. URL https://CRAN.R-project.org/package=plotROC. R package version 2.2.1. [p56]

I. Schiller and N. Dendukuri. *HSROC: Meta-Analysis of Diagnostic Test Accuracy when Reference Test is Imperfect*, 2015. URL https://CRAN.R-project.org/package=HSROC. R package version 2.1.8. [p56]

T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer. *ROCR: Visualizing the Performance of Scoring Classifiers*, 2015. URL https://CRAN.R-project.org/package=ROCR. R package version 1.0-7. [p56]

S. Steinhauser, M. Schumacher, and G. Rücker. Modelling multiple thresholds in meta-analysis of diagnostic test accuracy studies. *BMC Medical Research Methodology*, 16(97):1–15, 2016. URL https://doi.org/10.1186/s12874-016-0196-1. [p60]

T. M. Therneau. *survival: Survival Analysis*, 2018. URL https://CRAN.R-project.org/package=survival. R package version 2.43-3. [p60]

T. M. Therneau and P. M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. ISBN 9780387987842. [p68]

E. Venkatraman. A permutation test to compare receiver operating characteristic curves. *Biometrics*, 56 (4):1134–1138, 2000. URL https://doi.org/10.1111/j.0006-341X.2000.01134.x. [p56, 61, 62]

E. Venkatraman and C. B. Begg. A distribution-free procedure for comparing receiver operating characteristic curves from a paired experiment. *Biometrika*, 83(4):835–848, 1996. URL https://doi.org/10.1093/biomet/83.4.835. [p56, 61]

X.-H. Zhou, N. A. Obuchowski, and D. K. McClish. *Statistical Methods in Diagnostic Medicine*. Wiley, 2002. ISBN 9780471347729. URL https://doi.org/10.1002/9780470317082. [p55]

Sonia Pérez Fernández
Department of Statistics and Operational Research and Mathematics Didactics
University of Oviedo
Spain
perezsonia@uniovi.es

Pablo Martínez Camblor
The Dartmouth Institute for Health Policy and Clinical Practice
Geisel School of Medicine at Dartmouth
Hanover, NH, USA
Pablo.Martinez.Camblor@Dartmouth.edu

Peter Filzmoser
Institute of Statistics and Mathematical Methods in Economics
Vienna University of Technology
Austria
P.Filzmoser@tuwien.ac.at

Norberto Corral
Department of Statistics and Operational Research and Mathematics Didactics
University of Oviedo
Spain
norbert@uniovi.es

addhaz: Contribution of Chronic Diseases to the Disability Burden Using R

by Renata Tiene de Carvalho Yokota, Caspar WN Looman, Wilma Johanna Nusselder, Herman Van Oyen and Geert Molenberghs

Abstract The increase in life expectancy followed by the burden of chronic diseases contributes to disability at older ages. The estimation of how much chronic conditions contribute to disability can be useful to develop public health strategies to reduce the burden. This paper introduces the R package **addhaz**, which is based on the attribution method (Nusselder and Looman, 2004) to partition disability into the additive contributions of diseases using cross-sectional data. The R package includes tools to fit the additive hazard model, the core of the attribution method, to binary and multinomial outcomes. The models are fitted by maximizing the binomial and multinomial log-likelihood functions using constrained optimization. Wald and bootstrap confidence intervals can be obtained for the parameter estimates. Also, the contribution of diseases to the disability prevalence and their bootstrap confidence intervals can be estimated. An additional feature is the possibility to use parallel computing to obtain the bootstrap confidence intervals. In this manuscript, we illustrate the use of **addhaz** with several examples for the binomial and multinomial models, using the data from the Brazilian National Health Survey, 2013.

Introduction

The increase in longevity observed worldwide is usually followed by the burden of chronic diseases, which are among the leading causes of disability late in life (Beard et al., 2016). Disability has become a public health priority due to its adverse effects on health outcomes and quality of life, resulting in increased costs of health care (Yang et al., 2014). Therefore, the identification of which chronic diseases are the main contributors to the disability burden plays an important role in the formulation of public health response to population aging (Klijs et al., 2011).

Although prospective studies are better suited to establish the causal relationship between chronic diseases and disability, they are costly and usually with limited sample size. Alternatively, cross-sectional data has been widely used to investigate the association of chronic diseases and disability. Among the methods based on cross-sectional data, the attribution method proposed by Nusselder and Looman (2004) has the advantage of partitioning the disability prevalence into the additive contributions of chronic diseases, taking into account multimorbidity and that disability can be present even in the absence of chronic diseases. The method was originally proposed for binary outcomes, but it was recently extended to multicategory response variables (Yokota et al., 2017) and it is based on the binomial and multinomial additive hazard models, respectively. The use of non-canonical link functions in the models imposes a constraint on the linear predictor, which limits the use of standard statistical software to fit the models, such as `glm` in R or `proc GLM` in SAS (SAS Institute Inc., 2008). Despite this practical difficulty, the attribution method for binary outcomes has been widely used previously with data from the Netherlands (Nusselder and Looman, 2004; Klijs et al., 2011), Belgium (Nusselder et al., 2005; Yokota et al., 2015b), Germany (Strobl et al., 2013), China (Chen et al., 2013), and Brazil (Yokota et al., 2016), owing to the development of the software in R to fit the binomial model and to estimate the contribution of diseases to the disability prevalence by Nusselder and Looman (2010) for non-R users, which is available upon request to the authors (w.nusselder@erasmusmc.nl).

In this manuscript we present the R package **addhaz**, which is an extension of the R software developed by Nusselder and Looman (2010), offering an open-source implementation of the binomial and multinomial additive hazard models. The R functions can also be used to calculate the contribution of chronic diseases to the disability burden for both models.

This paper is structured as follows. In Section 2, a brief description of the attribution method is presented, followed by the definition of the binomial and multinomial additive hazard models. Section 3 introduces some features and options of **addhaz**. The existing alternative software to fit the binomial and multinomial models is discussed in Section 4. Examples of how to use the R functions to fit the models and to estimate contributions are shown in Section 5, using the data of the 2013 Brazilian National Health Survey (BNHS). The main advantages and disadvantages of the attribution method and **addhaz** are discussed in Section 6. Finally, conclusions and recommendations for future research are outlined in Section 7.

Attribution method

Analogous to the mortality analysis, in which a single disease is assigned as underlying cause of death in the death certificate, the attribution method aims to assess the probability that a single reported disease was the cause of disability in a survey, taking into account that individuals can report more than one disease (multimorbidity) and that disability can be present without any reported diseases (Nusselder and Looman, 2004, 2010).

In the attribution method, the disability that is not associated with any disease included in the analysis is attributed to "background". The background can represent the effect of age-related losses in functioning; underreporting and underdiagnosed diseases; and other causes of disability that were not included in the survey or analysis. More details about the attribution method can be found elsewhere (Nusselder and Looman, 2004, 2010).

The following assumptions are required to fit the binomial and multinomial additive hazard models to cross-sectional data: (i) disability is caused by the diseases that are still present in the time of the survey and the background; (ii) the estimated cross-sectional cumulative rates reflect the transition rates that would have been estimated with longitudinal data (stationarity assumption); (iii) the recovery rate is zero; (iv) the ratio of the cause-specific cumulative rates to the overall cumulative rate is constant over time (proportionality assumption); (v) the start of the time (age) at risk to become disabled is the same for all diseases; (vi) individuals from the same age group are exposed to the same cumulative rate of disability for background; (vii) diseases and background act as independent competing causes of disability (Nusselder and Looman, 2004, 2010).

Binomial additive hazard model

For binary outcomes, the binomial additive hazard model is defined as:

$$\begin{aligned} y_i &\sim Bernoulli(\pi_i) \\ \pi_i &= 1 - \exp(-\eta_i) \\ \eta_i &= \alpha_{a_i} + \sum_{d=1}^{m} \beta_d X_{di} \end{aligned} \qquad (1)$$

where y_i is the binary disability outcome; π_i is the disability probability for individual i; η_i is the linear predictor representing the overall cumulative rate (or cumulative hazard) of disability for individual i; a_i denotes the age group of individual i (with f age groups, a_i can only get values between $1, \ldots, f$); α_a is the cumulative rate of disability for background by age group a ($a = 1, \ldots, f$); β_d is the cumulative rate of disability (or disabling impact) for disease $d(1, \ldots, m)$; and X_{di} is the indicator variable for disease d and individual i.

A linear inequality constraint is applied to the linear predictor ($\eta_i \geq 0$) to ensure that π_i lies between $(0, 1)$. The standard errors (SE) for the regression coefficients are estimated based on the inverse of the observed information matrix. The 95% Wald confidence intervals (Wald CI) can be obtained using the standard errors described above (Wald CI) as showed in 2 or via bootstrapping (Efron and Tibshirani, 1994).

$$\begin{aligned} \text{95\% Wald CI} &= \widehat{\alpha}_a \pm 1.96(\widehat{SE}) \\ \text{95\% Wald CI} &= \widehat{\beta}_d \pm 1.96(\widehat{SE}) \end{aligned} \qquad (2)$$

Multinomial additive hazard model

The multinomial additive hazard model is an extension of the binomial model:

$$\begin{aligned} y_{ij} &\sim Multinomial(1, \Pi_i) \\ \pi_{ij} &= \left[1 - \exp(-\sum_{q=1}^{c} \eta_{iq})\right] \left(\frac{\eta_{ij}}{\sum_{q=1}^{c} \eta_{iq}}\right) \\ \eta_{ij} &= \alpha_{a_i j} + \sum_{d=1}^{m} \beta_{dj} X_{di} \end{aligned} \qquad (3)$$

where y_{ij} is the multinomial response variable (disability) with one independent observation and vector of disability probabilities $\Pi_i = (\pi_{i0}, \ldots, \pi_{ij}, \ldots, \pi_{ic})$ per individual i; π_{ij} is the probability of disability category j for individual i; η_{ij} is the linear predictor (overall cumulative rate) for disability category j and individual i; a_i denotes the age group of individual i (with f age groups, a_i can only get values between $1, \ldots, f$); α_{aj} is the cumulative rate of disability category j for background by age group a $(a = 1, \ldots, f)$; β_{dj} is the cumulative rate of disability category j or disabling impact for disease $d(1, \ldots, m)$; and X_{di} is the indicator variable for disease d and individual i.

Besides the inequality constraint in the linear predictor $\eta_{ij} \geq 0$, an additional constraint is required: $\sum_{j=1}^{c} \pi_{ij} < 1$, to ensure that all $\pi_{ij} > 0$. Similar to the binomial case, the standard errors are estimated by the inverse of the observed information matrix and the 95% Wald CI and bootstrap percentile confidence intervals (bootstrap CI) can be obtained using **addhaz**.

Contribution of chronic diseases and background to the disability prevalence

The attribution of disability to chronic diseases depends on the disease prevalence (X_d) and the disabling impacts of the diseases (β_d and β_{dj}) (Nusselder and Looman, 2004, 2010). The contribution of chronic diseases and background to the disability prevalence can be calculated in five steps for both binary and multicategory response variables.

Binary case

For the binary case, the cause-specific disability probabilities for individual i and each chronic condition ($\widehat{D}_{di}$) and the background ($\widehat{B}_i$) are estimated based on the proportionality assumption, analogous to the competing risks setting in mortality analysis (Manton and Stallard, 1984; Chiang, 1991):

$$\begin{aligned} \widehat{D}_{di} &= \widehat{\pi}_i \left(\frac{\widehat{\beta}_d X_{di}}{\widehat{\eta}_i} \right) \\ \widehat{B}_i &= \widehat{\pi}_i \left(\frac{\widehat{\alpha}_{ai}}{\widehat{\eta}_i} \right) \end{aligned} \tag{4}$$

Next, the number of disabled individuals by each disease ($\widehat{N}_d$) and background ($\widehat{N}_b$) are estimated as:

$$\begin{aligned} \widehat{N}_d &= \sum_{i=1}^{n} \widehat{D}_{di} \\ \widehat{N}_b &= \sum_{i=1}^{n} \widehat{B}_i \end{aligned} \tag{5}$$

The absolute contribution of each disease ($\widehat{AC}_d$) and background ($\widehat{AC}_b$) to the total disability prevalence is obtained by:

$$\begin{aligned} \widehat{AC}_d &= \frac{\widehat{N}_d}{n} \\ \widehat{AC}_b &= \frac{\widehat{N}_b}{n} \end{aligned} \tag{6}$$

The absolute contribution of background and diseases defined above sum to the disability prevalence ($\widehat{P}$):

$$\widehat{P} = \widehat{AC}_b + \sum_{d=1}^{m} \widehat{AC}_d \tag{7}$$

Finally, the relative contribution of diseases ($\widehat{RC}_d$) and background ($\widehat{RC}_b$) to the disability prevalence is estimated by:

$$\begin{aligned} \widehat{RC}_d &= \frac{\widehat{AC}_d}{\widehat{P}} \\ \widehat{RC}_b &= \frac{\widehat{AC}_b}{\widehat{P}} \end{aligned} \tag{8}$$

The relative contributions ($\widehat{RC}_d$ and $\widehat{RC}_b$) sum to 1.

Multinomial case

Analogous to the binomial case, the contribution of chronic diseases to the disability prevalence for multinomial outcomes can be obtained in five steps for each category j of the outcome:

1. Cause-specific disability probabilities for each disease ($\widehat{D}_{dij}$) and background ($\widehat{B}_{ij}$) for individual i:

$$\widehat{D}_{dij} = \widehat{\pi}_{ij} \left(\frac{\widehat{\beta}_{dj} X_{di}}{\widehat{\eta}_{ij}} \right)$$
$$\widehat{B}_{ij} = \widehat{\pi}_{ij} \left(\frac{\widehat{\alpha}_{a_i j}}{\widehat{\eta}_{ij}} \right) \tag{9}$$

2. Number of disabled individuals by each disease ($\widehat{N}_{dj}$) and background ($\widehat{N}_{bj}$):

$$\widehat{N}_{dj} = \sum_{i=1}^{n} \widehat{D}_{dij}$$
$$\widehat{N}_{bj} = \sum_{i=1}^{n} \widehat{B}_{ij} \tag{10}$$

3. Absolute contribution of each disease ($\widehat{AC}_{dj}$) and background ($\widehat{AC}_{bj}$) to the total disability prevalence:

$$\widehat{AC}_{dj} = \frac{\widehat{N}_{dj}}{n}$$
$$\widehat{AC}_{bj} = \frac{\widehat{N}_{bj}}{n} \tag{11}$$

4. Total disability prevalence ($\widehat{P}_j$):

$$\widehat{P}_j = \widehat{AC}_{bj} + \sum_{d=1}^{m} \widehat{AC}_{dj} \tag{12}$$

5. Relative contribution of diseases ($\widehat{RC}_{dj}$) and background ($\widehat{RC}_{bj}$) to the disability prevalence:

$$\widehat{RC}_{dj} = \frac{\widehat{AC}_{dj}}{\widehat{P}_j}$$
$$\widehat{RC}_{bj} = \frac{\widehat{AC}_{bj}}{\widehat{P}_j} \tag{13}$$

The absolute contributions defined in equations 11 sum to the prevalence of disability for each category j and the relative contributions defined in equations 13 sum to 1 for each disability category j. The confidence intervals for the absolute and relative contributions for the binary and multinomial cases can be estimated via bootstrapping (Efron and Tibshirani, 1994) in **addhaz**.

Features of addhaz

In this section, a brief explanation about the constrained optimization, the parallel option to obtain the bootstrap CI for the parameter estimates, and the option to perform the likelihood ratio test for model selection is provided.

Constrained optimization

The binomial and multinomial additive hazard models are generalized linear models with non-canonical link functions $\eta_i = \log\left(\frac{1}{1-\pi_i}\right)$ for the binomial model and $\eta_{ij} = [-\log(1 - \sum_{q=1}^{c} \pi_{iq})] \left(\frac{\pi_{ij}}{\sum_{q=1}^{c} \pi_{iq}}\right)$ for the multinomial model. For both models, the model parameters are estimated using maximum likelihood. The use of non-canonical link functions requires a constraint in the linear predictors ($\eta_i \geq 0$ and $\eta_{ij} \geq 0$) to ensure that the disability probabilities (π_i and π_{ij}) are valid, i.e., the probabilities lie between 0 and 1. In **addhaz**, this constraint is implemented in the optimization procedure, with an adaptive barrier algorithm (Lange, 2010), by calling `constrOptim` in R.

Parallel computing for the bootstrap CI

Besides the option to estimate the confidence intervals for the parameter estimates based on the standard errors obtained by the inverse of the information matrix for the binomial and multinomial models (Wald CI), **addhaz** also offers the user the option to obtain the bootstrap CI based on empirical percentiles of the replicates (Efron and Tibshirani, 1994).

To reduce computation time, parallel computing can be used to obtain the bootstrap CI. By default R does not use all the cores available on a computer. The basic idea of parallel computing is to split the work to more than one core of the computer, to execute it in parallel, and then to collect the results. Several R packages can be used to implement parallel computation. In **addhaz** it is implemented by calling the functions `boot` and `boot.ci` in the **boot** package (Canty and Ripley, 2016; Davison and Hinkley, 1997).

Likelihood ratio test

The package also includes a function to perform the likelihood ratio test to compare two binomial or multinomial nested models that can be used for model selection.

The likelihood ratio test is defined as -2*log(likelihood model 1/likelihood model 2).The resulting test statistic is assumed to follow a χ^2 distribution, with degrees of freedom (df) equal to the difference of the df between the models. If the test is statistically significant, the model with more variables fits the data significantly better than the model with less variables.

Alternative software

The original software for the attribution method (Nusselder and Looman, 2004, 2010) was developed in R, but it is not available as an R package, since it focuses on non-R users: an Excel file is used to input the model arguments and this file is called in the R code. This software is restricted to binary outcomes and it is freely available upon request to the authors. Different from **addhaz**, a penalty term is added to the binomial likelihood function when $\pi_i \leq 0$, to ensure that valid probabilities are obtained. The original software also allows the user to obtain the bootstrap CI for the model parameters and contributions. Additionally, it offers the options: (i) reduced rank regression (RRR) (Yee, 2014) to reduce the number of parameters when interactions between diseases and age groups are of interest; and (ii) model selection, using the likelihood ratio test.

Besides the original software, the parameter estimates of the binomial additive hazard model can be obtained using the R packages **stats** with `glm` and **logbin** with the function `logbin` (Donoghoe, 2016). In both packages, the log-binomial model, i.e., $\pi_i = \exp(\eta_i)$, used to estimate relative risks (Marschner and Gillett, 2012), can be fitted to a transformed version of the response variable $y^* = 1 - y$, with the log link function. The estimated model parameters should be multiplied by -1, since $1 - \pi_i = \exp(-\eta_i)$. However, care should be taken with `glm`: by specifying the option `family = binomial(link = log)` to fit the log-binomial model, convergence failure may occur with the iterative weighted least squares (IWLS) algorithm when the maximum likelihood estimates (MLE) lie on the boundary of the parameter space. In `glm`, the IWLS is modified so that if constraints are violated, step-halving is used iteratively until they are respected. Although this should not result in invalid estimates, it may cause difficulty in convergence. An advantage of **logbin** is that it includes constrained optimization as an option to optimize the binomial log-likelihood function `(method = "ab")`. This is done by calling `constrOptim` in R to constrain the parameter space.

Since **addhaz** was developed with focus on the attribution method, apart from estimating the model parameters for the binomial additive hazard model, it also gives the user the option to obtain the contribution of diseases to the disability prevalence and to obtain bootstrap CI for the parameter estimates and the contributions, using parallel computing to reduce computation time.

To our knowledge, there is no other package available to fit the multinomial additive hazard model. Although it is possible to fit the log-multinomial model (Blizzard and Hosmer, 2007), i.e., $\pi_{ij} = \exp(\eta_{ij})$, with the function `vglm` in **VGAM** (Yee, 2016), different from the binomial case, no simple transformation of the outcome can be applied to obtain the parameter estimates of the multinomial additive hazard model using the log-multinomial model.

Using the package addhaz

In this section, the use of the functions `BinAddHaz`, `MultAddHaz`, and `LRTest` in **addhaz** are illustrated using a subset of the 2013 BNHS available in the package. A selected output of the results is shown.

Data description

The Brazilian National Health Survey (BNHS) (“Pesquisa Nacional de Saúde") was a nationally representative survey of the Brazilian adult population ($\geq$ 18 years) with approximately 60,000 individuals, conducted in 2013. A multistage sampling design with simple random sampling (census tracts) and clustering (households and adults) was used. The response rate was 77%. Survey weights were included to take into account the complex design of the sample. Detailed information about the BNHS can be found in previous publications (Szwarcwald et al., 2014; Yokota et al., 2016).

In **addhaz**, a subset of the BNHS data is available, with women aged $\geq$ 60 years (n = 6,294) and the following variables: disability as binary and multinomial outcomes, survey weight, age, diabetes, arthritis, and stroke (Table 1).

Variable name	Definition	Type	Categories
wgt	survey weight	continuous	-
age	age group	binary	0: 60-79 years 1: $\geq$80 years
diab	diabetes	binary	0: no 1: yes
arth	arthritis	binary	0: no 1: yes
stro	stroke	binary	0: no 1: yes
dis.bin	binary disability outcome	binary	0: no disability 1: disabled
dis.mult	multinomial disability outcome	categorical	0: no disability 1: mild disability 2: severe disability

Table 1: *Description of the variables included in the analysis. Brazilian National Health Survey, 2013.*

The binomial and multinomial disability outcomes were defined based on five instrumental activities of daily living (IADL): shopping, handling finances, taking own medications, going to the doctor, and using transportation. Participants were asked about the degree of difficulty in performing IADL tasks, with possible answers: “1. Unable", “2. A lot of difficulty", “3. Some difficulty", or “4. No difficulty". The definition of the binary and multinomial outcomes is shown in Table 2. The reference category “No disability" represents answer “4" to all IADL questions.

Outcome	Outcome category	Answer to at least one IADL question
Binary	Disabled	1, 2 or 3
Multinomial	Mild disability	3
	Severe disability	1 or 2

Table 2: *Definition of the binary and multinomial disability outcomes. Brazilian National Health Survey, 2013. “IADL” refers to instrumental activities of daily living.*

A summary of the characteristics of the study population is shown in Table 3. A higher proportion of older women ($\geq$80 years), diabetes, arthritis, stroke, and the disease pairs was observed in women with mild and severe disability compared to women without disability.

Examples with binary outcomes

The function `BinAddHaz` fits the binomial additive hazard model and estimates the contribution of diseases to the disability burden for binary outcomes in **addhaz**. To illustrate the use of `BinAddHaz`, five models were fitted with the binary disability outcome: model 1 - with three diseases (no background and diseases by age); model 2 - with only background by age, with bootstrap CI; model 3 - with only diseases by age; model 4 - with background and diseases by age, with bootstrap CI; model 5 - with two-way interaction between diseases. To illustrate how the `LRTest` function can be used for model selection, models 2 and 4 are compared.

Characteristic	Total		No disability		Mild disability		Severe disability	
	N	%	N	%	N	%	N	%
Age (years)								
60-79	5379	85.5	3946	93.5	642	78.3	791	63.0
≥80	915	14.5	273	6.5	178	21.7	464	37.0
Diseases								
Diabetes	1243	19.7	697	16.5	190	23.2	356	28.4
Arthritis	1428	22.7	819	19.4	211	25.7	398	31.7
Stroke	286	4.5	100	2.4	41	5.0	145	11.6
Diabetes and arthritis	298	4.7	135	3.2	53	6.5	110	8.8
Diabetes and stroke	91	1.4	31	0.7	13	1.6	47	3.7
Arthritis and stroke	79	1.3	28	0.7	7	0.9	44	3.5

Table 3: *Characteristics of the study population. Brazilian National Health Survey, 2013. The percentages refer to unweighted proportions, i.e., without taking into account the survey weights.*

Model 1 - Binomial model with three diseases

Before fitting model 1, **addhaz** and the data can be loaded and the names of the variables can be checked using:

```
library("addhaz")
data("disabData")
names(disabData)

[1] "dis.bin" "dis.mult" "wgt" "age" "diab" "arth" "stro"
```

The first binomial model can be fitted with:

```
model1 <- BinAddHaz(dis.bin ~ diab + arth + stro , data = disabData, weights = wgt,
                   attrib = TRUE, type.attrib = "both", collapse.background = FALSE,
                   attrib.disease = FALSE)
```

Since no attribution variable such as age was included in the model, the arguments `collapse.background` and `attrib.disease` were set to `FALSE`. The results of the model were stored as an object called `model1`, which can be checked with the `summary` function:

```
summary(model1)

$`call`
BinAddHaz(formula = dis.bin ~ diab + arth + stro, data = disabData,
         weights = wgt, attrib = TRUE, collapse.background = FALSE,
         attrib.disease = FALSE, type.attrib = "both")

$bootstrap
[1] FALSE

$coefficients
             Estimate      StdErr   t.value       p.value
(Intercept) 0.2970833 0.009426403 31.516082 1.498823e-202
diab        0.1331831 0.023821666  5.590838  2.354697e-08
arth        0.1306445 0.022203662  5.883917  4.212860e-09
stro        0.5927519 0.075697633  7.830521  5.663378e-15

attr(,"class")
[1] "summary.binaddhazmod"
```

The first element of the output `call` is the formula used to fit the model. The `bootstrap`, indicates if the bootstrap CI was requested. Since it was not requested, its value is `FALSE`.

Next, the coefficients are printed, with their estimates, standard errors (calculated based on the inverse of the observed information matrix), the t value (value of the t statistic), and the p value. The intercept represents the background. According to this output, all diseases were significant in the model. To identify the most disabling diseases, i.e., the diseases with highest cumulative rate of disability, the coefficients can be sorted in decreasing order using:

```
sort(model1$coefficients, decreasing = TRUE)

   stro    (Intercept)    diab        arth
0.5927519   0.2970833   0.1331831   0.1306445
```

Stroke was the most disabling disease, while arthritis was the least disabling disease. The 95% Wald CI can be obtained by:

```
model1$ci

                 CI2.5     CI97.5
(Intercept) 0.27860754 0.3155590
diab        0.08649261 0.1798735
arth        0.08712532 0.1741637
stro        0.44438455 0.7411193
```

Both the relative and absolute contributions were requested (`attrib = TRUE,type.attrib = "both"`) and can be assessed with:

```
model1$contribution

$`att.rel`
               att.rel
(Intercept) 0.80405374
diab        0.06938567
arth        0.07451155
stro        0.05204903

$att.abs
               att.abs
disab       0.30853338
(Intercept) 0.24807742
diab        0.02140780
arth        0.02298930
stro        0.01605886
```

In the above output, the relative contribution (`att.rel`: the contributions sum to 1) is shown at the top and the absolute contribution (`att.abs`: the contributions sum to the disability prevalence) is presented at the bottom. No confidence intervals are provided for the contributions, as they can only be calculated via bootstrapping and this option was not requested.

In the output for the absolute contribution, the disability prevalence (`disab`) was 30.9%. The absolute contribution can be sorted in decreasing order using:

```
model1$contribution$att.abs[order(model1$contribution$att.abs[, 1], decreasing = TRUE), ]

    disab   (Intercept)     arth       diab        stro
0.30853338  0.24807742  0.02298930  0.02140780  0.01605886
```

The background (`Intercept`) was the main contributor to the disability burden in this population. In this case, it can represent other causes not included in the model such as dementia or back pain, which are important causes of disability in the older population, but were not included in the analysis. Among the three diseases, arthritis was the main contributor to the disabilitiy burden in older Brazilian women.

It is interesting to note that, although stroke was the most disabling disease, it showed the lowest contribution to the total disability prevalence. This low contribution can be a consequence of the low occurrence of stroke in this population - 4.5% (Table 3) - as the contribution of chronic conditions to the disability prevalence depends on both, the disease occurrence and the disabling impact (Nusselder and Looman, 2010).

Model 2 - Binomial model with only background by age, with bootstrap CI

In model 2, the background is modelled by age group, but the diseases are not. The model can be fitted by:

```
model2 <- BinAddHaz(dis.bin ~ factor(age) -1 + diab + arth + stro , data = disabData,
                    weights = wgt, attrib = TRUE, type.attrib = "both",
                    collapse.background = FALSE, attrib.disease = FALSE, seed = 111,
                    bootstrap = TRUE, conf.level = 0.95, nbootstrap = 1000,
                    parallel = TRUE, type.parallel = "snow", ncpus = 4)
```

Since the background is modelled by age group, `factor(age)` is included in the model. The `-1` is included in the `model.formula` to obtain the coefficients for both age groups, including the reference category. Since the background is modelled by age, it should not be collapsed by age (`collapse.background = FALSE`). As no interaction between diseases and age were included in the model, the argument `attrib.disease` is `FALSE`. The option `seed = 111` allows the user to specify a random number to make the results of the bootstrapping reproducible. In the example above, the random number used was 111. The bootstrap CI for the regression coefficients and the contributions was requested (`bootstrap = TRUE`), with confidence level = 0.95 (`conf.level = 0.95`). The bootstrap CI was based on 1000 replicates (`nbootstrap = 1000`) and it was obatined with parallel computing (`parallel = TRUE`) on Windows (`type.parallel = "snow"`) with 4 CPUS (`ncpus = 4`).

The summary of the model can be assessed with:

```
summary(model2)

$`call`
BinAddHaz(formula = dis.bin ~ factor(age) - 1 + diab + arth + stro, data = disabData,
          weights = wgt, attrib = TRUE, type.attrib = "both",
          collapse.background = FALSE, attrib.disease = FALSE, seed = 111,
          bootstrap = TRUE, conf.level = 0.95, nbootstrap = 1000,
          parallel = TRUE, type.parallel = "snow", ncpus = 4)

$bootstrap
[1] TRUE

$coefficients
               Estimate      CILow    CIHigh
factor(age)0 0.22345184 0.19892365 0.2529054
factor(age)1 1.10472873 0.95279272 1.2705886
diab         0.12935797 0.06191508 0.2044457
arth         0.08531865 0.02375110 0.1513319
stro         0.52453664 0.28688675 0.8509963

$conf.level
[1] 0.95

attr(,"class")
[1] "summary.binaddhazmod"
```

Since the bootstrap CI was requested, `bootstrap` is `TRUE`. The coefficients show that age and all diseases were significant (the null value, i.e. 0, is not included in the bootstrap CI). The `factor(age)0` and `factor(age)1` represents the background cumulative rates for age groups 0 and 1, respectively. The contributions can be checked with:

```
model2$contribution

$att.rel
             Contribution      CILow     CIHigh
factor(age)0   0.53992912 0.51937657 0.56054196
factor(age)1   0.29842186 0.27575265 0.32163433
diab           0.06702577 0.06162503 0.07256112
arth           0.04869951 0.04513817 0.05269298
stro           0.04592374 0.03666781 0.05519789

$att.abs
             Contribution      CILow     CIHigh
disab          0.30902546 0.30227641 0.31623119
factor(age)0   0.16685185 0.16405831 0.16947954
factor(age)1   0.09221995 0.08372162 0.10131428
diab           0.02071267 0.01906935 0.02254047
```

```
arth          0.01504939 0.01396744 0.01628476
stro          0.01419161 0.01127267 0.01727091
```

The contributions and the 95% bootstrap CI are shown. The background is the main contributor to the disability prevalence. Note that by allowing the background to vary by age does not change the disability prevalence (30.9%), as compared to model 1.

Model 3 - Binomial model with only diseases by age

In Model 3, we allow only the diseases to vary by age group by including interactions between age and diseases in the model. Before fitting model 3, a matrix with the diseases to be included in the model can be defined by:

```
disease <- as.matrix(disabData[, c("diab", "arth", "stro")])
```

The first six elements of the matrix created can be checked using:

```
head(disease)

    diab arth stro
36     1    0    0
98     0    0    0
113    0    1    1
347    1    0    0
352    1    0    0
436    0    0    0
```

The binomial additive hazard model can be fitted with the function:

```
model3 <- BinAddHaz(dis.bin ~ disease:factor(age), data = disabData, weights = wgt,
                    attrib = TRUE, attrib.var = age, type.attrib = "abs",
                    collapse.background = FALSE, attrib.disease = TRUE)
summary(model3)

$`call`
BinAddHaz(formula = dis.bin ~ disease:factor(age), data = disabData, weights = wgt,
          attrib = TRUE, attrib.var = age, collapse.background = FALSE,
          attrib.disease = TRUE, type.attrib = "abs")

$bootstrap
[1] FALSE

$coefficients
                           Estimate      StdErr    t.value       p.value
(Intercept)              0.29425017 0.009339432 31.5062168 1.991333e-202
diseasediab:factor(age)0 0.07487954 0.022708458  3.2974294  9.811601e-04
diseasearth:factor(age)0 0.01218173 0.020156904  0.6043454  5.456358e-01
diseasestro:factor(age)0 0.44896271 0.072553106  6.1880563  6.474653e-10
diseasediab:factor(age)1 0.83733434 0.128901534  6.4959223  8.884711e-11
diseasearth:factor(age)1 1.32865873 0.143790133  9.2402636  3.299325e-20
diseasestro:factor(age)1 1.60530144 0.373531351  4.2976351  1.752351e-05

attr(,"class")
[1] "summary.binaddhazmod"
```

The (Intercept) represents the background, which was not modelled by age. The diseases with factor(age)0 and factor(age)1 represent the regression coefficients for age 0 (60-79 years) and age 1 ($\geq$80 years). The output above shows that arthritis was not significant for the reference age category (60-79years) (diseasearth:factor(age)0). Only the absolute contribution was requested (type.attrib = "abs") and it can be assessed with:

```
model3$contribution

                               att.abs
disab.0                    0.277835632
backgrnd.0                 0.251005540
```

```
diseasediab:factor(age)0.0 0.012575547
diseasearth:factor(age)0.0 0.002220039
diseasestro:factor(age)0.0 0.012034506
disab.1                    0.493678649
backgrnd.1                 0.206353130
diseasediab:factor(age)1.1 0.089832332
diseasearth:factor(age)1.1 0.158378063
diseasestro:factor(age)1.1 0.039115125
```

The attribution is presented by level of the attribution variable (`attrib.var`), which in this example is age. The first five rows show the results for attribution variable level 0, which in this case represents age group 60-79 years and the last five rows represent the results for attribution variable level 1 ($\geq$80 years). The results indicate that the disability prevalence in the older women (49.4%) was much larger than in the younger age group (27.8%). While the three diseases included in the model showed a low contribution to the disability prevalence in women aged 60-79 years (<1.5%), arthritis was by far the most important disease contributing to the disability prevalence in the oldest women (15.9%).

Model 4 - Binomial model with background and diseases by age, with bootstrap CI

The same matrix of diseases defined for model 3 will be used in model 4. This model can be fitted with the function:

```
model4 <- BinAddHaz(dis.bin ~ factor(age) - 1 + disease:factor(age), data = disabData,
                    weights = wgt, attrib = TRUE, attrib.var = age, type.attrib = "abs",
                    collapse.background = FALSE, attrib.disease = TRUE, seed = 111,
                    bootstrap = TRUE, conf.level = 0.95, nbootstrap = 1000,
                    parallel = TRUE, type.parallel = "snow", ncpus = 4)
```

The `-1` in the `model.formula` is used to obtain a different parameterization than the default: here we obtain the parameter estimates for all the age groups, including the reference category. Since we want to estimate the contributions for background by age, the argument `collapse.background` is set to `FALSE`. If this argument would be set to `TRUE`, only one background, common to all age groups, would be estimated. Since the contributions of diseases should be estimated by age group, the argument `attrib.disease` was set to `TRUE`. The parallel option for the bootstrap CI was used (`parallel = TRUE`) on a Windows computer (`type.parallel = "snow"`) with 4 cores (`ncpus = 4`). The option `seed = 111` allows the user to specify a random number (in this case 111) to make the results of the bootstrapping reproducible. The summary of the model can be checked with:

```
summary(model4)

$call
BinAddHaz(formula = dis.bin ~ factor(age) - 1 + disease:factor(age), data = disabData,
          weights = wgt, attrib = TRUE, attrib.var = age, collapse.background = FALSE,
          attrib.disease = TRUE, type.attrib = "abs", seed = 111, bootstrap = TRUE,
          conf.level = 0.95, nbootstrap = 1000, parallel = TRUE,
          type.parallel = "snow", ncpus = 4)

$bootstrap
[1] TRUE

$coefficients
                           Estimate        CILow    CIHigh
factor(age)0             0.22661055  0.201218693 0.2568179
factor(age)1             0.94910725  0.784733478 1.1292937
factor(age)0:diseasediab 0.12749849  0.056516120 0.2036685
factor(age)1:diseasediab 0.24124648 -0.181208625 0.7471999
factor(age)0:diseasearth 0.06884402  0.009035558 0.1345238
factor(age)1:diseasearth 0.75879140  0.349882903 1.2234047
factor(age)0:diseasestro 0.48788899  0.255772550 0.8331633
factor(age)1:diseasestro 1.13333515  0.426477637 2.2240599

$conf.level
[1] 0.95
```

```
attr(,"class")
[1] "summary.binaddhazmod"
```

The output with the results of the model is shown for `factor(age)0`, which represents the age group of 60-79 years, and `factor(age)1`, representing the age group of ≥80 years. Diabetes was not significant for age group 1, as the bootstrap CI includes the null value, i.e. 0. To identify the most disabling diseases, two objects (`coef.age0` and `coef.age1`) with the coefficients for each age group can be created and sorted in decreasing order using:

```
coef.age0 <- model4$coefficients[seq(1, length(model4$coefficients), 2)]
coef.age1 <- model4$coefficients[seq(0, length(model4$coefficients), 2)]

sort(coef.age0, decreasing = TRUE)
factor(age)0:diseasestro            factor(age)0 factor(age)0:diseasediab
              0.48788899              0.22661055               0.12749849
factor(age)0:diseasearth
              0.06884402

sort(coef.age1, decreasing = TRUE)
factor(age)1:diseasestro            factor(age)1 factor(age)1:diseasearth
               1.1333352               0.9491072                0.7587914
factor(age)1:diseasediab
               0.2412465
```

Stroke was the most disabling disease in both age groups. Arthritis and diabetes showed the lowest disabling impact in women aged 60-79 years and ≥80 years, respectively.

Since only the absolute contribution was requested (`type.attrib = "abs"`), the results for the absolute contribution can be assessed with:

```
model4$contribution

                               att.abs      CILow     CIHigh
disab.0                     0.24442949 0.24102940 0.24813627
backgrnd.0                  0.19743946 0.19694283 0.19790210
factor(age)0:diseasediab.0 0.02141352 0.01956557 0.02342391
factor(age)0:diseasearth.0 0.01253887 0.01153068 0.01363192
factor(age)0:diseasestro.0 0.01303764 0.01003208 0.01636211
disab.1                     0.69484947 0.68537515 0.70574268
backgrnd.1                  0.54868976 0.53993448 0.55643174
factor(age)1:diseasediab.1 0.02637877 0.02080951 0.03249654
factor(age)1:diseasearth.1 0.09137348 0.07746954 0.10759760
factor(age)1:diseasestro.1 0.02840746 0.01932950 0.03894183
```

The CILow and CIHigh refers to the 2.5th and 97.5th percentiles of the 1,000 bootstrap replicates, since the bootstrap CI was requested (`bootstrap = TRUE`) with `conf.level = 0.95`. To identify the main contributors to the disability burden, two objects (one for each age group) can be defined with the absolute contribution and bootstrap CI using:

```
cont.age0 <- model4$contribution[c(1:5), ]
cont.age1 <- model4$contribution[c(6:10), ]
cont.age0[order(cont.age0[, 1], decreasing = TRUE), ]

                               att.abs      CILow     CIHigh
disab.0                     0.24442949 0.24102940 0.24813627
backgrnd.0                  0.19743946 0.19694283 0.19790210
factor(age)0:diseasediab.0 0.02141352 0.01956557 0.02342391
factor(age)0:diseasestro.0 0.01303764 0.01003208 0.01636211
factor(age)0:diseasearth.0 0.01253887 0.01153068 0.01363192

cont.age1[order(cont.age1[, 1], decreasing = TRUE), ]

                               att.abs      CILow     CIHigh
disab.1                     0.69484947 0.68537515 0.70574268
backgrnd.1                  0.54868976 0.53993448 0.55643174
factor(age)1:diseasearth.1 0.09137348 0.07746954 0.10759760
factor(age)1:diseasestro.1 0.02840746 0.01932950 0.03894183
factor(age)1:diseasediab.1 0.02637877 0.02080951 0.03249654
```

According to the results, the disability prevalence in the oldest women (69.5%) was 2.8 times larger than in women aged 60-79 years (24.4%). The background was the main contributor to the disability prevalence in both age groups. Among the chronic conditions, diabetes was the main contributor to the disability prevalence in women aged 60-79 years (2.1%) while arthritis contributed most to the disability burden in older women (9.1%).

Model 5 - Binomial model with two-way interaction between diseases

In model 5, the independence assumption (assumption vii) is violated and two-way interactions between diseases are included in the model. In total, 6 parameters and the intercept will be estimated in model 5. The model can be fitted by:

```
model5 <- BinAddHaz(dis.bin ~ (diab + arth + stro)^2, data = disabData, weights = wgt,
                    attrib = TRUE, collapse.background = FALSE, attrib.disease = FALSE,
                    type.attrib = "both")

summary(model5)

$`call`
BinAddHaz(formula = dis.bin ~ (diab + arth + stro)^2, data = disabData, weights = wgt,
          attrib = TRUE, collapse.background = FALSE, attrib.disease = FALSE,
          type.attrib = "both")

$bootstrap
[1] FALSE

$coefficients
              Estimate      StdErr    t.value       p.value
(Intercept)  0.2988163 0.009586541 31.1703950 1.803828e-198
diab         0.1178815 0.026104065  4.5158287  6.422107e-06
arth         0.1101293 0.023712731  4.6443118  3.481543e-06
stro         0.8145107 0.116867992  6.9694938  3.505379e-12
diab:arth    0.1563155 0.067097034  2.3296932  1.985387e-02
diab:stro   -0.5072111 0.154344770 -3.2862213  1.020980e-03
arth:stro   -0.1494752 0.176853232 -0.8451937  3.980349e-01

attr(,"class")
[1] "summary.binaddhazmod"
```

The main effects of all the diseases were significant, but only the interaction between diabetes and arthritis and between diabetes and stroke were significant. The negative disabling impact of the interaction term between diabetes and stroke should be carefully interpreted, as it is based on a small sample size ($n = 91$) (Table 3).

Likelihood ratio test for model selection

To illustrate the use of the function `LRTest` to perform the likelihood ratio test (LRT) for model selection, models 2 (`model2`) and 4 (`model4`) are compared. The LRT can be performed with:

```
LRTest(model4, model2)

Likelihood ratio test
Model 1:
dis.bin ~ factor(age) - 1 + disease:factor(age)
Model 2:
dis.bin ~ factor(age) - 1 + diab + arth + stro
  Res.df  Res.Dev df Deviance   Pr(>Chi)
1   6286 6916.818
2   6289 6946.224  3   29.405 1.8407e-06
```

The output shows the models that are being compared: `Model 1` is the model with the interactions with diseases and age (previous model 4) and `Model 2` is the model without the interactions between diseases and age (previous model 2). The degrees of freedom for each model (Res.df), the residual deviance, i.e. the 2*log-likelihood of each model (Res.Dev), the difference in the degrees of freedom between the models (df), the difference between the 2*log-likelihood of the models, i.e. the value of the likelihood ratio test statistic (Deviance), and the p-value of the test statistic, based on the χ^2 distribution (Pr(>Chi)) are presented. Since the test was statistically significant at 0.05 significance level, model 4, which includes interaction between diseases and age, fits the data better than model 2.

Examples with multinomial outcomes

To fit the multinomial additive hazard model and to estimate the contribution of chronic conditions to the disability burden for multinomial outcomes, the function MultAddHaz can be used. As an illustration, two models were fitted: model 6 - with only background by age; and model 7 - with background and diseases by age, with bootstrap CI.

Model 6 - Multinomial model with only background by age

Model 6 can be fitted with the function:

```
model6 <- MultAddHaz(dis.mult ~ factor(age) - 1 + diab + arth + stro, data = disabData,
                     weights = wgt, attrib = TRUE, seed = 111, collapse.background = FALSE,
                     attrib.disease = FALSE,  type.attrib = "both")
```

The results of the model can be visualized using:

```
summary(model6)

$`call`
MultAddHaz(formula = dis.mult ~ factor(age) - 1 + diab + arth +
           stro, data = disabData, weights = wgt, attrib = TRUE, seed = 111,
           collapse.background = FALSE, attrib.disease = FALSE, type.attrib = "both")

$bootstrap
[1] FALSE

$coefficients
                   Estimate      StdErr    t.value      p.value
factor(age)0.y1 0.117959944 0.006083174 19.3911842 2.109290e-81
factor(age)1.y1 0.285541582 0.022330639 12.7869869 5.585601e-37
diab.y1         0.002717701 0.011329960  0.2398685 8.104400e-01
arth.y1         0.015602747 0.011534798  1.3526675 1.762105e-01
stro.y1         0.024923984 0.025498946  0.9774515 3.283833e-01
factor(age)0.y2 0.107643802 0.005969496 18.0323096 6.503330e-71
factor(age)1.y2 0.817043178 0.043161129 18.9300697 9.103853e-78
diab.y2         0.124326766 0.017245323  7.2093036 6.283854e-13
arth.y2         0.063767963 0.014095689  4.5239336 6.181719e-06
stro.y2         0.499262854 0.064799754  7.7047030 1.514581e-14

attr(,"class")
[1] "summary.multaddhazmod"
```

In the above output, the results identified with $y1$ refer to the outcome (dis.mult) = 1 (mild disability) and the results with $y2$ refer to the outcome (dis.mult) = 2 (severe disability). For mild disability only the background (factor(age)0.y1) and (factor(age)1.y1) was significant while all the diseases and the background were signifcant for women with severe disability. Similar to the binomial model, the most disabling diseases can be identified by:

```
coef.mild <- model6$coefficients[1:5, ]
coef.sev <- model6$coefficients[6:10, ]

sort(coef.mild, decreasing = TRUE)
factor(age)1.y1 factor(age)0.y1         stro.y1         arth.y1         diab.y1
    0.285541582     0.117959944     0.024923984     0.015602747     0.002717701

sort(coef.sev, decreasing = TRUE)
```

```
factor(age)1.y2         stro.y2         diab.y2 factor(age)0.y2         arth.y2
     0.81704318      0.49926285      0.12432677      0.10764380      0.06376796
```

Background and stroke showed the highest disabling impact for mild and severe disability. The relative and absolute contributions can be checked with:

```
model6$contribution

$`att.rel`
                    att.rel
factor(age)0.y1 0.804099194
factor(age)1.y1 0.164283693
diab.y1         0.003665546
arth.y1         0.023317949
stro.y1         0.004633619
factor(age)0.y2 0.426874738
factor(age)1.y2 0.336655536
diab.y2         0.105566151
arth.y2         0.058984332
stro.y2         0.071919244

$att.abs
                     att.abs
disab.y1        0.1265087428
factor(age)0.y1 0.1017255781
factor(age)1.y1 0.0207833234
diab.y1         0.0004637236
arth.y1         0.0029499244
stro.y1         0.0005861933
disab.y2        0.1901766667
factor(age)0.y2 0.0811816147
factor(age)1.y2 0.0640240276
diab.y2         0.0200762187
arth.y2         0.0112174436
stro.y2         0.0136773621
```

It is interesting to note that the severe disability prevalence (19.0%) was 1.5 times higher than the mild disability prevalence (12.7%). The results for the relative contribution can be sorted in decreasing order by:

```
rel.cont.mild <- model6$contribution$att.rel[1:5, ]
rel.cont.sev <- model6$contribution$att.rel[6:10, ]

sort(rel.cont.mild, decreasing = TRUE)
factor(age)0.y1 factor(age)1.y1         arth.y1         stro.y1         diab.y1
    0.804099194     0.164283693     0.023317949     0.004633619     0.003665546

sort(rel.cont.sev, decreasing = TRUE)
factor(age)0.y2 factor(age)1.y2         diab.y2         stro.y2         arth.y2
     0.42687474      0.33665554      0.10556615      0.07191924      0.05898433
```

The background was the main contributor to the disability burden, representing 96.8% (0.80 + 0.16) and 76.4% (0.43 + 0.34) of the mild and severe disability prevalence, respectively. Arthritis (2.3%) was the main contributor to the mild disability prevalence while diabetes (10.6%) contributed most to the severe disability prevalence.

Model 7 - Multinomial model with background and diseases by age, with bootstrap CI

The matrix with the diseases (disease) defined for model 3 is used to fit model 7:

```
model7 <- MultAddHaz(dis.mult ~ factor(age) -1 + disease:factor(age),
                     data = disabData, weights = wgt, attrib = TRUE, attrib.var = age,
                     seed = 111, collapse.background = FALSE, attrib.disease = TRUE,
                     type.attrib = "both", bootstrap = TRUE, conf.level = 0.95,
                     nbootstrap = 1000, parallel = TRUE, type.parallel = "snow",
                     ncpus = 4)
```

The -1 was added to the model.formula to obtain the parameter estimates for the background for all age groups, including the reference category. Since the background should be estimated by age, collapse.background is set to FALSE. Additionally, attrib.disease is set to TRUE, as interactions between age and diseases were included in the model and the contribution of diseases should be estimated by age. The seed argument in MultAddHaz is used to obtain reproducible results for the starting values used in the constrained optimization, which are randomly generated, and for the bootstrap CI. Besides the summary function, the disabling impacts and the bootstrap CI can also be assessed with:

```
cbind(model7$coefficients, model7$ci)

                           Coefficients       CILow     CIHigh
factor(age)0.y1             0.117255379  0.10064630 0.13689281
factor(age)1.y1             0.277818866  0.20558349 0.37304023
factor(age)0:diseasediab.y1  0.005926107 -0.03440926 0.04770883
factor(age)1:diseasediab.y1 -0.027900849 -0.16964726 0.15898841
factor(age)0:diseasearth.y1  0.012814735 -0.02467113 0.05156243
factor(age)1:diseasearth.y1  0.110733103 -0.08447433 0.30975351
factor(age)0:diseasestro.y1  0.032163873 -0.03657685 0.14572706
factor(age)1:diseasestro.y1 -0.028504716 -0.22807053 0.22952796
factor(age)0.y2             0.109165655  0.09146724 0.13167070
factor(age)1.y2             0.672941316  0.53792263 0.83185332
factor(age)0:diseasediab.y2  0.121508443  0.06618176 0.17864913
factor(age)1:diseasediab.y2  0.282986455 -0.09742139 0.80712949
factor(age)0:diseasearth.y2  0.054335292  0.01095538 0.09957643
factor(age)1:diseasearth.y2  0.635463641  0.31671319 1.05424237
factor(age)0:diseasestro.y2  0.456594023  0.24959719 0.72863913
factor(age)1:diseasestro.y2  1.233578243  0.49988818 2.27216717
```

In the output, factor(age)0 and factor(age)1 refers to the age groups 60-79 years and $\geq$ 80 years, respectively. y1 refers to disability category 1, which here represents mild disability and y2 refers to disability category 2, representing severe disability.

Two coefficients (for diabetes and stroke in women aged $\geq$ 80 years with mild disability) were negative. This suggests a “protective" effect of these conditions. However, these results should be carefully interpreted as they were not statistically significant.

To identify the most disabling diseases for mild and severe disability by age group, the following code can be used:

```
mild.age0 <- model7$coefficients[seq(1, length(model7$coefficients), 2), ][1:4]
sev.age0 <- model7$coefficients[seq(1, length(model7$coefficients), 2), ][5:8]
mild.age1 <- model7$coefficients[seq(0, length(model7$coefficients), 2), ][1:4]
sev.age1 <- model7$coefficients[seq(0, length(model7$coefficients), 2), ][5:8]

mild.age0[order(mild.age0, decreasing = TRUE)]
            factor(age)0.y1 factor(age)0:diseasestro.y1
                0.117255379                 0.032163873
factor(age)0:diseasearth.y1 factor(age)0:diseasediab.y1
                0.012814735                 0.005926107

sev.age0[order(sev.age0, decreasing = TRUE)]
factor(age)0:diseasestro.y2 factor(age)0:diseasediab.y2
                 0.45659402                  0.12150844
            factor(age)0.y2 factor(age)0:diseasearth.y2
                 0.10916565                  0.05433529

mild.age1[order(mild.age1, decreasing = TRUE)]
            factor(age)1.y1 factor(age)1:diseasearth.y1
                 0.27781887                  0.11073310
factor(age)1:diseasediab.y1 factor(age)1:diseasestro.y1
                -0.02790085                 -0.02850472

sev.age1[order(sev.age1, decreasing = TRUE)]
factor(age)1:diseasestro.y2             factor(age)1.y2
                  1.2335782                   0.6729413
```

```
factor(age)1:diseasearth.y2 factor(age)1:diseasediab.y2
                  0.6354636                   0.2829865
```

Stroke was the most disabling disease in women with severe disability in both age groups and in women aged 60-79 years with mild disability while arthritis was the most disabling disease in women aged $\geq$ 80 years with mild disability.

The main contributors to the disability burden, based on the absolute contribution can be assessed with:

```
cont.mild.age0 <- model7$contribution$att.abs[1:5, ]
cont.sev.age0 <- model7$contribution$att.abs[6:10, ]
cont.mild.age1 <- model7$contribution$att.abs[11:15, ]
cont.sev.age1 <- model7$contribution$att.abs[16:20, ]

cont.mild.age0[order(cont.mild.age0[, 1], decreasing = TRUE), ]
                                 att.abs        CILow      CIHigh
disab.0.y1                  0.1146399721 0.1142911352 0.115027470
backgrnd.0.y1               0.1103973843 0.1103736506 0.110418787
factor(age)0:diseasearth.0.y1 0.0024598331 0.0022352202 0.002706063
factor(age)0:diseasediab.0.y1 0.0010238914 0.0009230407 0.001137036
factor(age)0:diseasestro.0.y1 0.0007588633 0.0005256459 0.001035586

cont.sev.age0[order(cont.sev.age0[, 1], decreasing = TRUE), ]
                                att.abs       CILow     CIHigh
disab.0.y2                  0.137612800 0.133986991 0.14160126
backgrnd.0.y2               0.095139459 0.094884399 0.09537052
factor(age)0:diseasediab.0.y2 0.020450103 0.018538859 0.02259582
factor(age)0:diseasestro.0.y2 0.012179369 0.009234356 0.01571648
factor(age)0:diseasearth.0.y2 0.009843869 0.008984070 0.01077563

cont.mild.age1[order(cont.mild.age1[, 1], decreasing = TRUE), ]
                                  att.abs        CILow       CIHigh
disab.1.y1                   0.2532173709  0.248985442  0.257917633
backgrnd.1.y1                0.2408954310  0.240163632  0.241550122
factor(age)1:diseasearth.1.y1  0.0170621273  0.012596001  0.022104725
factor(age)1:diseasestro.1.y1 -0.0006391061 -0.001067613 -0.000299713
factor(age)1:diseasediab.1.y1 -0.0041010813 -0.005510841 -0.002757878

cont.sev.age1[order(cont.sev.age1[, 1], decreasing = TRUE), ]
                               att.abs      CILow     CIHigh
disab.1.y2                  0.52797382 0.51474229 0.54101616
backgrnd.1.y2               0.38747404 0.38073529 0.39414511
factor(age)1:diseasearth.1.y2 0.07581147 0.06237721 0.09017923
factor(age)1:diseasestro.1.y2 0.03293044 0.02088364 0.04734430
factor(age)1:diseasediab.1.y2 0.03175788 0.02394332 0.04002170
```

The severe disability prevalence (60-79 years = 13.8%; $\geq$80 years = 52.8%) was larger than the mild disability prevalence (60-79 years = 11.5%; $\geq$80 years = 25.3%) in both age groups. Arthritis was the main contributor to the mild disability prevalence in both age groups and to the severe disability prevalence in women aged $\geq$80 years, while diabetes was the main contributor to the severe disability prevalence in women aged 60-79 years.

Discussion

In this paper we introduced the R package **addhaz** developed to fit the binomial and multinomial additive hazard models to estimate the contribution of diseases to the disability prevalence using cross-sectional data.

The R package **addhaz** was developed based on the R functions developed by Nusselder and Looman (Nusselder and Looman, 2010) for binomial disability outcomes and for non-R users. The main advantages of **addhaz** compared to the original R functions are: (i) option to use the attribution method for multinomial responses using the function `MultAddHaz`; and (ii) option to do parallel computing for the calculation of the bootstrap percentile confidence intervals. However, the possibility to use reduced rank regression (Yee, 2014) to estimate the cause-specific disability rates by age group, for example, which is available in the original R functions (Nusselder and Looman, 2010), is not available in **addhaz**. Nonetheless, in **addhaz** these interactions can be estimated by including full interaction terms between chronic conditions and age groups.

Although the parameter estimates of the binomial additive hazard model can also be obtained with the R package **logbin**, the contribution of diseases to the disability prevalence is not provided, since **logbin** was not developed with focus on the attribution method. Therefore, for analysis aimed at the attribution of disability to diseases, we recommend the use of **addhaz**. For multinomial outcomes, no other software is available to fit the multinomial additive hazard model and to calculate the contributions, to our knowledge.

One could argue that instead of using the multinomial model, two binomial models could be fitted: (i) no x mild disability; and (ii) no x severe disability. Although this is indeed possible, with a minor loss of precision (larger standard errors) for the parameter estimates when the reference category ("no disability", in our example) is the most frequent category in the population (which is the case for the subset of the BNHS used here, as 67% were not disabled, 13% reported mild disability, and 20% were severely disabled) (Agresti, 2002), the prevalence of the various disability categories do not sum to 100%, as can be observed in a previous study that assessed the difference in the mild and severe disability burden using two binomial models (Yokota et al., 2015a).

Different models with different options were presented using **addhaz**, showing a wide possibility of application to the users. One example is the investigation of the role of multimorbidity on the disability prevalence, which was assessed by the inclusion of two-way interactions between diseases in the model, as presented in model 2. Even though the examples only included the combination of two diseases, higher order interactions can also be included in the models, with the sample size being the limiting factor. In addition, since the prevalence of chronic conditions tends to increase over age, the model parameterization to include interactions between diseases and age groups was also shown for the binary (models 3 and 4) and multinomial disability outcomes (model 7). Although age group was used as the stratification variable to estimate the disabling impacts of the diseases and background, other variables can be used, such as education attainment and sex.

Furthermore, we illustrated how the likelihood ratio test (LRT) can be performed for model selection using the function `LRTest`. The LRT can be also performed for model selection with the multinomial additive hazard model.

The attribution method has some limitations that should be considered. The main limitation of the method is the causality assumption. Although a causal relationship between diseases and disability is plausible and has been proposed in several disability models (Verbrugge and Jette, 1994), it cannot be assessed with cross-sectional data. As a consequence, disability is incorrectly attributed to diseases when disability occurred before the diseases. Although the parallel option reduces significantly computation time for calculating bootstrap confidence intervals, fitting the multinomial model to high dimensional data can still be time-consuming. For example, the computational time to fit model 7, in a Windows computer Intel(R) Core (TM) i7-4600 CPU with 2.1GHz and 2.7GHz, 8GB (RAM), using the parallel option with 4 cores, was 23.04 hours.

Summary

In conclusion, **addhaz** is a publicly available tool to assess the contribution of chronic conditions to the disability prevalence, using cross-sectional data. The results produced by the tool can be used by policymakers to reduce the disability burden. Future areas of interest to improve the package include the extension of the multinomial model to ordinal responses and alternatives to reduce computation time for high dimensional data.

Bibliography

A. Agresti. *Categorical Data Analysis*. John Wiley & Sons, 2002. ISBN 0-471-36093-7. [p92]

J. R. Beard, A. Officer, I. A. de Carvalho, R. Sadana, A. M. Pot, J.-P. Michel, P. Lloyd-Sherlock, J. E. Epping-Jordan, G. G. Peeters, W. R. Mahanani, et al. The world report on ageing and health: A policy framework for healthy ageing. *The Lancet*, 387(10033):2145–2154, 2016. URL `http://dx.doi.org/10.1016/S0140-6736(15)00516-4`. [p75]

L. Blizzard and D. Hosmer. The log multinomial regression model for nominal outcomes with more than two attributes. *Biometrical Journal*, pages 889–902, 2007. URL `https://doi.org/10.1002/bimj.200610377`. [p79]

A. Canty and B. Ripley. **boot**: *Bootstrap R (S-PLUS) Functions*, 2016. URL `https://CRAN.R-project.org/package=boot`. R package version 1.3-18. [p79]

H. Chen, H. Wang, E. M. Crimmins, G. Chen, C. Huang, and X. Zheng. The contributions of diseases to disability burden among the elderly population in china. *Journal of Aging and Health*, pages 261–82, 2013. URL `https://doi.org/10.1177/0898264313514442`. [p75]

C. Chiang. Competing risks in mortality analysis. *Annual Review of Public Health*, pages 281–307, 1991. URL `https://doi.org/10.1146/annurev.pu.12.050191.001433`. [p77]

A. Davison and D. Hinkley. *Bootstrap Methods and Their Applications*. Cambridge University Press, 1997. ISBN 0-521-57391-2. [p79]

M. W. Donoghoe. **logbin**: *Relative Risk Regression Using the Log-Binomial Model*, 2016. URL `https://CRAN.R-project.org/package=logbin`. R package version 2.0.1. [p79]

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC Press, 1st edition, 1994. ISBN 0412042312. [p76, 78, 79]

B. Klijs, W. J. Nusselder, C. W. Looman, and J. P. Mackenbach. Contribution of chronic disease to the burden of disability. *PLoS One*, 6(9):e25325, 2011. URL `https://doi.org/10.1371/journal.pone.0025325`. [p75]

K. Lange. *Numerical Analysis for Statisticians*. Springer-Verlag, 2nd edition, 2010. ISBN 978-1-4419-5944-7. [p78]

K. Manton and E. Stallard. *Recent Trends in Mortality Analysis*. Academic Press, 1984. ISBN 0124700209. [p77]

I. Marschner and A. Gillett. Relative risk regression: Reliable and flexible methods for log-binomial models. *Biostatistics*, pages 179–192, 2012. URL `https://doi.org/10.1093/biostatistics/kxr030`. [p79]

W. Nusselder and C. Looman. WP7: Decomposition tools - Technical report on attribution tool. Technical report, 2010. URL `http://www.eurohex.eu/pdf/Reports{_}2010/2010TR7.2{_}TRonattributiontool.pdf`. [p75, 76, 77, 79, 82, 91, 92]

W. J. Nusselder and C. W. Looman. Decomposition of differences in health expectancy by cause. *Demography*, 41(2):315–334, 2004. URL `https://doi.org/10.1353/dem.2004.00`. [p75, 76, 77, 79]

W. J. Nusselder, C. W. Looman, J. P. Mackenbach, M. Huisman, H. Van Oyen, P. Deboosere, S. Gadeyne, and A. E. Kunst. The contribution of specific diseases to educational disparities in disability-free life expectancy. *American Journal of Public Health*, 95(11):2035–2041, 2005. URL `http://doi.org/10.2105/AJPH.2004.054700`. [p75]

SAS Institute Inc. The sas system, version 9.2, 2008. URL `http://www.sas.com/`. [p75]

R. Strobl, M. Müller, R. Emeny, A. Peters, and E. Grill. Distribution and determinants of functioning and disability in aged adults-results from the german kora-age study. *BMC Public Health*, 13(1):1, 2013. URL `http://doi.org/10.1186/1471-2458-13-137`. [p75]

C. Szwarcwald, D. Malta, C. Pereira, M. Vieira, W. Conde, P. Souza Júnior, G. Damacena, L. Azevedo, G. Silva, M. Theme Filha, C. Lopes, D. Romero, W. Almeida, and C. Monteiro. Pesquisa nacional de saúde no brasil: Concepção e metodologia de aplicação. *Ciência & Saúde Coletiva*, 19(2):333–342, 2014. URL `http://dx.doi.org/10.1590/1413-81232014192.14072012`. [p80]

L. Verbrugge and A. Jette. The disablement process. *Social Science & Medicine*, pages 1–14, 1994. URL `http://dx.doi.org/10.1016/0277-9536(94)90294-1`. [p92]

M. Yang, X. Ding, and B. Dong. The measurement of disability in the elderly: A systematic review of self-reported questionnaires. *Journal of the American Medical Directors Association*, 15(2):150–e1, 2014. URL `https://doi.org/10.1016/j.jamda.2013.10.004`. [p75]

T. Yee. Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics & Data Analysis*, pages 889–902, 2014. URL `https://doi.org/10.1016/j.csda.2013.01.012`. [p79, 91]

T. W. Yee. **VGAM**: *Vector Generalized Linear and Additive Models*, 2016. URL `https://CRAN.R-project.org/package=VGAM`. R package version 1.0-2. [p79]

R. Yokota, J. Van der Heyden, S. Demarest, J. Tafforeau, W. Nusselder, P. Deboosere, and H. Van Oyen. Contribution of chronic diseases to mild and severe disability burden in Belgium. *Archives of Public Health*, page 37, 2015a. URL `http://doi.org/10.1186/s13690-015-0083-y`. [p92]

R. T. C. Yokota, N. Berger, W. J. Nusselder, J.-M. Robine, J. Tafforeau, P. Deboosere, and H. Van Oyen. Contribution of chronic diseases to the disability burden in a population 15 years and older, Belgium, 1997-2008. *BMC Public Health*, 15(1):1, 2015b. URL `https://doi.org/10.1186/s12889-015-1574-z`. [p75]

R. T. C. Yokota, L. de Moura, S. S. C. de Araújo Andrade, N. N. B. de Sá, W. J. Nusselder, and H. Van Oyen. Contribution of chronic conditions to gender disparities in disability in the older population in Brazil, 2013. *International Journal of Public Health*, 61(9):1003–1012, 2016. URL `https://doi.org/10.1007/s00038-016-0843-7`. [p75, 80]

R. T. C. Yokota, H. Van Oyen, C. W. N. Looman, W. J. Nusselder, M. Otava, Y. W. Kifle, and G. Molenberghs. Multinomial additive hazard model to assess the disability burden using cross-sectional data. *Biometrical J.*, 2017. URL `https://doi.org/10.1002/bimj.201600157`. [p75]

Renata Tiene de Carvalho Yokota
Epidemiology and Public Health, Sciensano
Interface Demography, Vrije Universiteit Brussel
Belgium
`Renata.Yokota@sciensano.be`

Caspar W. N. Looman
Department of Public Health, Erasmus Medical Center
Netherlands
`c.looman@erasmusmc.nl`

Wilma Johanna Nusselder
Department of Public Health, Erasmus Medical Center
Netherlands
`w.nusselder@erasmusmc.nl`

Herman Van Oyen
Epidemiology and Public Health, Sciensano
Department of Public Health, Universiteit Gent
Belgium
`Herman.VanOyen@sciensano.be`

Geert Molenberghs
Interuniversity Institute for Biostatistics and statistical Bioinformatics (I-BioStat), Universiteit Hasselt and Katholieke Universiteit Leuven
Belgium
`geert.molenberghs@uhasselt.be`

Snowboot: Bootstrap Methods for Network Inference

by Yuzhou Chen, Yulia R. Gel, Vyacheslav Lyubchich, and Kusha Nezafati

Abstract Complex networks are used to describe a broad range of disparate social systems and natural phenomena, from power grids to customer segmentation to human brain connectome. Challenges of parametric model specification and validation inspire a search for more data-driven and flexible nonparametric approaches for inference of complex networks. In this paper we discuss methodology and R implementation of two bootstrap procedures on random networks, that is, patchwork bootstrap of Thompson et al. (2016) and Gel et al. (2017) and vertex bootstrap of Snijders and Borgatti (1999). To our knowledge, the new R package **snowboot** is the first implementation of the vertex and patchwork bootstrap inference on networks in R. Our new package is accompanied with a detailed user's manual, and is compatible with the popular R package on network studies **igraph**. We evaluate the patchwork bootstrap and vertex bootstrap with extensive simulation studies and illustrate their utility in an application to analysis of real world networks.

Introduction

Traditionally, the structural network analysis, that is, the analysis of data in the form of graphs, has been primarily approached as a descriptive task, in contrast to an inferential task, while the employed analytic tools have rooted mainly within research areas outside of "mainstream" statistical methodology. In the recent years, however, there has been a flare of interest in developing new statistical methods for inference on complex networks (see overviews by Goldenberg et al., 2010; Kolaczyk and Csárdi, 2014; Brugere et al., 2017, and references therein). Despite the explosive growth of statistical methods for network analysis, the developed methodology for inference on graph-structured data remains predominantly parametric and model-based. At the same time, nonparametric bootstrap and resampling appears as an appealing and flexible data-driven alternative for network inference, especially, if only a single realization of a large complex network exists. That is, we can follow the same route as the classical bootstrap of Efron (1979) which was proposed four decades ago as an alternative to conventional parametric methods for independent and identically distributed data, and later was extended to various weakly dependent space-time processes (Hall, 1992; Shao and Tu, 1995; Chernick, 2008).

We attribute the first pioneering attempt to develop nonparametric bootstrap on networks to Snijders and Borgatti (1999). The Snijders–Borgatti procedure, called vertex bootstrap, allows evaluating estimation uncertainty for network density and testing one- and two-sample hypotheses for densities, under the assumption that all the network information is available upfront. Vertex bootstrap is widely used in social studies and is implemented in such highly popular software for social network analysis as UCINET (Borgatti et al., 2002). However, vertex bootstrap remains largely unknown in statistics and there still exists no implementation of this procedure in R. The next attempt to draw bootstrap inference on complex networks, with a focus on uncertainty quantification in network degree estimation, has been suggested by Thompson et al. (2016) and Gel et al. (2017). The idea is to borrow the "blocking" argument, developed for bootstrapping of time series and re-tiling of spatial data, and adapt it to random networks. The resulting procedure, called patchwork bootstrap, starts from sampling a set of multiple ego networks of varying orders and forming a patch (i.e., a network block analogue), and then proceeds to resampling the data within patches. Patchwork bootstrap allows to quantify estimation uncertainties for network degree distribution and its functions, and to construct reliable and sharp confidence intervals in a fully data-driven way. Furthermore, patchwork bootstrap is applicable to ultra-sparse and only partially observable networks.

The new R package **snowboot** (Ramirez-Ramirez et al., 2018) is the first to offer vertex and patchwork bootstrap in R. Moreover, to our knowledge, there currently exist only two more R packages implementing bootstrap analysis of networks, **bootnet** and **sna**. The package **bootnet** assesses uncertainty in estimation of edge-weights and centrality indices but does not account for network dependence structure among vertices. The package **sna** implements parametric bootstrapping of edges to generate new random graphs that can be further used for hypothesis testing of matching between the observed network and randomized baseline network as well as canonical correlation coefficients. This paper aims to further fill the gap and showcase utility of nonparametric resampling for network inference, with a particular focus on vertex and patchwork bootstraps. **snowboot** imports the R packages **graphics**, **igraph** (Csárdi and others, 2018), **parallel**, **Rcpp** (Eddelbuettel et al., 2017), **Rdpack** (Boshnakov, 2018), **stats**, and **VGAM** (Yee, 2018).

The paper is organized as follows. In the next section, we discuss methodology and implementation of vertex and patchwork bootstraps. In the simulation studies, we evaluate the implemented bootstrap methods in an application to synthetic data. Our case studies illustrate application of the bootstrap to analysis of airline networks, power grids, and the David Copperfield network. The paper is concluded by a discussion.

Bootstrap algorithms on networks

Patchwork bootstrap

While the first results on sampling on networks go back to the 1960s (see, e.g., Goodman, 1961; Frank, 1968; Granovetter, 1976) and while nowadays there exist numerous graph sampling procedures (see overviews by Scott and Carrington, 2011; Ahmed et al., 2014; Kolaczyk, 2009; Simpson et al., 2015; Zhang et al., 2015, and references therein), still surprisingly little is known on how to reliably and efficiently quantify sampling uncertainties, without imposing typically unverifiable model specification constraints. In this section, we discuss the new method of patchwork sampling and bootstrap (based on algorithms of Thompson et al., 2016 and Gel et al., 2017) that enables us to quantify sampling estimation uncertainties for network degree distribution and its functions, while using only a small proportion of network information.

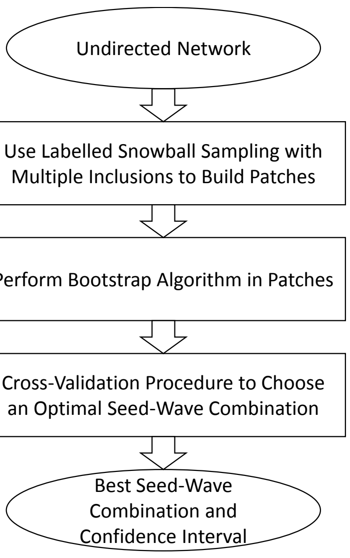

Undirected Network
Use Labelled Snowball Sampling with Multiple Inclusions to Build Patches
Perform Bootstrap Algorithm in Patches
Cross-Validation Procedure to Choose an Optimal Seed-Wave Combination
Best Seed-Wave Combination and Confidence Interval

Figure 1: The workflow of the patchwork bootstrap.

Assumptions

Consider an undirected random graph $G = (V, E)$ with a set of vertices, $V(G)$, and a set of edges, $E(G)$. The order and size of G are defined as the number of vertices and edges in G, i.e., $|V(G)|$ and $|E(G)|$, respectively. We assume that G has no self-loops, i.e., $u \neq v$ for any edge $e_{uv} \in E$. The degree of a vertex v is the number of edges incident to v. We denote the probability that a randomly selected vertex has a degree k by $f(k)$, the degree distribution of G by $F = \{f(k), k \geq 0\}$, and the mean degree of G by $\mu(G)$.

Let graph G represent some hypothetical "true" random graph of interest that is never fully observed, and its degree distribution F with finite mean and its order are unknown. Instead, we observe a random graph G_n of order n with a degree distribution $F_n = \{f_n(k), k \geq 0\}$. Let $N_k^{(n)}$ be the number of vertices with a degree k in G_n. Observed graph G_n can be viewed as a realization of G in a sense that as $n \to \infty$, $N_k^{(n)}/n \to f(k)$ in probability (empirical distribution F_n converges in probability to F) and joint degree distribution of G_n approaches that of G (see Britton et al., 2006; van der Hofstad, 2017 and references therein).

Furthermore, we assume that G is *involution invariant* (Lovász, 2012; Orbanz and Roy, 2015; Crane, 2018). Note that involution invariance is linked to unimodularity (Aldous and Lyons, 2007). That is, let us select a vertex v, $v \in V$ and perform a single step random walk from v to one of its neighbors u, $u \in V$; then the distribution of the neighborhood of v will be the same as the distribution of the neighborhood of u. I.e., from the vantage point of any randomly selected vertex, the rest of the connected network is probabilistically the same. The property of involution invariance can be viewed as a network analogue of stationarity of stochastic processes (Orbanz and Roy, 2015; Crane, 2018). Indeed, as per Aldous and Lyons (2007), unimodularity, or involution invariance, relates to "statistical homogeneity" or "spatial stationarity" of a network. In turn, stationarity is typically an essential condition for consistency of block bootstrap for space and time dependent data, thus, again linking our bootstrap procedure with the "blocking" argument. However, similarly to strong stationarity in time series analysis, involution invariance is not a formally *verifiable* condition. In practical terms of network analysis, the proposed patchwork bootstrap is applicable to networks that are believed to be "homogenous," that is, their distributional properties are the same across the whole network, which includes, for example, but not limited to exchangeable graphs (Crane, 2018). In turn, the patchwork bootstrap will not be, for example, applicable to networks with community structures. Note that in contrast to exchangeable networks which are dense, involution invariance (and the proposed patchwork bootstrap) is also applicable to sparse networks (Orbanz, 2017).

To the best of our knowledge, there currently exists no competing bootstrap procedure for quantifying uncertainty in estimators of network degree distribution. The subsampling procedure of Bhattacharyya and Bickel (2015) focuses on assessment of uncertainty in motif, or subgraph counts in dense exchangeable networks and is not feasible for inference on degree distribution. The method of Ali et al. (2016) also targets inference on counts of small sub-graphs and is based on the notion of dependency graphs under the network exchangeability framework. The vertex bootstrap approach of Snijders and Borgatti (1999), implemented in our R package **snowboot**, does not impose density limitations but assumes availability of the whole network upfront and is applicable to only small networks. In turn, Lusseau et al. (2008) and Epskamp et al. (2018) propose bootstrap for edge-weights and centrality indices without accounting for network dependence structure among vertices.

The workflow of the proposed patchwork bootstrap consists of the three key steps, that is, Labeled Snowball Sampling with Multiple Inclusions, Resampling, and Cross-Validation (see Figure 1).

Labeled snowball sampling with multiple inclusions

The central element of our technique for network sampling and inference is a *patch*, which is a structured sample of vertices and edges joining them. Patch sampling is performed in the following three steps:

1. Sample randomly without replacement several vertices from a network (Figure 2(a)).
2. Construct a Labeled Snowball with Multiple Inclusions (LSMI) independently around each vertex (Figure 2(b)):
   (a) Label each of the sampled vertices as a *seed*.
   (b) Select adjacent vertices by following the edges emanating from the seed and label them as *the first wave of non-seeds*.
   (c) Select and label neighborhoods of second and higher orders by following the edges emanating from the first wave of non-seeds. A vertex with degree $k > 1$ is included into the sample as many times as it is selected by following the previously unused edges (multiple inclusions are allowed).
3. Join all LSMIs into one patch (Figure 2(c)).

Detailed steps of the patch formation are given in Figure 3, which is a modified version of respective algorithms by Thompson et al. (2016) and Gel et al. (2017). The advantage of the algorithm in Figure 3 is that we explore patches with a smaller number of seeds by taking a subset from the seeds we have sampled, rather than by sampling new seeds from a network. This further improves computational efficiency of the algorithm and minimizes the amount of information obtained from a network.

To demonstrate the algorithm, we use one of the artificial networks stored in the R package **snowboot**:

```
> library(snowboot)
> net <- artificial_networks[[1]]
```

Function `lsmi` allows us to create a patch with seeds being randomly sampled or pre-specified. For example, a patch with two random seeds and one wave of neighbors around them:

```
> set.seed(1)
> lsmi(net, n.seed = 2, n.wave = 1)
  [[1]]
```

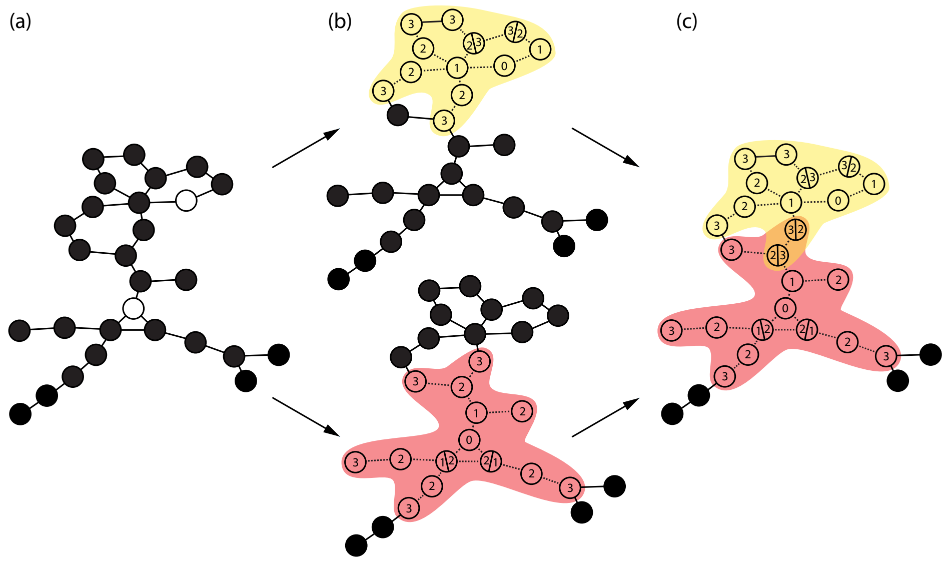


Figure 2: (a) Two seeds sampled from a network; (b) Two LSMIs grown independently, up to the third wave; (c) A patch sample of two seeds and three waves is obtained by joining the two LSMIs together. The waves are denoted with numbers from 0 (seed) to 3 (third wave) in each LSMI.

```
  [[1]][[1]]
  [1] 532
  [[1]][[2]]
  [1] 524 763

  [[2]]
  [[2]][[1]]
  [1] 744
  [[2]][[2]]
  [1] 145 858
```

The output is structured as a list of length equal to the number of seeds sampled. Each element of this list is a list itself, where first element contains the seed ID; second element contains IDs of vertices in the first wave, and so on. This structure allows us to keep track of neighborhoods around each seed separately, including the labels – waves in which the vertices appear. The above output shows that two seeds were sampled, with IDs 532 and 744. The first-order neighbors of vertex 532 are 524 and 763; of vertex 744 – vertices 145 and 858.

Alternatively, we can specify particular seeds and select the neighborhoods around them:

```
> lsmi(net, seeds = c(532, 744), n.wave = 1)
```

This option can be used to study specific vertices in a network (e.g., select the neighborhood around Erdös in the Erdös collaboration network of mathematical scientists; Thompson et al., 2016).

The next function, `lsmi_union`, records information about patches obtained by subsetting the originally sampled seeds (Figure 3):

```
> patches <- lsmi_union(net, n.seeds = c(2, 5, 10), n.wave = 2)
> ls(patches)
  [1] "lsmi_big"       "sequence_seeds"
> patches$sequence_seeds
  [[1]]
  [1]  124  352  403  411 1146 1255 1319 1795 1816 1886

  [[2]]
  [1]  411 1146 1319 1816 1886

  [[3]]
  [1] 1146 1886
```

The output is a list of two elements: a patch with the biggest number of seeds and waves from those specified (in the example above, `patches$lsmi_big` is a patch with 10 seeds and 2 waves around each

```
Input  : Graph $G_n$, where $n$ is number of vertices (network order); numbers of seeds
         to sample, $\{n.seeds\}_{i=1}^{b}$; number of waves to select, $d$.
Output : $patch(\max(n.seeds), d) = \{LSMI(d)\}_{i=1}^{\max(n.seeds)}$, i.e., one patch sample for
         the seed-wave combination with the largest number of seeds and $d$ waves;
         $sequence\_seeds$ – a list of length $b$ comprising subsamples of seeds.
1  lsmi_union($G_n$, $\{n.seeds\}_{i=1}^{b}$, $d$)
2      $\{n.seeds\}_{i=1}^{b}$ = sort $n.seeds$ in decreasing order
3      $\{sequence\_seeds_1\}_{i=1}^{n.seeds_1}$ = sample randomly without replacement $n.seeds_1$
        vertices from $G_n$
4      for $i = 1, \dots, n.seeds_1$ do
5          $\{LSMI(d)\}_i$ = lsmi($G_n$, $\{sequence\_seeds_1\}_i$, $d$)
6      end
7      for $j = 2, \dots, b$ do
8          $\{sequence\_seeds_j\}_{i=1}^{n.seeds_j}$ = sample randomly without replacement $n.seeds_j$
            vertices from $\{sequence\_seeds_{j-1}\}_{i=1}^{n.seeds_{j-1}}$
9      end
10     return
1  lsmi($G_n$, $seed$, $n.wave$)
2      $\{wave_0\} = seed$
3      for $i = 1, \dots, n.wave$ do
4          let $\{wave_i\}$ be all immediate neighbors of the vertices from the set $\{wave_{i-1}\}$
5          eliminate all edges that were used to locate $\{wave_i\}$
6      end
7      $LSMI(n.wave) = \{wave_0\} \cup \dots \cup \{wave_{n.wave}\}$
```

Figure 3: Obtaining a set of patches with b different number of seeds, and waves from 1 to d.

seed) and a list of seeds and their subsamples (in the example above, those are vectors of lengths 10, 5, and 2 with IDs of the vertices) for constructing smaller patches.

The current version of the LSMI algorithm (Figure 3) pays special attention to counting the edges to precisely record of the vertices' degrees. In each patch, estimates of the probabilities for a vertex to have a certain degree k ($k > 0$) are obtained with a modified Horvitz–Thompson estimator, whereas $\hat{f}(0)$, the probability of vertices with zero degree, is approximated by the proportion of seeds with zero degree, $\hat{p}_0$. These estimates can be employed to calculate functions of the network degree distribution, e.g., mean degree μ (Thompson et al., 2016; Gel et al., 2017):

$$\begin{aligned}\hat{f}(0) &= \hat{p}_0 = \frac{|\{d_s = 0\}|}{|\{d_s\}|},\\ \hat{f}(k) &= \frac{|\{d_s = k\}| + (1-\hat{p}_0)\hat{\mu}_s|\{d_{ns} = k\}|k^{-1}}{|\{d_s\}| + \hat{\mu}_s\sum_{k\geqslant 1}|\{d_{ns} = k\}|k^{-1}},\\ \hat{\mu}(G) &= \sum_{k\geqslant 0} k\hat{f}(k),\end{aligned} \tag{1}$$

where d_s are the degrees of the sampled seeds; d_{ns} are the degrees of non-seeds; $|\cdot|$ denotes cardinality of a set; $\hat{\mu}_s$ is the estimated mean degree based on $\{d_s\}$:

$$\hat{\mu}_s = \sum_{k\geqslant 0} k\frac{|\{d_s = k\}|}{|\{d_s\}|}.$$

Since the probability of non-seed vertices to be included into an LSMI is proportional to their degree, the estimates $\hat{f}(k)$ in (1) use information from non-seeds downweighted by k^{-1}.

Using the lsmi_dd function, calculate $\hat{f}(k)$ from the patch we obtained earlier:

```
> empdd <- lsmi_dd(patches$lsmi_big, net)
> empdd$mu
  [1] 2.40574
> empdd$fk
            0           1           2           3           4           5
  0.000000000 0.369724254 0.248171075 0.207653348 0.082301632 0.040517727
            6           7           8           9
  0.004220597 0.025323579 0.016460326 0.005627462
```

The output is an object of class 'snowboot' containing the estimates of mean degree and degree distribution. The length of the latter vector depends on what was the maximal degree ($\max(k)$) of the vertices included into the current patch. In the example above, $\max(k) = 9$.

Resampling procedure

To quantify uncertainty associated with the sample estimates (1) of network statistics, we apply a weighted bootstrap procedure. The probability of each non-seed vertex to appear in a bootstrap sample is assigned a weight of d_{ns}^{-1}, i.e., reciprocal of the vertex's degree. We obtain B bootstrap samples and compute a network statistic on each of them to approximate the distribution of the sample estimates. Each combination of the number of seeds and waves gives different estimates (1), hence, the bootstrap procedure is applied separately to patches of different seed-wave combinations. The bootstrap counterparts of the estimates (1) are as follows:

$$\begin{aligned} \hat{f}^*(0) &= \hat{p}_0^* = \frac{|\{d_s^* = 0\}|}{|\{d_s^*\}|}, \\ \hat{f}^*(k) &= \frac{|\{d_s^* = k\}| + (1-\hat{p}_0^*)|\{d_{ns}^* = k\}|}{|\{d_s^*\}| + |\{d_{ns}^*\}|}, \\ \hat{\mu}^*(G) &= \sum_{k\geqslant 0} k\hat{f}^*(k), \end{aligned} \tag{2}$$

where $\{d_s^*\}$ and $\{d_{ns}^*\}$ are the respective sets of bootstrapped seeds and non-seeds.

The empirical bootstrap degree distribution can be obtained using the boot_dd function from the **snowboot** package:

```
> B <- 50
> bootdd <- boot_dd(empdd, B)
```

A part of the output is a $(1 + \max(k)) \times B$ matrix of bootstrap estimates $\hat{f}^*(k)$, where $k = 0, 1, \ldots, \max(k)$, and a vector of length B with $\hat{\mu}^*$:

```
> dim(bootdd$fkb)
  [1] 10 50
> length(bootdd$mub)
  [1] 50
```

The bootstrap degree distributions are used to quantify estimation uncertainty. That is, let η be the parameter of interest (i.e., the population value, μ or $f(k)$), $\hat{\eta}_n^j$ and $\hat{\eta}_n^{j*}$ be the respective conventional and bootstrap estimators of η based on a graph G_n ($j = 1, \ldots, J$ are the different seed-wave combinations for constructing patches). Then the Efron's $100(1-\alpha)\%$ bootstrap confidence interval for η:

$$CI^j_{percentile} = \left(\hat{\eta}^{j*}_{n,[B\alpha/2]}, \hat{\eta}^{j*}_{n,[B(1-\alpha/2)]}\right), \tag{3}$$

where B is the number of bootstrap samples; $\hat{\eta}^{j*}_{n,[B\alpha/2]}$ and $\hat{\eta}^{j*}_{n,[B(1-\alpha/2)]}$ are the empirical quantiles from the bootstrap distribution.

Having all bootstrapped values available from the output of boot_dd, we can employ a variety of methods for calculating bootstrap intervals alternative to the percentile method (3). At this time, the function boot_ci can be switched to compute "basic" bootstrap intervals (see Equation 5.6 by Davison and Hinkley, 1997):

$$CI^j_{basic} = \left(2\hat{\eta}^j_n - \hat{\eta}^{j*}_{n,[B(1-\alpha/2)]}, 2\hat{\eta}^j_n - \hat{\eta}^{j*}_{n,[B\alpha/2]}\right). \tag{4}$$

In our synthetic network example, $\hat{f}(3) = 0.208$, and its 95% bootstrap confidence interval (3), based on the 50 bootstrap samples, is

```
> CIpercentile <- boot_ci(bootdd)
> CIpercentile$fk_ci[,"3"]
       2.5%     97.5%
  0.1286842 0.2631579
```

The outputs of the functions lsmi_dd, boot_dd, and boot_ci are estimates of the network degree distribution $f(k)$ and mean degree μ based on a single patch. They are recognized as objects of class 'snowboot' and can be plotted with the S3 method plot for this class (Figure 4).

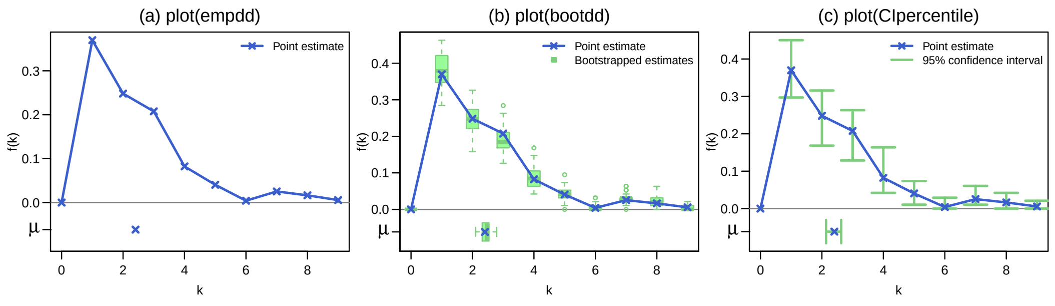


Figure 4: Results of plotting three different objects of class 'snowboot'.

Cross-validation with LSMI

Growing snowball samples to higher waves is used in sampling surveys when obtaining new seeds is prohibitively expensive, for example, in hard-to-reach and "hidden" subpopulations of HIV high risk individuals. Even if no more new seeds can be obtained, varying the number of seeds in a patch offers an additional flexibility, so that multiple seed-wave combinations can be evaluated and an optimal seed-wave combination is selected for a given observed network.

Given bootstrap confidence intervals from patches for each jth seed-wave combination ($j = 1, \ldots, J$), we use Algorithm 2 of Gel et al. (2017) to decide which patch provides confidence intervals of the best coverage for our statistic η. We approximate the ground truth using proxy samples. A proxy sample is obtained by resampling (with or without replacement) vertices already used in the LSMIs, so no new information is needed from the network. From multiple such samples, proxy statistics $\hat{\eta}^{proxy}$ are estimated, and then for each j a proportion of proxy statistics within the interval CI^j is calculated. This proportion aims to approximate the true coverage probability of the bootstrap confidence intervals, so an interval $CI^{j_{opt}}$ with the coverage closer to the nominal level $1 - \alpha$ can be selected.

Cross-validation is performed automatically by an upper-level function `lsmi_cv` in the **snowboot** package. This function obtains multiple patches for each of the specified seed-wave combinations and runs cross-validation to select the optimal bootstrap confidence interval based on proxy values for the mean (by default, 19 proxy samples are obtained, each comprising 30 vertices):

```
> cv <- lsmi_cv(net, n.seeds = c(10, 20, 30), n.wave = 5, B = 100)
> cv
  $`bci`
      2.5%    97.5%
  1.749167 2.952500

  $estimate
  [1] 2.387042

  $best_combination
  n.seed n.wave
      10      2

  $seeds
  [1]  218  323  851 1003 1039 1097 1410 1624 1723 1756
```

The output of `lsmi_cv` contains the optimal seed-wave combination (`best_combination`), the corresponding point estimate (`estimate`) and bootstrap confidence interval (`bci`), and, for information purposes, the IDs of the actual seeds that were used in the patch with the selected seed-wave combination (`seeds`). In this example, a fixed increment grid is used for the number of seeds and waves, however, a user may choose to do a stochastic search in the parameter space.

Why does patchwork bootstrap work and its areas of limitations

Bias vs. variance Many estimators of graph totals based solely on seeds are known to be unbiased (Frank, 1977). However, variance of such seed-based estimator might be high if the number of seeds is low. In turn, in many applications sampling more seeds might be prohibitively expensive, e.g., due to data privacy and cost restrictions (see overview by Illenberger and Flötteröd, 2012 and references therein). Adding information from non-seeds into the degree estimator increases bias but reduces variance. Hence, the choice of optimal number of seeds (egos) and waves of non-seeds in LSMI implies a classical bias vs. variance trade-off, and we address it using the cross-validation procedure described above.

Network properties Furthermore, as in many non-network settings, in order to ensure consistency of the bootstrap estimator $\hat{\eta}^*_n$ of the statistic of interest η_n, based on a degree distribution $F_n = \{f_n(k), k \geq 0\}$ of a random graph G_n, we typically need to ensure that F_n is a satisfactory approximation of F as $n \to \infty$. That is, we need to ensure that the empirical degree distribution F_n cannot be drastically different from the population degree distribution F and shall approach the population degree distribution F with increasing n, which in turn is a necessary condition for $\eta_n \to \eta$ as $n \to \infty$. In addition, F needs to satisfy some invariance properties. Those conditions give rise to the assumptions for patchwork bootstrap on page 96. Note that while nowadays there are no formal tests for assessing involution invariance, similarly as there exists no test to assess strong stationarity in time series, a class of involution invariant graphs includes, for example, such a large subclass as exchangeable graphs (Lovász, 2012; Orbanz and Roy, 2015). Asymptotic properties of patchwork bootstrap for $\hat{\eta}^*_n$ for involution invariant graphs are discussed in more details in Gel et al. (2017). In addition, Thompson et al. (2016) consider the two-phase conditional inference framework to derive asymptotic properties for patchwork bootstrap estimator of mean degree μ.

Sampling design Finally, we would like to emphasize that properties of bootstrap on networks largely depend not only on the underlying network topology but on the closely linked question on how sampling is performed. In this section we focus only on properties of a network degree distribution and argue that LSMI appears to be a suitable choice (see more discussion in Illenberger and Flötteröd, 2012; Gel et al., 2017). In turn, bootstrap and, generally, finite-sample inference for other network statistics will typically require adjustment of a sampling design (Kolaczyk, 2009).

Hence, the algorithms of LSMI sampling and estimation realized in the **snowboot** package target such network statistics as network degree distribution (probabilities of observing vertices with a specific degree) and its functions (e.g., mean degree and network density). In turn, the algorithm of patchwork bootstrap aims to quantify the uncertainty of those estimates.

Switching to other network statistics, such as clustering coefficient or motifs, requires implementation of different sampling and bootstrap schemes, for example, via combination of patchwork bootstrap with the algorithm of Ali et al. (2016) (see Orbanz, 2017, for discussion on sampling and subsampling designs). Hence, this extension constitutes a natural direction for future methodological research and R code implementation.

Vertex bootstrap

Vertex bootstrap (Figure 5), or the Snijders–Borgatti procedure, is the pioneering approach to non-parametric bootstrap inference on graphs. Nevertheless, despite its high popularity in social sciences, vertex bootstrap remains largely unknown in statistics. Vertex bootstrap employs an induced graph sampling for quantification of standard errors in network density estimation and allows hypothesis testing on density of two networks. The algorithm assumes availability of the entire network data upfront and requires resampling of the entire data set and, hence, is limited to relatively small networks due to computational costs. To the best of our knowledge, there exists no analysis of asymptotic properties and theoretical guarantees for the Snijders–Borgatti procedure.

We implement vertex bootstrap in the R package **snowboot** in a function `vertboot`, which resembles calculations under the option "Network → Compare densities" of the UCINET software (Borgatti et al., 2002).

The `vertboot` function not only generates similar results compared with UCINET, but also returns results with higher precision and faster run times. In **snowboot**, the algorithm is written in C++.

Another important improvement is that `vertboot` allows users to compare statistics of interest for multiple networks, whereas the number of different networks, T (Figure 5), in UCINET is limited to 1 or 2. Finally, the output of `vertboot` (Figure 5) is a bootstrapped network, which implies that users can analyze various network statistics besides the network density.

We demonstrate the `vertboot` function using prison network data. The data were collected from 67 prison inmates; each inmate could choose as few or as many "friends" as he desired. A direct factor analysis of these sociometric data was performed by MacRae (1960).

- Get the observed adjacency matrix for the prison network:

```
> a <- scan("http://vlado.fmf.uni-lj.si/pub/networks/data/ucinet/prison.dat",
+           skip = 4)
> A <- matrix(a, sqrt(length(a)), byrow = TRUE)
```

```
Input  : $T$ observed adjacency matrices $A_t$ ($t = 1, \dots, T$).
Output : $T$ bootstrapped adjacency matrices $A_t^*$.
for $t$ in $1, \dots, T$ do
    $list = \{1, \dots, nrow(A_t)\}$
    $list^* = sample\ with\ replacement\ elements\ of\ list$
    for $i$ in $1, \dots, length(list^*)$ do
        $ii = list^*[i]$
        for $j$ in $1, \dots, length(list^*)$ do
            $jj = list^*[j]$
            if $ii \neq jj$ then
                $A_t^*[i, j] = A_t[ii, jj]$
            end
            else
                $ii = random\ sample\ from\ list$
                $jj = random\ sample\ from\ list$
                while $ii = jj$ do
                    $jj = random\ sample\ from\ list$
                end
                $A_t^*[i, j] = A_t[ii, jj]$
                $A_t^*[j, i] = A_t[ii, jj]$
            end
        end
    end
end
```

Figure 5: Vertex bootstrap.

- Apply vertex bootstrap (Figure 5) B times to generate B bootstrapped adjacency matrices (networks):

```
> set.seed(1)
> B <- 500
> Astar <- vertboot(A, B)
```

- Get the densities of B bootstrapped networks:

```
> library(igraph)
> densities <- sapply(1:B, function(x)
+    graph.density(graph_from_adjacency_matrix(Astar[[x]])))
```

- Estimate density of the observed network and bootstrap standard error of the estimates:

```
> density_obs <- graph.density(graph_from_adjacency_matrix(A))
> densities_se <- sd(densities)
```

- Obtain a 95% bootstrap confidence interval for the population density:

```
> quantile(densities, c(0.025, 0.975))
      2.5%      97.5%
0.03301673 0.05178652
```

We can use the outputs of the vertex bootstrap (Figure 5) to build bootstrap confidence intervals for other statistics:

- Mean degree:

```
> mu_star <- sapply(1:B, function(x)
+    mean(degree(graph_from_adjacency_matrix(Astar[[x]]), mode = "in")))
> quantile(mu_star, c(0.025, 0.975))
    2.5%    97.5%
2.179104 3.417910
```

- Transitivity:

```
> net_transitivity <- sapply(1:B, function(x)
+    transitivity(graph_from_adjacency_matrix(Astar[[x]]), type = "undirected"))
> quantile(net_transitivity, c(0.025, 0.975))
      2.5%      97.5%
0.08890533 0.21852252
```

Simulation studies

Light- and heavy-tailed degree distributions

In this section, we demonstrate the performance of the patchwork bootstrap algorithm for inference on mean degree of simulated graphs. We consider random graphs of orders 1000, 5000, and 10000 that are constructed from two polylogarithmic distributions of the same mean (population mean degree $\mu(G) = 2.67$ for all simulated networks), but with different tail behavior. Particularly, polylogarithmic distribution with parameters $\delta = 0.001$, $\lambda = 2.13$ has a light tail, whereas polylogarithmic distribution with parameters $\delta = 0.987$, $\lambda = 5$ has a heavy tail (Figure 6).

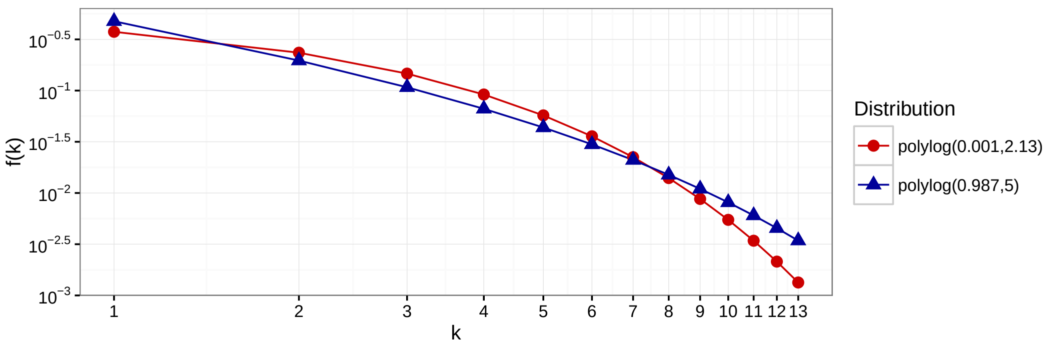


Figure 6: Polylogarithmic distributions with the same mean ($\mu = 2.67$), but different thickness of tails. The graphs are log-log graphs, where both axes use a logarithmic scale.

The results of our simulation study (Table 1) show a good performance of the patchwork bootstrap approach with automatic selection of patch sizes (i.e., seed-wave combinations) using the cross-validation procedure. Thus, the observed coverage probabilities of the confidence intervals are close to the nominal level 95% for both light- and heavy-tailed networks. At the same time, confidence intervals in the considered light-tailed networks are approximately 25% narrower. Time complexity of the patchwork procedure on any random graph is $\mathcal{O}([|d_s| \cdot \hat{\mu}_s]^{|d_{ns}|})$.

		Network order n		
Distribution	Mean degree $\mu(G)$	1000	5000	10000
polylog($\delta = 0.001$, $\lambda = 2.13$)	2.67	97.7% (0.794)	96.4% (0.812)	97.7% (0.805)
polylog($\delta = 0.987$, $\lambda = 5$)	2.67	98.6% (1.051)	98.7% (1.078)	97.7% (1.069)

Table 1: Observed coverage probabilities of 95% patchwork bootstrap confidence intervals for the mean degree (average interval width is in parentheses). These intervals come from the optimal seed-wave combination (one for each random graph) defined via a cross-validation over a grid of 20 combinations: waves from 1 to 5; and number of seeds 20, 30, 40, and 50. In cross-validation, proxy mean degrees are estimated 13 times from 100 vertices sampled without replacement from the patch data. The number of bootstrap samples, B, is 500. The experiment uses 1000 Monte Carlo simulations, carried out as parallel processes on a distributed computing cluster.

Vertex removal

In this section, we randomly remove vertices (and edges emanating from those vertices) from the simulated networks prior to applying the patchwork and vertex bootstrap methods for the inference on mean degree. This allows us to assess the methods' robustness and capabilities of providing reliable inference when part of the network information is missing. Our goal is to estimate when performance of the methods decreases significantly.

As Table 2 shows, when only 1% of the vertices are removed, both methods deliver confidence intervals with coverage close to the declared 95% level. When 2% of network vertices are removed, coverage probabilities of vertex bootstrap intervals decline to 81.8% and 89.8% for light- and heavy-tailed networks, respectively. The performance of vertex bootstrap further rapidly declines as more vertices are removed, and the decline is more severe for the light-tailed polylogarithmic distribution. Remarkably, patchwork bootstrap performs consistently well on both distributions (observed coverage probabilities are close to the nominal level), even when 5% of the vertices are removed (Table 2). Using the example of the light-tailed polylogarithmic distribution, we show that coverage of vertex bootstrap declines to zero when 10% or 15% of vertices are removed, while respective coverage of the patchwork bootstrap intervals is 93.9% and 85.1%, respectively (Figure 7).

	Mean		Number of vertices removed			
Distribution	degree $\mu(G)$	Method	1%	2%	3%	5%
polylog($\delta = 0.001, \lambda = 2.13$)	2.67	Patchwork	96.0% (0.807) [0.187]	97.8% (0.801) [0.194]	97.6% (0.806) [0.191]	97.5% (0.795) [0.196]
		Vertex bootstrap	96.3% (0.170) [0.090]	81.8% (0.170) [0.091]	54.7% (0.169) [0.090]	5.8% (0.169) [0.090]
polylog($\delta = 0.987, \lambda = 5$)	2.67	Patchwork	99.4% (1.072) [0.230]	99.0% (1.062) [0.239]	98.9% (1.062) [0.249]	99.4% (1.032) [0.237]
		Vertex bootstrap	96.3% (0.212) [0.113]	89.8% (0.211) [0.113]	74.0% (0.211) [0.113]	25.7% (0.209) [0.112]

Table 2: Observed coverage probabilities of 95% nonparametric bootstrap confidence intervals. Average interval width is in parentheses, standard errors are in square brackets. Network order $n = 5000$; number of bootstrap samples $B = 500$; 1000 Monte Carlo simulations. For the patchwork method, $J = 20$ seed-wave combinations: number of seeds 20, 30, 40, and 50; number of neighbors from 1 to 5.

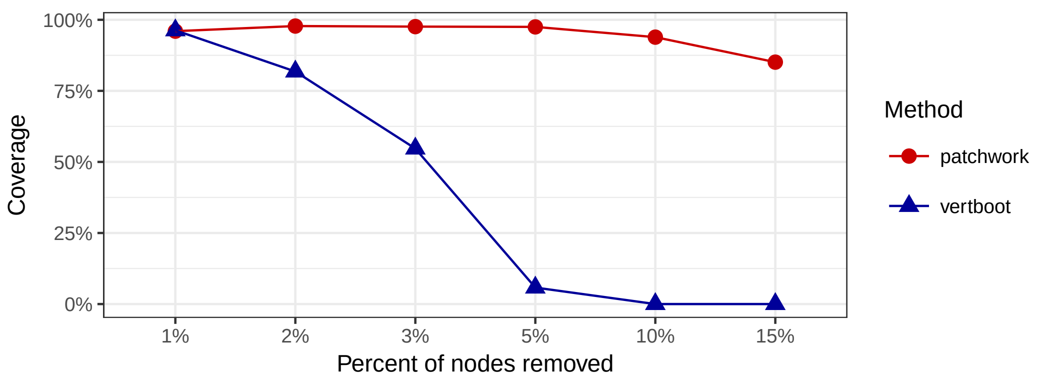


Figure 7: Observed coverage probabilities for 95% confidence intervals for network mean degree, delivered by the two bootstrap methods when different numbers of vertices are removed from the network. Network order $n = 5000$; degree distribution polylog($\delta = 0.001, \lambda = 2.13$); $B = 500$; 1000 Monte Carlo simulations.

Hence, the patchwork bootstrap approach is a competitive alternative when analyzing large complex networks, both in terms of computational speed and reliability of inference when a part of the network information is missing. Moreover, the patchwork method is both computationally efficient and information-greedy, i.e., only a small proportion of the network data is required, the procedure subsets the sampled seeds to consider patches of different sizes and re-uses information from the patches in the cross-validation procedure. In contrast, the vertex bootstrap employs information of the whole target network so that its time complexity is $\mathcal{O}(n^2)$. At the same time, for small networks with all the network information being available upfront, the vertex bootstrap is the preferred method as it provides noticeably sharper confidence intervals under the same level of calibration.

Case studies

In this section, we illustrate utility of the **snowboot** package for analysis of airline networks, power grids, and the David Copperfield network.

The David Copperfield network

We start from a smaller network, namely, the David Copperfield network collected by Newman (2006). It examines the lexicon of Charles Dickens's classic 19th century novel. The network vertices are common nouns and adjectives; undirected edges connect adjacent words (Figure 8).

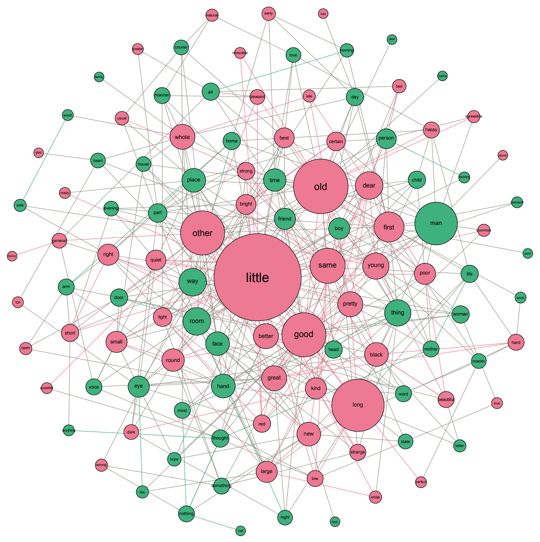


Figure 8: Lexical network: graph of nouns and adjectives found in the novel David Copperfield.

The number of vertices and edges in the David Copperfield network are 112 and 425, respectively. This is a relatively small network, so the vertex bootstrap algorithm is suitable for analysis. Some basic network statistics of the David Copperfield network are presented in Table 3.

Order	Density	Number of edges	Mean degree	Clustering coefficient
112	0.068	425	7.589	0.157

Table 3: Parameters of the David Copperfield network.

Perform the vertex bootstrap in the following steps:

- Load the network data and obtain an adjacency matrix:

```
> library(igraph)
> graph_david <- read.graph(
+    "http://networkdata.ics.uci.edu/data/adjnoun/adjnoun.gml", format = "gml")
> A <- as.matrix(as_adjacency_matrix(graph_david))
```

- Use vertex bootstrap (Figure 5) to obtain bootstrapped adjacency matrices:

```
> library(snowboot)
> B <- 500
> set.seed(1)
> Astar <- vertboot(A, B)
```

- Use these bootstrapped networks to calculate 95% bootstrap confidence intervals for the density and bootstrap standard error:

```
> boot_density <- sapply(1:B, function(x)
+    graph.density(graph_from_adjacency_matrix(Astar[[x]])))
> CIvertboot <- quantile(boot_density, c(0.025, 0.975))
> CIvertboot
        2.5%      97.5%
0.05473174 0.08579271
> bootstrap_standard_error <- sd(boot_density)
> bootstrap_standard_error
[1] 0.007681788
```

That is, we find that the resulting 95% confidence interval from the vertex bootstrap is (0.055, 0.086); the interval contains the observed density of 0.068. Now let us apply the patchwork bootstrap to the David Copperfield network.

- Use the patchwork bootstrap to calculate 95% bootstrap confidence intervals for the density:

```
> set.seed(5)
> igraph_david <- igraph_to_network(graph_david)
> CIpatchwork <- lsmi_cv(igraph_david, n.seeds = c(3:5), n.wave = 1, B = B)
> CIpatchwork$bci/(igraph_david$n - 1)
       2.5%      97.5%
  0.04305405 0.08957658
```

We find that the 95% confidence interval from the patchwork bootstrap also contains the observed density of 0.068. However, the patchwork bootstrap CI is wider than the vertex bootstrap CI (i.e., 0.047 for patchwork vs. 0.031 for vertex procedure). At the same time, the patchwork bootstrap used only about 31.4% of the available vertices.

Larger networks

Flight networks of the airline alliances

With constantly increasing costs of operation and rigid legal restrictions on ownership of national flag carriers, no single airline can provide a comprehensive global network, which is vital to its success. An urgent need for expansion propels airlines to effectively cooperate on multiple levels: from interlining (combining flights from different airlines in one travel itinerary) to joint frequent flyer programs, to sharing revenue, costs, and benefits (Pearce and Doernhoefer, 2011).

Nowadays, three major passenger airline alliances (Star Alliance, Oneworld, and SkyTeam) share more than 70% of the world market. There exists a constant, fierce competition in customer acquisition and retention among these three alliances, and one of its key factors is claimed to be a route map (number of served destinations and better connections). Traditional indicators, such as total passenger enplanements or number of aircraft movements, fail to capture the competitiveness of airline networks (Burghouwt and Redondi, 2013). However, for a fixed network structure, e.g., with hubs and spokes, dense air flight networks (with a high mean degree) are more convenient, since they further minimize the required number of connections. The International Air Transport Association (IATA) study of European Union countries showed that a 10% rise of the number of destinations served and/or the frequency of service can increase long-run gross domestic product by 1.1% (Smyth and Pearce, 2006).

We then evaluate densities of the three flight networks (Star Alliance, SkyTeam, and Oneworld) to assess which alliance offers the most convenient travel network to its passengers. Since flight connections are easier if a transfer occurs within one airport, we focus on airport networks. This approach treats all airports as separate vertices even some of them have the same service area (for example, Toronto Pearson International Airport and Billy Bishop Toronto City Airport).

To construct the networks, we obtained the crowdsourced air flights data from OpenFlights.org on October 23, 2013. At the time of analysis, Star Alliance, Oneworld, and SkyTeam included 28, 13, and 19 members, respectively. We used the airline codes to select all flights for each airline alliance, but removed self-loops and the cases with missing airport identifications (up to 0.4% of the records). To eliminate the repeated entries, including the codeshare flights within each alliance, we kept only unique links between airports. About 11.0% of all remaining vertices had different in-degree and out-degree, but we neglected the directions for simplicity. Thus, for each airline alliance, we obtained a network where the vertices stand for airports, and each unweighted undirected edge represents existing flight connection at least in one direction (Figure 9).

Table 4 reports the networks orders n and network densities $\hat{d}(G_n)$ estimated on all observed data along with the results of the patchwork bootstrap. The network order (i.e., number of served airports) is close to 1,000 for all three alliances. (Given a relatively high order of the airline networks, we do not consider the vertex bootstrap in this case study.) The optimal seed-wave combinations suggested by the cross-validation procedure differ among the networks: 20 seeds and 1 wave for SkyTeam, 30 seeds and 1 wave for Star Alliance, and 40 seeds and 1 wave for Oneworld. The width of the confidence interval for Star Alliance and Oneworld is only 0.00296 and 0.00321, respectively, compared with 0.00497 for SkyTeam. Since all three confidence intervals overlap with each other, we conclude that currently there exists no significant difference in flight connections offered by the three major airline alliances. Thus, the loyalty of frequent fliers and acquisition of new customers are likely to be attributed to other factors, such as customer service, loyalty program benefits, ticket prices, and availability of flights in particular regions (e.g., Figure 9 shows that Oneworld network provides almost no service in African airports).

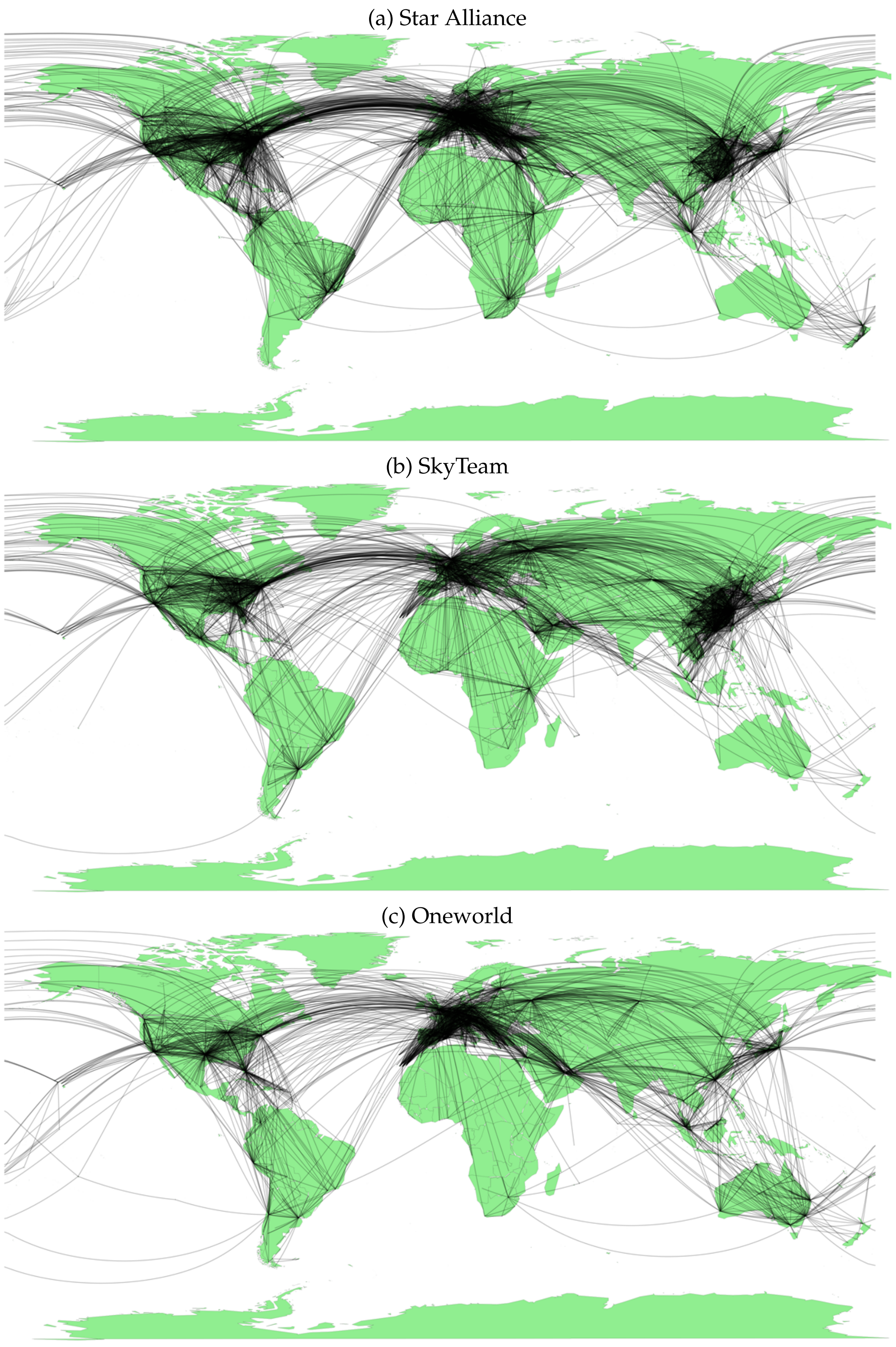


Figure 9: Airport networks of the airline alliances.

United States and Germany power grids

The power grid serves as the backbone of the critical infrastructure sector and is essential for today's society as an enabling infrastructure. A combination of three substations, i.e., generator, transmission, and distribution, connected by high voltage transmission lines, provide the United States and Germany with electrical power people so heavily rely on. As with many other large scale infrastructures, the

			Optimal combination		95% confidence bounds for the density $d(G)$	
Network	n	$\hat{d}(G_n)$	Seeds	Waves	Lower	Upper
Star Alliance	1289	0.00621	30	1	0.00492	0.00788
SkyTeam	1040	0.00736	20	1	0.00561	0.01058
Oneworld	914	0.00655	40	1	0.00533	0.00854

Table 4: The 95% bootstrap confidence intervals for the density of airline alliance networks, replicating connections of the airports (vertices) with the flights of member airlines (edges). Considered 12 seed-wave combinations: waves from 1 to 3, seeds 20, 30, 40, and 50. Number of bootstrap resamples is 500 per each combination. Cross-validation is based on a random selection of 100 vertices 10 times.

power grid serves users who may notice its presence and realize its importance only when the system fails in some way. One of the main issues with the system is that failures or disruptive events like hurricanes, earthquakes, and attacks, can cause cascading failures in a power grid. As the power grids increase in size and complexity, it is of a paramount importance to study their vulnerability.

To better understand the effects of power system failures, the power grids can be analyzed from the perspective of random networks. In this case study, we consider two power grids that are represented as undirected networks, that is, the power grids of the western states of the United States (Watts and Strogatz, 1998) and Germany (Matke et al., 2016). Each edge represents a power supply line and vertex is a generator, a transformer, or a substation (Figure 10).

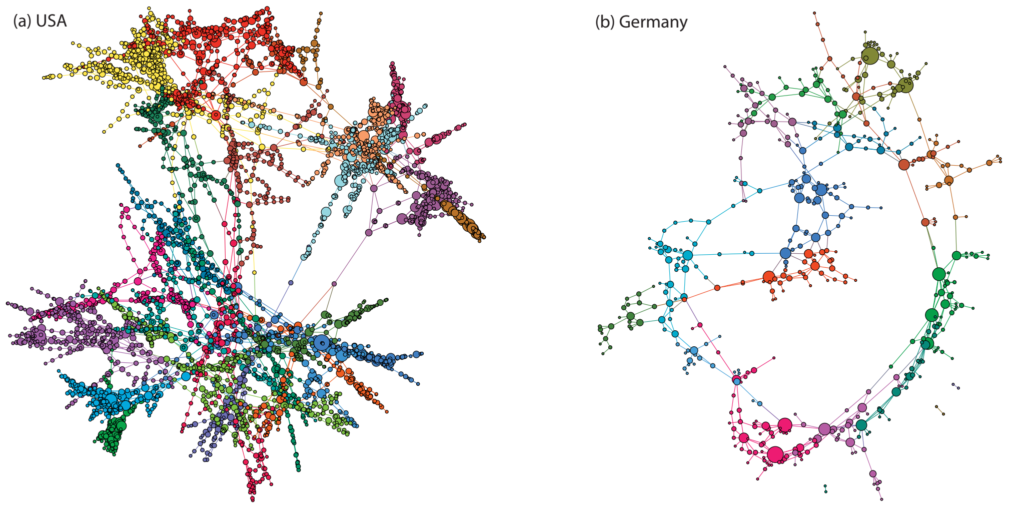


Figure 10: Power grids of the western states of the United States and Germany. The colors show network modules; connections between the vertices within modules are denser than between vertices from different modules.

Centrality statistics are one of the most widely explored attributes of a power grid network. Some studies focus on the relationship between various centrality statistics and resilience of the power grid networks (Pagani and Aiello, 2013). Another potential indicator of power system robustness and resilience is network density (Cuadra et al., 2015). In addition, Sóle et al. (2008) and Rosas-Casals and Corominas-Murtra (2009) propose to use a characteristic parameter γ, based on fitting an exponential distribution to an empirical cumulative distribution of each grid as a classifier of grid fragility. That is, the network is robust if $\gamma < 1.5$ and is fragile otherwise. In this study, we would like to examine the difference in fragility properties of the power grids in Germany and the western states of the United States, in terms of the γ parameter and the proportion of distribution stations in high-voltage networks. Given a relatively high order of the networks, the vertex bootstrap is not feasible, and we apply the patchwork bootstrap to compare the two power grids (see Table 5). We find that both power grids deliver the characteristic parameter $\hat{\gamma}$ of higher than 1.5, that is, 2.09 and 2.62 for the US and Germany, respectively, and hence both grids shall be classified as fragile. However, the respective 90% patchwork bootstrap confidence intervals do not overlap, and we can conclude that the US power grid in the western states tends to be less fragile than the German power grid. Remarkably, the 90% confidence intervals for densities of the two networks also do not overlap, hence, indicating that there likely exists a significant difference in connectivity of the two power grid networks.

While it is premature to conclude that the Western US power grid system is more robust than the German power system due to its higher sparsity, especially given the lack of a uniformly accepted notion of power system fragility and robustness (Pagani and Aiello, 2013; Cuadra et al., 2015; Dey et al., 2017; Islambekov et al., 2018), it is reasonable to conclude that the two systems exhibit significant differences in their structure. In turn, the bootstrap methodology provides a route how power grids and their network properties can be systematically evaluated and compared in a framework of statistical hypothesis testing.

				90% confidence bounds			
				γ		$d(G)$	
Power grid network	n	$\hat{\gamma}$	$\hat{d}(G_n)$	Lower	Upper	Lower	Upper
United States	4941	2.08943	0.00054	1.75241	2.12021	0.00048	0.00056
Germany	523	2.62055	0.00642	2.20408	2.78395	0.00586	0.00692

Table 5: The 90% bootstrap confidence intervals for the fragility parameter γ and density $d(G)$ of two power grid networks. Considered 20 seed-wave combinations: waves from 1 to 5, seeds 20, 30, 40, and 50. Number of bootstrap resamples is 500 per each combination. Cross-validation is based on a random selection of 100 vertices 13 times.

Conclusion

In this paper we discuss utility and implementation of the two bootstrap methods for nonparametric inference on complex networks, that is, patchwork bootstrap of Thompson et al. (2016) and Gel et al. (2017) and vertex bootstrap of Snijders and Borgatti (1999). We primarily focus on developing inference on network degree distribution and its functions, e.g., mean and density. Furthermore, we perceive the observed network data as a single realization of some "true" unobserved network, and our target is to draw statistical inference in a model-free data-driven way, given only this single network realization. While there is an ever increasing interest in nonparametric inference on complex networks and despite the fact that the vertex bootstrap has been implemented in UCINET software for social network analysis for more than a decade, to our knowledge, there exists no single implementation of bootstrap methods on graphs in R. Our new package **snowboot** fills this gap and offers a flexible data-driven alternative for parametric analysis of complex networks. Furthermore, **snowboot** is fully compatible with **igraph**, and provides a number of options, such as Labeled Snowball Sampling with Multiple Inclusions and cross-validation on graphs – the functionality of standalone interest for graph mining and network analysis.

Acknowledgements

The authors thank Steve P. Borgatti for the help with the vertex bootstrap and Marti Rosas-Casals for the help with interpretation of γ fragility parameter in the power grid analysis. This work was partially supported by grants from the National Science Foundation (NSF) of the United States, IIS 1633331/1633355, DMS 1736368 and ECCS 1824710.

Bibliography

N. K. Ahmed, N. Duffield, J. Neville, and R. Kompella. Graph sample and hold: a framework for big-graph analytics. In *Proc. of the 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1446–1455, 2014. URL `http://dx.doi.org/10.1145/2623330.2623757`. [p96]

D. Aldous and R. Lyons. Processes on unimodular random networks. *Electronic Journal of Probability*, 12(54):1454–1508, 2007. URL `https://doi.org/10.1214/EJP.v12-463`. [p96, 97]

W. Ali, A. E. Wegner, R. E. Gaunt, C. M. Deane, and G. Reinert. Comparison of large networks with sub-sampling strategies. *Scientific Reports*, 6:28955, 2016. URL `https://doi.org/10.1038/srep28955`. [p97, 102]

S. Bhattacharyya and P. J. Bickel. Subsampling bootstrap of count features of networks. *The Annals of Statistics*, 43(6):2384–2411, 2015. URL `https://doi.org/10.1214/15-AOS1338`. [p97]

S. P. Borgatti, M. G. Everett, and L. C. Freeman. *Ucinet 6 for Windows: Software for Social Network Analysis*. Analytic Technologies, Harvard, MA, 2002. URL https://sites.google.com/site/ucinetsoftware/home. [p95, 102]

G. N. Boshnakov. *Rdpack: Update and Manipulate Rd Documentation Objects*, 2018. URL https://CRAN.R-project.org/package=Rdpack. R package version 0.10-1. [p95]

T. Britton, M. Deijfen, and A. Martin-Löf. Generating simple random graphs with prescribed degree distribution. *J. of Statistical Physics*, 124(6):1377–1397, 2006. URL https://doi.org/10.1007/s10955-006-9168-x. [p96]

I. Brugere, B. Gallagher, and T. Y. Berger-Wolf. Network structure inference, a survey: Motivations, methods, and applications. *arXiv preprint arXiv:1610.00782*, 2017. URL http://arxiv.org/abs/1610.00782. [p95]

G. Burghouwt and R. Redondi. Connectivity in air transport networks: An assessment of models and applications. *J. of Transport Economics and Policy*, 47(1):35–53, 2013. URL http://www.jstor.org/stable/24396351. [p107]

M. R. Chernick. *Bootstrap Methods: A Guide for Practitioners and Researchers*. John Wiley & Sons, Hoboken, NJ, second edition, 2008. URL http://dx.doi.org/10.1002/9780470192573. [p95]

E. Crane. Steady state clusters and the Ráth–Tóth forest fire model. *arXiv preprint arXiv:1809.03462*, 2018. URL https://arxiv.org/abs/1809.03462. [p96, 97]

G. Csárdi and others. *Igraph: Network Analysis and Visualization*, 2018. URL https://CRAN.R-project.org/package=igraph. R package version 1.2.2. [p95]

L. Cuadra, S. Salcedo-Sanz, J. Del Ser, S. Jiménez-Fernández, and Z. W. Geem. A critical review of robustness in power grids using complex networks concepts. *Energies*, 8(9):9211–9265, 2015. URL http://dx.doi.org/10.3390/en8099211. [p109, 110]

A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge, 1997. URL https://doi.org/10.1017/CBO9780511802843. [p100]

A. K. Dey, Y. R. Gel, and H. V. Poor. Motif-based analysis of power grid robustness under attacks. In *Signal and Information Processing (GlobalSIP), 2017 IEEE Global Conference on*, pages 1015–1019. IEEE, 2017. URL https://doi.org/10.1109/GlobalSIP.2017.8309114. [p110]

D. Eddelbuettel, R. Francois, J. J. Allaire, K. Ushey, Q. Kou, N. Russell, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2017. URL https://CRAN.R-project.org/package=Rcpp. R package version 0.12.19. [p95]

B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. URL http://dx.doi.org/10.1214/aos/1176344552. [p95]

S. Epskamp, D. Borsboom, and E. I. Fried. Estimating psychological networks and their accuracy: A tutorial paper. *Behavior Research Methods*, 50(1):195–212, 2018. URL https://doi.org/10.3758/s13428-017-0862-1. [p97]

O. Frank. Structure inference and stochastic graphs. Technical Report AD0687176, Research Institute of National Defence Stockholm, 1968. [p96]

O. Frank. Estimation of graph totals. *Scandinavian Journal of Statistics*, 4(2):81–89, 1977. [p101]

Y. R. Gel, V. Lyubchich, and L. L. Ramirez Ramirez. Bootstrap quantification of estimation uncertainties in network degree distributions (a short version appeared as "Fast patchwork bootstrap for quantifying estimation uncertainties in sparse random networks" in SIGKDD MLG2016). *Sci. Rep.*, 7:5807, 2017. URL http://dx.doi.org/10.1038/s41598-017-05885-x. [p95, 96, 97, 99, 101, 102, 110]

A. Goldenberg, A. X. Zheng, S. E. Fienberg, and E. M. Airoldi. A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2):129–233, 2010. URL http://dx.doi.org/10.1561/2200000005. [p95]

L. A. Goodman. Snowball sampling. *The Annals of Mathematical Statistics*, 32(1):148–170, 1961. URL http://www.jstor.org/stable/2237615. [p96]

M. Granovetter. Network sampling: Some first steps. *American Journal of Sociology*, 81(6):1287–1303, 1976. URL http://doi.org/10.1086/226224. [p96]

P. Hall. *The Bootstrap and Edgeworth Expansion*. Springer-Verlag, New York, 1992. URL http://doi.org/10.1007/978-1-4612-4384-7. [p95]

J. Illenberger and G. Flötteröd. Estimating network properties from snowball sampled data. *Social Networks*, 34(4):701–711, 2012. URL https://doi.org/10.1016/j.socnet.2012.09.001. [p101, 102]

U. Islambekov, A. K. Dey, Y. R. Gel, and H. V. Poor. Role of local geometry in robustness of power grid networks. In *Global Conference on Signal and Information Processing (GlobalSIP), 2018 IEEE*, 2018. [p110]

E. D. Kolaczyk. *Statistical Analysis of Network Data: Methods and Models*. Springer-Verlag, New York, 2009. URL http://doi.org/10.1007/978-0-387-88146-1. [p96, 102]

E. D. Kolaczyk and G. Csárdi. *Statistical Analysis of Network Data with R*, volume 65 of *Use R!* Springer-Verlag, New York, 2014. URL http://doi.org/10.1007/978-1-4939-0983-4. [p95]

L. Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. [p96, 102]

D. Lusseau, H. Whitehead, and S. Gero. Incorporating uncertainty into the study of animal social networks. *Animal Behavior*, 75(5):1809–1815, 2008. URL https://doi.org/10.1016/j.anbehav.2007.10.029. [p97]

D. MacRae. Direct factor analysis of sociometric data. *Sociometry*, 23(4):360–371, 1960. URL http://doi.org/10.2307/2785690. [p102]

C. Matke, W. Medjroubi, and D. Kleinhans. SciGRID – An Open Source Reference Model for the European Transmission Network (v0.2), 2016. URL http://www.scigrid.de. [p109]

M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74:036104, 2006. URL http://doi.org/10.1103/PhysRevE.74.036104. [p106]

P. Orbanz. Subsampling large graphs and invariance in networks. *arXiv preprint arXiv:1710.04217*, 2017. URL https://arxiv.org/abs/1710.04217. [p97, 102]

P. Orbanz and D. M. Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 37(2):437–461, 2015. URL https://doi.org/10.1109/TPAMI.2014.2334607. [p96, 97, 102]

G. A. Pagani and M. Aiello. The Power Grid as a complex network: a survey. *Physica A: Statistical Mechanics and its Applications*, 392(11):2688–2700, 2013. URL http://doi.org/10.1016/j.physa.2013.01.023. [p109, 110]

B. Pearce and G. Doernhoefer. *The Economic Benefits Generated by Alliances and Joint Ventures. IATA Economics Briefing*. IATA Economics, 2011. [p107]

L. L. Ramirez-Ramirez, K. Nezafati, Y. Chen, V. Lyubchich, and Y. R. Gel. *Snowboot: Bootstrap Methods for Network Inference*, 2018. URL https://CRAN.R-project.org/package=snowboot. R package version 1.0.0. [p95]

M. Rosas-Casals and B. Corominas-Murtra. Assessing European power grid reliability by means of topological measures. *WIT Transactions on Ecology and the Environment*, 121:527–537, 2009. URL http://doi.org/10.2495/ESUS090471. [p109]

J. Scott and P. J. Carrington. *The SAGE Handbook of Social Network Analysis*. SAGE Publications, Thousand Oaks, CA, 2011. URL http://dx.doi.org/10.4135/9781446294413. [p96]

J. Shao and D. Tu. *The Jackknife and Bootstrap*. Springer-Verlag, New York, 1995. URL http://dx.doi.org/10.1007/978-1-4612-0795-5. [p95]

O. Simpson, C. Seshadhri, and A. McGregor. Catching the head, tail, and everything in between: a streaming algorithm for the degree distribution. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 979–984. IEEE, 2015. URL http://dx.doi.org/10.1109/ICDM.2015.47. [p96]

M. Smyth and B. Pearce. *Airline Network Benefits. IATA Economics Briefing #3*. IATA Economics, 2006. [p107]

T. A. B. Snijders and S. P. Borgatti. Non-parametric standard errors and tests for network statistics. *Connections*, 22(2):161–170, 1999. [p95, 97, 110]

R. V. Sóle, M. Rosas-Casals, B. Corominas-Murtra, and S. Valverde. Robustness of the European power grids under intentional attack. *Physical Review E*, 77(2):026102, 2008. URL http://doi.org/10.1103/PhysRevE.77.026102. [p109]

M. E. Thompson, L. L. Ramirez Ramirez, V. Lyubchich, and Y. R. Gel. Using the bootstrap for statistical inference on random graphs. *Canadian Journal of Statistics*, 44(1):3–24, 2016. URL http://doi.org/10.1002/cjs.11271. [p95, 96, 97, 98, 99, 102, 110]

R. van der Hofstad. *Random Graphs and Complex Networks*. Cambridge University Press, Cambridge, 2017. URL https://doi.org/10.1017/9781316779422. [p96]

D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. URL http://doi.org/10.1038/30918. [p109]

T. W. Yee. *VGAM: Vector Generalized Linear and Additive Models*, 2018. URL https://CRAN.R-project.org/package=VGAM. R package version 1.0-6. [p95]

Y. Zhang, E. D. Kolaczyk, and B. D. Spencer. Estimating network degree distributions under sampling: An inverse problem, with applications to monitoring social media networks. *Annals of Applied Statistics*, 9(1):166–199, 2015. URL http://doi.org/10.1214/14-AOAS800. [p96]

Yuzhou Chen
Department of Statistical Science
Southern Methodist University
P.O. Box 750332
Dallas, Texas, 75275
USA
E-mail: yuzhouc@smu.edu

Yulia R. Gel, Kusha Nezafati
Department of Mathematical Sciences
University of Texas at Dallas
800 West Campbell Road
Richardson, Texas, 75080
USA
E-mail: ygl@utdallas.edu, kusha.nezafati@utdallas.edu

Vyacheslav Lyubchich
Chesapeake Biological Laboratory
University of Maryland Center for Environmental Science
146 Williams Street / P.O. Box 38
Solomons, Maryland, 20688
USA
E-mail: lyubchich@umces.edu

revengc: An R package to Reverse Engineer Summarized Data

by Samantha Duchscherer, Robert Stewart, and Marie Urban

Abstract Decoupled (e.g. separate averages) and censored (e.g. $>$ 100 species) variables are continually reported by many well-established organizations, such as the World Health Organization (WHO), Centers for Disease Control and Prevention (CDC), and World Bank. The challenge therefore is to infer what the original data could have been given summarized information. We present an R package that reverse engineers censored and/or decoupled data with two main functions. The `cnbinom.pars()` function estimates the average and dispersion parameter of a censored univariate frequency table. The `rec()` function reverse engineers summarized data into an uncensored bivariate table of probabilities.

Introduction

The **revengc** R package was originally developed to help model building occupancy (Stewart et al., 2016). Household size and area of residential structures are typically found in any given national census. If a census revealed the raw data or provided a full uncensored contingency table (household size $\times$ area), computing interior density as people per area would be straightforward. However, household size and area are often reported as decoupled variables (separate univariate frequency tables, average values, or a combination of the two). Furthermore, if a contingency table is provided, it typically left ($<$, $\leq$), right ($>$, $\geq$, $+$), and interval ($-$) censored. This summarized information is problematic for numerous reasons. How can a people per area ratio be calculated when no affiliation between the variables exist? If a census reports a household size average of 5.3, then how many houses are there with 1 person, 2 people, ..., 10 people? If a census reports that there are 100 houses in an area of 26$-$50 square meters, then how many houses are in 26, 27, ..., 50 square meters?

A tool that approximates negative binomial parameters from a censored univariate frequency table as well as estimates interior cells of a contingency table governed by negative binomial and/or Poisson marginals can also be useful for other areas ranging from demographic and epidemiological data to ecological inference problems. For example, population and community ecologists could unpack censored organism counts or average life expectancy values. Moreover, other summarized examples include the average number of births, the number of new disease cases, the number of mutations in a gene or average mutation rate, etc. We attempt to accommodate for various application of count data by offering five scenarios that can be reverse engineered:

1. `cnbinom.pars()` - An univariate frequency table estimates an average and dispersion parameter
2. `rec()` - Decoupled averages estimates an uncensored contingency table of probabilities
3. `rec()` - Decoupled frequency tables estimates an uncensored contingency table of probabilities
4. `rec()` - An average and frequency table estimates an uncensored contingency table of probabilities
5. `rec()` - A censored contingency table estimates an uncensored contingency table of probabilities

This paper proceeds with our reverse engineering methodology for the two main functions, `cnbinom.pars()` and `rec()`. We provide an in-depth analysis of how we implemented both the negative binomial and Poisson distribution as well as the **truncdist** (Nadarajah and Kotz, 2006; Novomestky and Nadarajah, 2016) and **mipfp** R package (Barthélemy and Suesse, 2018a,b). Since the **revengc** package has specific input requirements for both `cnbinom.pars()` and `rec()`, we continue with an explanation of how to format the input tables properly. We then provide coded examples that implements **revengc** on national census data (household size and area) and end with concluding remarks.

Methodology: `cnbinom.pars()`

The methodology for the `cnbinom.pars()` function is relatively straightforward. To estimate an average μ and dispersion r parameter, a censored frequency table is fit to a negative binomial distribution using a maximum log-likelihood function customized to handle left ($<$, $\leq$), right ($>$, $\geq$, $+$), and interval ($-$) censored data. To show an example, first recall the negative binomial distribution $P(X = x \mid \mu, r)$ parameterized as a distribution of the number of failures X before the r^{th} success in independent trials

(1). With success probability p in each trail, $r \geq 0$ and $0 \leq p \leq 1$ (Lindén and Mäntyniemi, 2011).

$$\begin{aligned} P(X = x|r,p) = \binom{x+r-1}{x} p^r (1-p)^y \equiv P(X = x|r,\mu) \binom{x+r-1}{x} \left(\frac{r}{\mu+r}\right)^r \left(\frac{\mu}{\mu+r}\right)^y \\ E(X) = \frac{r(1-p)}{p} = \mu \\ V(X) = \frac{r(1-p)}{p^2} = \mu + \frac{\mu^2}{r} \end{aligned} \tag{1}$$

Now consider an arbitrary censored frequency table x that has a combination of left censored ($x < c$), interval censored ($a \leq x \leq b$), and right censored ($x > d$) data (i.e. a, b, c, and d represent the censoring limits). The optimal μ and r parameter for x maximizes its custom log-likelihood function (2).

$$\begin{aligned} L(\mu, r|x)_{\log} = & \\ & + \sum_{x<c} \log\left(P\left(x < c|\mu, r\right)\right) \\ & + \sum_{a \leq x \leq b} \log\left(P\left(a \leq x \leq b|\mu, r\right)\right) \\ & + \sum_{x>d} \log\left(P\left(x > d|\mu, r\right)\right) \end{aligned} \tag{2}$$

Methodology: `rec()`

Overview

`rec()` is a statistical approach that estimates the probabilities of a 'true' contingency table given summarized information: two averages, two univariate frequency tables, a combination of an average and univariate frequency table, and a censored contingency table. Figure 1 presents a methodology workflow.

Negative Binomial and Poisson Distribution

When only an average is provided, we assume the average and variance are equal and rely on a Poisson distribution (i.e. the probability of observing x events in a given interval is given by Equation 3). We understand that there are many cases where data has more variation than what is indicated by the Poisson distribution (e.g. overdispersion). However, with limited data, the Poisson distribution is implemented due to its convenient property of having only one parameter, λ = average. For the cases with more data (e.g. univariate frequency table(s) or censored contingency table), we account for dispersion by relying on the more flexible negative binomial distribution (1). Hence, in these cases, the `cnbinom.pars()` function estimates the optimal average μ and dispersion r parameters.

$$\begin{aligned} P(X = x) = e^{-\lambda} \frac{\lambda^x}{x!} \\ E(X) = \lambda \\ V(X) = \lambda \end{aligned} \tag{3}$$

The truncdist R package

With the negative binomial (μ and r) and/or Poisson (λ) parameters, `rec()` calculates truncated distributions to represent uncensored row (`Xlowerbound:Xupperbound`) and column (`Ylowerbound:Yupperbound`) margins. Calculations use the **truncdist** R package, and to provide a reference, Equation 4 gives the probability density function of a truncated X distribution over the interval *(a,b]* (i.e. the negative binomial and/or Poisson probability density function is represented by $g(\cdot)$ and their corresponding cumulative distribution function is denoted by $G(\cdot)$). Note, truncated distributions are very practical in this context because the distributions (margins) are restricted to a desired row and column length.

$$f_X(x) = \begin{cases} \frac{g(x)}{G(b)-G(a)}, & \text{if } a < x \leq b \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The $(a, b]$ interval needed for both X (row of contingency table) and Y (column of contingency table) can be selected intuitively or with a brute force method. If `rec()` outputs a final contingency

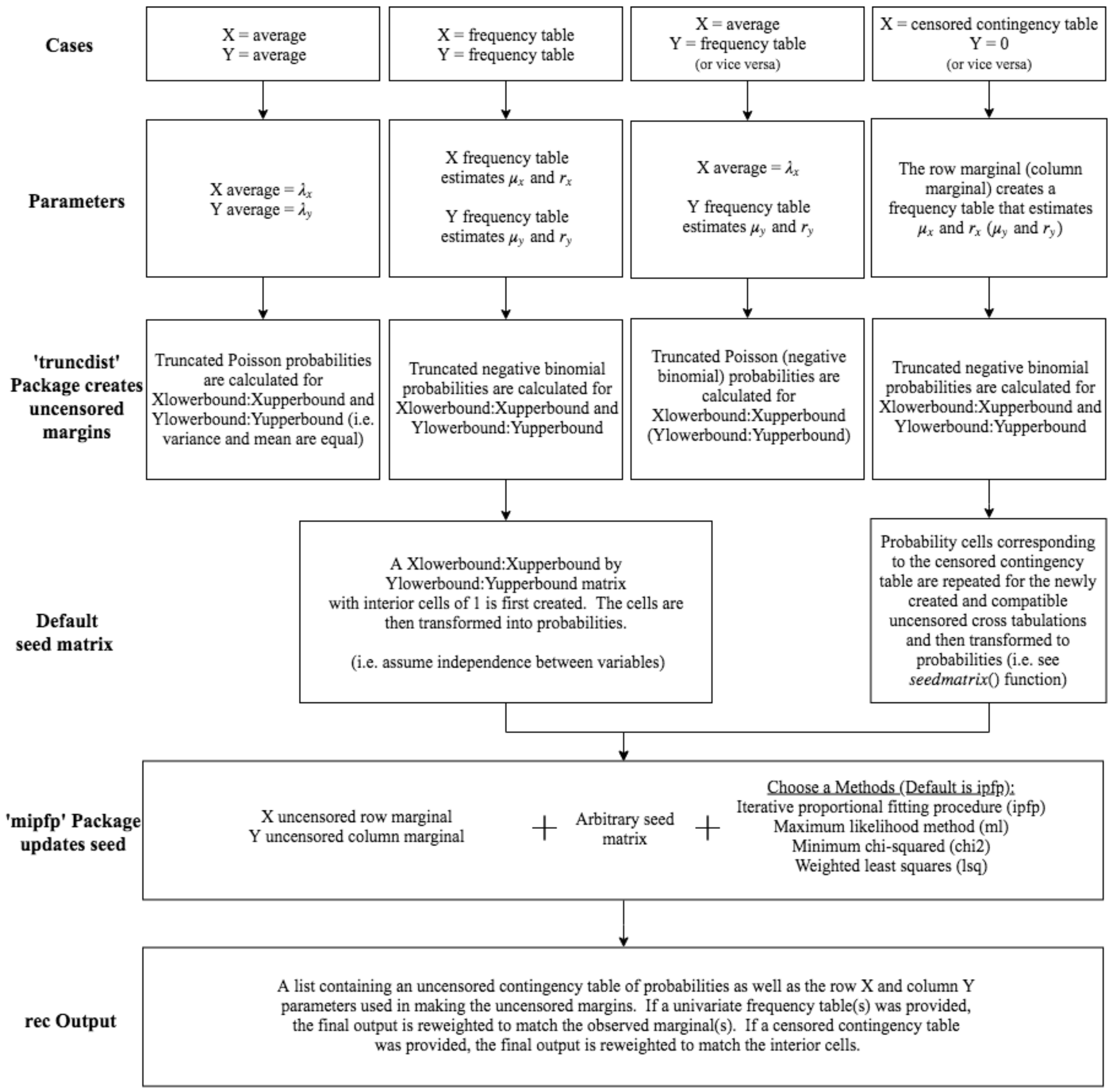

Cases
X = average
Y = average
X = frequency table
Y = frequency table
X = average
Y = frequency table
(or vice versa)
X = censored contingency table
Y = 0
(or vice versa)
Parameters
X average = λ_x
Y average = λ_y
X frequency table estimates μ_x and r_x
Y frequency table estimates μ_y and r_y
X average = λ_x
Y frequency table estimates μ_y and r_y
The row marginal (column marginal) creates a frequency table that estimates μ_x and r_x (μ_y and r_y)
'truncdist' Package creates uncensored margins
Truncated Poisson probabilities are calculated for Xlowerbound:Xupperbound and Ylowerbound:Yupperbound (i.e. variance and mean are equal)
Truncated negative binomial probabilities are calculated for Xlowerbound:Xupperbound and Ylowerbound:Yupperbound
Truncated Poisson (negative binomial) probabilities are calculated for Xlowerbound:Xupperbound (Ylowerbound:Yupperbound)
Truncated negative binomial probabilities are calculated for Xlowerbound:Xupperbound and Ylowerbound:Yupperbound
Default seed matrix
A Xlowerbound:Xupperbound by Ylowerbound:Yupperbound matrix with interior cells of 1 is first created. The cells are then transformed into probabilities.
(i.e. assume independence between variables)
Probability cells corresponding to the censored contingency table are repeated for the newly created and compatible uncensored cross tabulations and then transformed to probabilities (i.e. see *seedmatrix*() function)
'mipfp' Package updates seed
X uncensored row marginal
Y uncensored column marginal
+
Arbitrary seed matrix
+
Choose a Methods (Default is ipfp):
Iterative proportional fitting procedure (ipfp)
Maximum likelihood method (ml)
Minimum chi-squared (chi2)
Weighted least squares (lsq)
rec Output
A list containing an uncensored contingency table of probabilities as well as the row X and column Y parameters used in making the uncensored margins. If a univariate frequency table(s) was provided, the final output is reweighted to match the observed marginal(s). If a censored contingency table was provided, the final output is reweighted to match the interior cells.

Figure 1: Workflow of `rec()` function.

table with higher probabilities near the edge(s) of the table, then it would make sense to increase the range of the bound(s). For both variables, this would just involve making the lower bound less, making the upper bound more, or doing a combination of the two. The opposite holds true as well. If the final contingency table in `rec()` has very low probabilities near the edge(s) of the table, the range of the particular bound(s) should be decreased.

The mipfp R package

`rec()` utilizing the **mipfp** R package to calculate cross tabulation probability estimates, and **mipfp** requires fixed marginals, a seed estimation method, and a seed matrix. The row and column marginals are uncensored truncated distributions (see The **truncdist** R package section) while opportunities for sensitivity analysis are presented with the seed estimation method and seed matrix. For example, **mipfp** offers four seed estimation methods (Table 1); the default method in `rec()` is the iterative proportional fitting procedure. Although the algorithms vary, they all adjust cell proportions p_{xy} in a $X \times Y$ contingency table to known marginal probabilities, π_{x+} and π_{+y} (i.e. all interior cell estimates $\hat{\pi}_{xy}$ are subject to marginal constraints (5)). For an expanded explanation of these methods, please refer to Little and Wu (1991) and Suesse et al. (2017).

$$\begin{aligned} &\sum_y \hat{\pi}_{x+} \qquad (x = 1, ..., X) \\ &\sum_x \hat{\pi}_{+y} \qquad (y = 1, ..., Y) \end{aligned} \tag{5}$$

Method	**Calculate $\hat{\pi}_{xy}$ by**
Iterative proportional fitting procedure - ipfp	Minimizing $\sum_x \sum_y \hat{\pi}_{xy} ln(\hat{\pi}_{xy}/p_{xy})$
Maximum likelihood method - ml	Maximizing $\sum_x \sum_y p_{xy} ln(\hat{\pi}_{xy})$
Minimum chi-squared - chi2	Minimizing $\sum_x \sum_j (\hat{\pi}_{xy} - p_{xy})^2/\hat{\pi}_{xy}$
Weighted least squares - lsq	Minimizing $\sum_x \sum_y (p_{xy} - \hat{\pi}_{xy})^2/p_{xy}$

Table 1: Algorithms to generate estimated cross tabulations.

The seed matrix input can be arbitrary, but `rec()` provides reasonable defaults. For the decoupled cases (two averages, two tables, or a combination of a table and average), the absence of additional information makes it difficult to say much about the joint distribution. Therefore, `rec()` assumes independence between the variables, which is equivalent in making the $X \times Y$ seed a matrix of ones; `rec()` converts the matrix of ones to probabilities. When a censored contingency table is provided, independence does not have to be assumed and the interior cells can be weighted. `rec()` creates the default seed matrix by first repeating probability cells, which correspond to the censored contingency table, for the newly created and compatible uncensored cross tabulations. Just as in the decoupled cases, the cell values in this matrix are also changed to probabilities. To see an example of the default seed for a censored contingency table, see Worked examples section.

Usage

cnbinom.pars()

The `cnbinom.pars()` function has the following format:

```
cnbinom.pars(censoredtable)
```

where `censoredtable` is a frequency table (censored and/or uncensored). A data.frame and matrix are acceptable classes. See Data entry section for formatting. The output is a list consisting of an estimated average μ and dispersion parameter r.

rec()

The `rec()` function has the following format:

```
rec(X, Y, Xlowerbound, Xupperbound, Ylowerbound, Yupperbound,
  seed.matrix, seed.estimation.method)
```

where

- `X`: Argument can be an average, a univariate frequency table, or a censored contingency table. The average value should be a numeric class while a data.frame or matrix are acceptable table classes. `Y` defaults to `NULL` if `X` argument is a censored contingency table. See Data entry section for formatting.
- `Y`: Same description as `X` but this argument is for the `Y` variable. `X` defaults to `NULL` if `Y` argument is a censored contingency table.
- `Xlowerbound`: A numeric class value to represent the left bound for `X` (row in contingency table). The value must strictly be a non-negative integer and cannot be greater than the lowest category/average value provided for `X` (e.g. the lower bound cannot be 6 if a table has '< 5' as a `X` or row category).
- `Xupperbound`: A numeric class value to represent the right bound for `X` (row in contingency table). The value must strictly be a non-negative integer and cannot be less than the highest category/average value provided for `X` (e.g. the upper bound cannot be 90 if a table has '> 100' as a `X` or row category).
- `Ylowerbound`: Same description as `Xlowerbound` but this argument is for `Y` (column in contingency table).
- `Yupperbound`: Same description as `Xupperbound` but this argument is for `Y` (column in contingency table).
- `seed.matrix`: An initial probability matrix to be updated. If decoupled variables is provided the default is a `Xlowerbound:Xupperbound` by `Ylowerbound:Yupperbound` matrix with interior cells of 1, which are then converted to probabilities. If a censored contingency table is provided the default is the `seedmatrix()$Probabilities` output.
- `seed.estimation.method`: A character string indicating which method is used for updating the `seed.matrix`. The choices are: `"ipfp"`, `"ml"`, `"chi2"`, or `"lsq"`. Default is `"ipfp"`.

The output is a list containing an uncensored contingency table of probabilities (rows range from `Xlowerbound:Xupperbound` and the columns range from `Ylowerbound:Yupperbound`) as well as the row and column parameters used in making the margins for the **mipfp** R package.

Data entry

The input tables are formatted to accommodate most open source data. The univariate frequency table used in `cnbinom.pars()` and/or `rec()` needs to be a data.frame or matrix class with two columns and n rows. The categories must be in the first column with the frequencies or probabilities in the second column. Row names should never be placed in this table (the default row names should always be 1:n). Column names can be any character string. The only symbols accepted for censored data are listed below.

- Left censored symbols: $<$, $\leq$, L, and LE
- Interval censored symbols: $-$ and I (symbol has to be placed in the middle of the two category values)
- Right censored symbols: $>$, $\geq$, $+$, G, and GE
- Uncensored symbol: no symbol (only provide category value)

Note, less than or equal to ($\leq$ and LE) is not equivalent to less than ($<$ and L) and greater than or equal to ($\geq$, $+$, and GE) is not equivalent to greater than ($>$ and G). **revengc** also uses closed intervals. Table 2 shows three different examples that all give the same `cnbinom.pars()` output.

The censored contingency table for `rec()` has a similar format. The censored symbols should follow the requirements listed above. The table's class can be a data.frame or a matrix. The column names should be the Y category values. The first column should be the X category values and the row names can be arbitrary. The inside of the table are $X \times Y$ frequencies or probabilities; the tabulations must be non-negative if the `seed.estimation.method = "ipfp"` or strictly positive if the `seed.estimation.method` is `"ml"`, `"lsq"`, or `"chi2"`. The row X and column Y marginal totals need to be placed in this table. The top left, top right, and bottom left corners of the table should be `NA` or blank. The bottom right corner can be a total cross tabulation sum value, `NA`, or blank. Table 3 is a formatted example.

Category	Frequency	Category	Frequency	Category	Frequency
≤ 6	11800	LE 6	11800	< 7	11800
7-12	57100	7 I 12	57100	7 I 12	57100
13-19	14800	13 I 19	14800	13-19	14800
20+	3900	GE 20	3900	≥ 20	3900

Table 2: Examples of correctly formatted univariate tables.

NA	<20	20-30	>30	NA
<5	18	19	8	45
5-9	13	8	12	33
≥10	7	5	10	21
NA	38	32	31	NA

Table 3: Example of a correctly formatted bivariate table.

Formatting tables in R

The code below shows how to format these tables properly in R.

```
# create univariate table
# note univariatetable.csv is a preloaded example that provides the same table
univariatetable <- cbind(as.character(c("1-2", "3-4", "5-6", "7-8", ">=9")),
  c(16.2, 41.7, 29.0, 9.0, 4.1))

# create contingency table
# note contingencytable.csv is a preloaded example that provides the same table
# fill a matrix
contingencytable <- matrix(c(
  6185, 9797, 16809, 11126, 6156, 3637, 908, 147, 69, 4,
  5408, 12748, 26506, 21486, 14018, 9165, 2658, 567, 196, 78,
  7403, 20444, 44370, 36285, 23576, 15750, 4715, 994, 364, 136,
  4793, 17376, 44065, 40751, 28900, 20404, 6557, 1296, 555, 228,
  2354, 11143, 32837, 33910, 26203, 19301, 6835, 1438, 618, 245,
  1060, 6038, 19256, 21298, 17774, 13864, 4656, 1039, 430, 178,
  273, 2521, 9110, 11188, 9626, 7433, 2608, 578, 196, 112,
  119, 1130, 4183, 5566, 5053, 3938, 1367, 318, 119, 66,
  33, 388, 1707, 2367, 2328, 1972, 719, 171, 68, 37,
  38, 178, 1047, 1672, 1740, 1666, 757, 193, 158, 164),
  nrow = 10, ncol = 10, byrow = TRUE)
# calculate row marginal
rowmarginal <- apply(contingencytable, 1, sum)

# add row marginal to matrix
contingencytable <- cbind(contingencytable, rowmarginal)

# calculate column marginal
colmarginal <- apply(contingencytable, 2, sum)

# add column marginal to matrix
contingencytable <- rbind(contingencytable, colmarginal)

# remove row names
row.names(contingencytable)[row.names(contingencytable) == "colmarginal"]<-""

# add row names as first column
contingencytable <- data.frame(c("1", "2", "3", "4", "5", "6", "7", "8", "9",
  "10+", NA), contingencytable)
```

```
# add column names
colnames(contingencytable) <- c(NA, "<20", "20-29", "30-39", "40-49",
  "50-69", "70-99", "100-149", "150-199", "200-299", "300+", NA)
```

Worked examples

Nepal

A Nepal Living Standards Survey (Government of Nepal, National Planning Commission Secretariat, 2011) provides both a censored table and average for urban household size. We use the censored table to show that the cnbinom.pars() function calculates a close approximation to the provided average household size (4.4 people). Note, there is overdispersion in the data.

```
# revengc has the Nepal household table preloaded as univariatetable.csv
cnbinom.pars(censoredtable = univariatetable.csv)
```

Indonesia

In 2010, the Population Census Data - Statistics Indonesia provided over 60 censored contingency tables containing household member size by floor area of dwelling unit (square meter) (Statistics Indonesia, 2010). The tables are separated by province, urban, and rural. Here we use the rural Aceh Province table to show the multiple coding steps and functions implemented inside rec(). This allows the user to see a methodology workflow in code form. The final uncensored household size by area estimated probability table, which implemented the "ipfp" seed estimation method and default seed matrix, has rows ranging from 1 (Xlowerbound) to 15 (Xupperbound) people and columns ranging from 10 (Ylowerbound) to 310 (Yupperbound) square meters.

```
# Packages needed if doing workflow of rec() step by step
library(stringr)
library(dplyr)
library(mipfp)
library(truncdist)
library(revengc)

# data = Indonesia's rural Aceh Province censored contingency table
# preloaded in revengc as 'contingencytable.csv'
contingencytable.csv

# provided upper and lower bound values for table
# X = row and Y = column
Xlowerbound = 1
Xupperbound = 15
Ylowerbound = 10
Yupperbound = 310

# table of row marginals provides average and dispersion for x
row.marginal.table <- row.marginal(contingencytable.csv)
x <- cnbinom.pars(row.marginal.table)
# table of column marginals provides average and dispersion for y
column.marginal.table <- column.marginal(contingencytable.csv)
y <- cnbinom.pars(column.marginal.table)

# create uncensored row and column ranges
rowrange <- Xlowerbound:Xupperbound
colrange <- Ylowerbound:Yupperbound

# new uncensored row marginal table = truncated negative binomial distribution
uncensored.row.margin <- dtrunc(rowrange, mu=x$Average, size = x$Dispersion,
  a = Xlowerbound-1, b = Xupperbound, spec = "nbinom")
# new uncensored column margin table = truncated negative binomial distribution
uncensored.column.margin <- dtrunc(colrange, mu=y$Average, size = y$Dispersion,
  a = Ylowerbound-1, b = Yupperbound, spec = "nbinom")
```

```
# sum of truncated distributions equal 1
# margins need to be equal for mipfp
sum(uncensored.row.margin)
sum(uncensored.column.margin)

# create seed of probabilities (rec() default)
seed.output <- seedmatrix(contingencytable.csv, Xlowerbound,
  Xupperbound, Ylowerbound, Yupperbound)$Probabilities

# run mipfp
# store the new margins in a list
tgt.data <- list(uncensored.row.margin, uncensored.column.margin)
# list of dimensions of each marginal constrain
tgt.list <- list(1,2)
# calling the estimated function
## seed has to be in array format for mipfp package
## ipfp is the selected seed.estimation.method
final1 <- Estimate(array(seed.output, dim=c(length(Xlowerbound:Xupperbound),
  length(Ylowerbound:Yupperbound))), tgt.list, tgt.data, method="ipfp")$x.hat

# filling in names of updated seed
final1 <- data.frame(final1)
row.names(final1) <- Xlowerbound:Xupperbound
names(final1) <- Ylowerbound:Yupperbound

# reweight estimates to known censored interior cells
final1 <- reweight.contingencytable(observed.table = contingencytable.csv,
  estimated.table = final1)

# final results are probabilities
sum(final1)

# rec() function outputs the same table
# default of rec() seed.estimation.method is ipfp
# default of rec() seed.matrix is the output of seedmatrix()$Probabilities
final2<-rec(
  X = contingencytable.csv,
  Xlowerbound = 1,
  Xupperbound = 15,
  Ylowerbound = 10,
  Yupperbound = 310)

# check that final1 and final2 have the same results
all(final1 == final2$Probability.Estimates)
```

Conclusion

revengc was designed to reverse engineer summarized and decoupled variables with two main functions: cnbinom.pars() and rec(). Relying on a negative binomial distribution, cnbinom.pars() approximates the average and dispersion parameter of a censored univariate frequency table. rec() fills in missing interior cell values from observed aggregated data (e.g. decoupled average(s) and/or censored frequency table(s) or a censored contingency table). It is worth noting the required assumptions in rec(). For instance, rec() relies on a Poisson distribution when only an average is provided, which is assuming the variance and average are equal. More descriptive input variables, such as univariate frequency tables or contingency tables, can account for dispersion found in data. However, independence between decoupled variables still has to be assumed when there is no external information about the joint distribution. For these reasons, **revengc** provides two options for sensitivity analysis: the seed matrix and the method used in updating the seed matrix are both arbitrary inputs.

Acknowledgments

This manuscript has been authored by employees of UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy. Accordingly, the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The publisher, by accepting the article for publication, acknowledges the above-mentioned conditions.

Bibliography

J. Barthélemy and T. Suesse. mipfp: An R package for multidimensional array fitting and simulating multivariate bernoulli distributions. *Journal of Statistical Software, Code Snippets*, 86(2):1–20, 2018a. URL https://doi.org/10.18637/jss.v086.c02. [p114]

J. Barthélemy and T. Suesse. *mipfp: Multidimensional Iterative Proportional Fitting and Alternative Models*, 2018b. URL https://CRAN.R-project.org/package=mipfp. R package version 3.2.1. [p114]

Government of Nepal, National Planning Commission Secretariat. Nepal living standards survey. Technical report, Central Bureau of Statistics, 2011. [p120]

A. Lindén and S. Mäntyniemi. Using the negative binomial distribution to model overdispersion in ecological count data. *Ecology*, 92(7):1414–1421, 2011. URL https://doi.org/10.1890/10-1831.1. [p115]

R. J. Little and M.-M. Wu. Models for contingency tables with known margins when target and sampled populations differ. *Journal of the American Statistical Association*, 86(413):87–95, 1991. URL https://doi.org/10.2307/2289718. [p117]

S. Nadarajah and S. Kotz. R programs for truncated distributions. *Journal of Statistical Software, Code Snippets*, 16(2):1–8, 2006. URL https://doi.org/10.18637/jss.v016.c02. [p114]

F. Novomestky and S. Nadarajah. *truncdist: Truncated Random Variables*, 2016. URL https://CRAN.R-project.org/package=truncdist. R package version 1.0-2. [p114]

Statistics Indonesia. Household by floor area of dwelling unit and households member size. Technical report, The 2010 Indonesia Population Census, 2010. [p120]

R. Stewart, M. Urban, S. Duchscherer, J. Kaufman, A. Morton, G. Thakur, J. Piburn, and J. Moehl. A Bayesian machine learning model for estimating building occupancy from open source data. *Natural Hazards*, 81(3):1929–1956, 2016. URL https://doi.org/10.1007/s11069-016-2164-9. [p114]

T. Suesse, M.-R. Namazi-Rad, P. Mokhtarian, and J. Barthélemy. Estimating cross-classified population counts of multidimensional tables: an application to regional Australia to obtain pseudo-census counts. *Journal of Official Statistics*, 33(4), 2017. URL https://doi.org/10.1515/jos-2017-0048. [p117]

Samantha Duchscherer
Oak Ridge National Laboratory
1 Bethel Valley Road Oak Ridge, TN 37831
USA
ORCiD: 0000-0002-2023-3106
sam.duchscherer@gmail.com

Robert Stewart
Oak Ridge National Laboratory
1 Bethel Valley Road Oak Ridge, TN 37831
USA
ORCiD: 0000-0002-8186-7559
stewartrn@ornl.gov

Marie Urban
Oak Ridge National Laboratory
1 Bethel Valley Road Oak Ridge, TN 37831
USA
ORCiD: 0000-0001-9571-832X
urbanml@ornl.gov

Basis-Adaptive Selection Algorithm in dr-package

by Jae Keun Yoo

Abstract Sufficient dimension reduction (SDR) turns out to be a useful dimension reduction tool in high-dimensional regression analysis. Weisberg (2002) developed the **dr**-package to implement the four most popular SDR methods. However, the package does not provide any clear guidelines as to which method should be used given a data. Since the four methods may provide dramatically different dimension reduction results, the selection in the **dr**-package is problematic for statistical practitioners. In this paper, a basis-adaptive selection algorithm is developed in order to relieve this issue. The basic idea is to select an SDR method that provides the highest correlation between the basis estimates obtained by the four classical SDR methods. A real data example and numerical studies confirm the practical usefulness of the developed algorithm.

Introduction

Sufficient dimension reduction (SDR) in the regression of $y \in \mathbb{R}^1|\mathbf{X} \in \mathbb{R}^p = (x_1, \ldots, x_p)^{\mathrm{T}}$ replaces the original p-dimensional predictors $\mathbf{X}$ with its lower-dimensional linearly transformed predictors $\mathbf{M}^{\mathrm{T}}\mathbf{X}$ without any loss of information on $y|\mathbf{X}$, which is equivalently expressed:

$$y \amalg \mathbf{X}|\mathbf{M}^{\mathrm{T}}\mathbf{X}, \tag{1}$$

where $\amalg$ stands for independence, $\mathbf{M}$ is a $p \times q$ matrix and $q \leq p$.

A space spanned by the columns of $\mathbf{M}$ to satisfy (1) is called a dimension reduction subspace. Hereafter, $\mathcal{S}(\mathbf{M})$ denotes the column subspace of a $p \times q$ matrix $\mathbf{M}$, and $\mathbf{M}^{\mathrm{T}}\mathbf{X}$ is called a sufficient predictor. The intersection of all possible dimension reduction subspaces is called the *central subspace* $\mathcal{S}_{y|\mathbf{X}}$, if it exists. The main goal of SDR is to infer $\mathcal{S}_{y|\mathbf{X}}$, which is done through the estimations of its true structural dimension d and orthonormal basis matrix.

According to Cook (1998a), for a non-singular transformation of $\mathbf{X}$ such that $\mathbf{Z} = \mathbf{A}^{\mathrm{T}}\mathbf{X}$, the following relation holds: $\mathcal{S}_{y|\mathbf{X}} = \mathbf{A}\mathcal{S}_{y|\mathbf{Z}}$. Considering a standardized predictor $\mathbf{Z} = \mathbf{\Sigma}^{-1/2}\{\mathbf{X} - E(\mathbf{X})\}$, we have that $\mathcal{S}_{y|\mathbf{X}} = \mathbf{\Sigma}^{-1/2}\mathcal{S}_{y|\mathbf{Z}}$, where $\mathbf{\Sigma} = cov(\mathbf{X})$ and $\mathbf{\Sigma}^{-1/2}\mathbf{\Sigma}^{-1/2} = \mathbf{\Sigma}^{-1}$. Typically, SDR methods estimate $\mathcal{S}_{y|\mathbf{Z}}$ first, then back-transform it to $\mathcal{S}_{y|\mathbf{X}}$. Hereafter kernel matrices to restore $\mathcal{S}_{y|\mathbf{Z}}$ for each method will be denoted as $\mathbf{M}_\bullet$, and it will be assumed that $\mathbf{M}_\bullet$ is exhaustively informative to $\mathcal{S}_{y|\mathbf{Z}}$ so that $\mathcal{S}(\mathbf{M}_\bullet) = \mathcal{S}_{y|\mathbf{Z}}$. With $\mathbf{Z}$-scale predictors, the classical but most popularly used SDR methodologies include:

(i) **sliced inverse regression** (SIR; Li, 1991):
$\mathbf{M}_{\mathrm{SIR}} = cov\{E(\mathbf{Z}|y)\}$. If y is categorical, a sample version of $E(\mathbf{Z}|y)$ is straightforward. If y is many-valued or continuous, y is categorized by dividing its range into h slices.

(ii) **sliced average variance estimation** (SAVE; Cook and Weisberg, 1991):
$\mathbf{M}_{\mathrm{SAVE}} = E[\{\mathbf{I}_p - cov(\mathbf{Z}|y)\}\{\mathbf{I}_p - cov(\mathbf{Z}|y)\}^{\mathrm{T}}]$. The sample version of $cov(\mathbf{Z}|y)$ is constructed in the same way as SIR.

(iii) **principal Hessian directions** (pHd; Li, 1992; Cook, 1998b):
$\mathbf{M}_{\mathrm{pHd}} = \mathbf{\Sigma}_{rzz} = E[\{y - E(y) - \beta^{\mathrm{T}}\mathbf{Z}\}\mathbf{Z}\mathbf{Z}^{\mathrm{T}}]$, where β is the ordinary least squares on the regression of $y|\mathbf{Z}$. The sample version of $\mathbf{M}_{\mathrm{pHd}}$ is constructed by replacing the population quantities with their usual moment estimators.

(iv) **covariance method** (cov$k$ Yin and Cook, 2002):
$\mathbf{M}_{\mathrm{cov}k} = \{E(\mathbf{Z}w), E(\mathbf{Z}w^2), \ldots, E(\mathbf{Z}w^k)\}$, where $w = \{y - E(y)\}/\sqrt{\mathrm{var}(y)}$. As can be easily seen, $E(\mathbf{Z}w^j)$ in $\mathbf{M}_{\mathrm{cov}k}$ is the covariance between w^j and $\mathbf{Z}$ as well as the ordinary least squares coefficient of $w^j|\mathbf{Z}$ for $j = 1, \ldots, k$.

These four SDR methods can be implemented in the **dr**-package. The package provides the basis estimates and dimension test results. A detailed review on the **dr**-package is given in Weisberg (2002). The following question is a critical issue in using **dr**: *which one among the four methods has to be used?* Even with the same data, one can obtain dramatically different dimension reduction results from the four methods in **dr**, yet there is no clear and up-to-date guideline on method selection. It is known that SIR and cov$k$ work better when a linear trend exists in regression, while SAVE and pHd are effective under a nonlinear trend, particularly in the case of a quadratic relationship. However, in practice, these may not be useful guidelines for choosing an SDR method, because it is not easy to check the existence of non/linear trends in regression. Even in the case that a linear trend exists, it is still not clear which of SIR and cov*k* should be used, as they yield different dimension test results.

The purpose of the paper is to develop a basis-adaptive selection algorithm for selecting an SDR method. The basic idea is to select the one that gives the highest correlation between the basis estimates obtained by all possible pairs of the four SDR methods. To measure the correlation, a trace correlation *r* (Hooper, 1959) will be used. The algorithm is data-driven and can be enhanced to other SDR methods, although it is highlighted with the use of the **dr**-package in this paper.

The organization of this article is as follows. We develop the idea of a basis-adaptive selection algorithm, and discuss its selection criteria. Next, we present a real data example and simulation studies. Finally, we summarize our work.

Basis-adaptive selection

Development of algorithm

We begin this section with soil evaporation data in Section 6 of Yin and Cook (2002). The data contains 46 observations of daily soil evaporation, daily air and soil temperature curves, daily humidity curves and wind speed. For illustration purposes, we consider a regression analysis of daily soil evaporation given the integrated area of and the range of the daily air and soil temperatures. We will revisit the data in a later section.

```
## loading data
evaporat <- read.table("evaporat.txt", header=T); attach(evaporat)
Rat <- Maxat-Minat; Rvh <- Maxh-Minh; Rst <- Maxst-Minst
w <- c(scale(evaporat$Evap,center=TRUE,scale=TRUE)); detach(evaporat)
evaporat <- data.frame(evaporat, Rat, Rvh, Rst)
w2 <- cbind(w, w^2); w3 <- cbind(w2, w^3); w4 <- cbind(w3, w^4)

## dr-package fitting
library(dr)
sir5 <- dr(Evap~Avat+Rat+Avst+Rst, data=evaporat, method="sir", nslice=5)
save5 <- update(sir5, method="save"); phdres <- update(sir5, method="phdres")
cov2 <- dr(w2~Avat+Rat+Avst+Rst,data=evaporat, method="ols")
cov3 <- dr(w3~Avat+Rat+Avst+Rst,data=evaporat, method="ols")
cov4 <- dr(w4~Avat+Rat+Avst+Rst,data=evaporat, method="ols")

## dimension test
round(dr.test(sir5),3)
round(dr.test(save5),3)
round(dr.test(phdres, numdir=4),3)
set.seed(100);round(dr.permutation.test(cov2,npermute=1000)$summary,3)
set.seed(100);round(dr.permutation.test(cov3,npermute=1000)$summary,3)
set.seed(100);round(dr.permutation.test(cov4,npermute=1000)$summary,3)
```

Table 1: Dimension test results for soil evaporation data

	SIR with 5 slices	SAVE with 5 slices	pHd	cov2	cov3	cov4
$H_0 : d = 0$	0.000	0.048	0.698	0.000	0.000	0.002
$H_0 : d = 1$	0.022	0.261	0.723	0.007	0.001	0.001
$H_0 : d = 2$	0.202	0.546	0.751	N/A	0.011	0.001
$H_0 : d = 3$	0.814	0.755	0.452	N/A	N/A	0.004

The *p*-values for the dimension estimation by the four methods in **dr** are reported in Table 1. For SAVE and pHd, the *p*-values under normal distributions are reported. Since a permutation test should be used for cov*k*, `set.seed(100)` was used to have reproducible results. According to Table 1, with level 5%, the SIR and the SAVE determine that $\hat{d} = 2$ and $\hat{d} = 1$, respectively. The pHd estimates that $\hat{d} = 0$, while the cov*k* estimates that $\hat{d} \geq 4$. In the evaporation data, the dimension estimation results are completely different for each of the four methods. Then, what methodological result should we use for the dimension reduction of the data? Unfortunately, there is no clear guidance for this issue.

Before developing selection criteria among the four SDR methods, there are some aspects that must be considered first. A selection based on the criteria should be data-driven, unless it is purposely pre-selected. In addition, the criteria should be generally extended to the other SDR methods, although the four methods in **dr** are highlighted in this paper.

In order to have some idea of how to select a method, consider a simulated example of $y|\mathbf{X} = (x_1, \ldots, x_{10})^{\mathrm{T}} = x_1 + \varepsilon$. The column of $\boldsymbol{\eta} = (1, 0, \ldots, 0)^{\mathrm{T}}$ spans $\mathcal{S}_{y|\mathbf{Z}}$, and the sufficient predictor of x_1 is well estimated not only by SIR but also by cov2. Therefore, it is expected that the estimates by SIR and cov2 will be more highly correlated than those of any other pairs of the four methods. Assuming that $\mathbf{X}$ and ε are independently normally-distributed with $\mu = 0$ and $\sigma^2 = 0.1^2$, the averages of the absolute correlations between the pairs of the estimates from the four methods as well as between x_1 and the estimate from each method are computed through 500 iterations.

```
sir.cov <- sir.save <- sir.phd <- save.cov <- save.phd <- phd.cov <- NULL
sir.eta <- save.eta <- phd.eta <- cov.eta <- NULL

set.seed(1)

## starting loop
for (i in 1:500){

## model construction
  X <- matrix(rnorm(100*10), c(100,10)); y <- X[,1] + rnorm(100)
  w <- c(scale(y,center=TRUE,scale=TRUE)); w2 <- cbind(w, w^2)

## obtaining basis estimates from the SDR methods
  dir.sir5<-dr.direction(dr(y~X, method="sir", nslice=5))[,1]
  dir.cov2<-dr.direction(dr(w2~X, method="ols"))[,1]
  dir.save5<-dr.direction(dr(y~X, method="save", nslice=5))[,1]
  dir.phd <-dr.direction(dr(y~X, method="phdres"))[,1]

## computing the absolute correlation between the pairs of the estimates from the four methods
  sir.cov[i]<-abs(cor(dir.sir5, dir.cov2)); sir.save[i]<-abs(cor(dir.sir5, dir.save5))
  sir.phd[i]<-abs(cor(dir.sir5, dir.phd)); save.cov[i]<-abs(cor(dir.save5, dir.cov2))
  save.phd[i]<-abs(cor(dir.save5, dir.phd)); phd.cov[i]<-abs(cor(dir.phd, dir.cov2))

## computing the absolute correlation between the estimate from each method and x1
  sir.eta[i]<-abs(cor(dir.sir5, X[,1])); cov.eta[i]<-abs(cor(dir.cov2, X[,1]))
  save.eta[i]<-abs(cor(dir.save5, X[,1])); phd.eta[i]<-abs(cor(dir.phd, X[,1]))
 }

## the averages of the absolute correlations by each pair of the methods
round(apply(cbind(sir.cov, sir.save, sir.phd, save.cov, save.phd, phd.cov), 2,mean),3)
 sir.cov sir.save  sir.phd save.cov save.phd phd.cov
   0.966    0.208    0.257    0.214    0.348   0.288

## the averages of the absolute correlations by each method and x1
round(apply(cbind(sir.eta, cov.eta, save.eta, phd.eta), 2, mean), 3)
 sir.eta  cov.eta save.eta  phd.eta
   0.941    0.939    0.215    0.267
```

The highest average of the six pairwise absolute correlations is highest between SIR and cov$k$; in addition, either of SIR or cov$k$ estimates x_1 well. If the regression model is changed to $y|\mathbf{X} = x_1^2 + \varepsilon$, the pair of SAVE and pHd yield the highest averages in the absolute correlations among the six pairs, and both SAVE and pHd are good SDR methods for the regression.

Let us think about this in a reverse manner. Suppose that the estimates of sufficient predictors by a pair of SAVE and pHd are more highly correlated than those of any other pairs of the SDR methods. Then, it would be not bad reasoning to believe that SAVE or pHd should be preferable to the data, although we do not know what the true model is. That is, the pair of the two methods that gives the highest correlation would be not a bad choice for estimating $\mathcal{S}_{y|\mathbf{X}}$, although it is not guaranteed to be the best. We will formalize this idea.

First, we list all six possible pairs of the four methods in the **dr**-package: (1) (SIR, SAVE); (2) (SIR, pHd); (3) (SIR, cov$k$); (4) (SAVE, pHd); (5) (SAVE, cov$k$); (6) (pHd, cov$k$). Let $\hat{\boldsymbol{\eta}}_a$ and $\hat{\boldsymbol{\eta}}_b$ be orthonormal basis estimates of $\mathcal{S}_{y|\mathbf{Z}}$ by any pair among the six under $d = m$. If $d = 1$, the correlation between $\hat{\boldsymbol{\eta}}_a^{\mathrm{T}}\mathbf{Z}$ and $\hat{\boldsymbol{\eta}}_b^{\mathrm{T}}\mathbf{Z}$ can be simply computed using the usual Pearson correlation coefficient. However, if $d \geq 2$, the correlation between $\hat{\boldsymbol{\eta}}_a^{\mathrm{T}}\mathbf{Z}$ and $\hat{\boldsymbol{\eta}}_b^{\mathrm{T}}\mathbf{Z}$ is not straightforward. Now it should be noted that the correlation between $\hat{\boldsymbol{\eta}}_a^{\mathrm{T}}\mathbf{Z}$ and $\hat{\boldsymbol{\eta}}_b^{\mathrm{T}}\mathbf{Z}$ depends on the similarity between $\hat{\boldsymbol{\eta}}_a$ and $\hat{\boldsymbol{\eta}}_b$ regardless of the value of d. The similarity of two matrices with the same column rank is equivalent to the distance between their column subspaces. Finally, a correlation between of $\hat{\eta}_a^{\mathrm{T}}\mathbf{Z}$ and $\hat{\eta}_b^{\mathrm{T}}\mathbf{Z}$ can be alternatively measured by a trace correlation r_{tr} (Hooper, 1959) between $\mathcal{S}(\hat{\eta}_a)$ and $\mathcal{S}(\hat{\eta}_b)$:

$$r_{\mathrm{tr}} = \sqrt{\frac{1}{m} trace\{\hat{\eta}_a\hat{\eta}_a^{\mathrm{T}}\hat{\eta}_b\hat{\eta}_b^{\mathrm{T}}\}}.$$

The trace correlation r_{tr} varies between 0 and 1, and higher values indicate that $\mathcal{S}(\hat{\eta}_a)$ and $\mathcal{S}(\hat{\eta}_b)$ are closer. If the two subspaces coincide, then we have $r_{\mathrm{tr}} = 1$.

The trace correlation is expected to be maximized under the true dimension of d, which is usually unknown. For a smaller choice of d, the true basis are underestimated, so each method will not normally estimate the same parts of $\mathcal{S}_{y|\mathbf{X}}$. This implies that r_{tr} will be smaller compared to it under the true dimension. In contrast, in the case of a larger selection of d, the true basis are overestimated. Then, there is redundancy in the estimates, and the redundancy will be random for each method. This indicates that smaller correlations should be expected. This discussion is confirmed through the following simulation example.

```
sir.cov <- sir.save <- sir.phd <- save.cov <- save.phd <- phd.cov <- NULL
sir.cov2 <- sir.save2 <- sir.phd2 <- save.cov2 <- save.phd2 <- phd.cov2 <- NULL
sir.cov3 <- sir.save3 <- sir.phd3 <- save.cov3 <- save.phd3 <- phd.cov3 <- NULL
sir.eta2 <- save.eta2 <- phd.eta2 <- cov.eta2 <- NULL

set.seed(1)

## true basis matrix of the central subspace
eta <- cbind(c(1, rep(0,9)), c(0, 1, rep(0,8)) )

## starting loop
for (i in 1:500){

## model construction
  X <- matrix(rnorm(100*10), c(100,10)); y <- X[,1] + X[,1]*X[,2]+ rnorm(100)
  w <- c(scale(y,center=TRUE, scale=TRUE)); w2 <- cbind(w, w^2); w3<- cbind(w2, w^3)

## obtaining basis estimates from the four methods
  sir5b <- dr(y~X, method="sir", nslice=5)$raw.evectors
  cov2b <- dr(w2~X, method="ols")$raw.evectors
  cov3b <- dr(w3~X, method="ols")$raw.evectors
  save5b <- dr(y~X, method="save", nslice=5)$raw.evectors
  phdb <- dr(y~X, method="phdres")$raw.evectors

## computing trace correlations under d=1 for the six pairs
  sir.cov[i] <- tr.cor(sir5b[,1], cov2b[,1], 1)
  sir.save[i] <- tr.cor(sir5b[,1], save5b[,1], 1)
  sir.phd[i] <- tr.cor(sir5b[,1], phdb[,1], 1)
  save.cov[i] <- tr.cor(save5b[,1], cov2b[,1], 1)
  save.phd[i] <- tr.cor(save5b[,1], phdb[,1], 1)
  phd.cov[i] <- tr.cor(phdb[,1], cov2b[,1], 1)

## computing trace correlations under d=2 for the six pairs
  sir.cov2[i] <- tr.cor(sir5b[,1:2], cov2b[,1:2], 2)
  sir.save2[i] <- tr.cor(sir5b[,1:2], save5b[,1:2], 2)
  sir.phd2[i] <- tr.cor(sir5b[,1:2], phdb[,1:2], 2)
  save.cov2[i] <- tr.cor(save5b[,1:2], cov2b[,1:2], 2)
  save.phd2[i] <- tr.cor(save5b[,1:2], phdb[,1:2], 2)
  phd.cov2[i] <- tr.cor(phdb[,1:2], cov2b[,1:2], 2)
  sir.cov3[i] <- tr.cor(sir5b[,1:3], cov3b[,1:3], 3)

## computing trace correlations under d=3 for the six pairs
  sir.save3[i] <- tr.cor(sir5b[,1:3], save5b[,1:3], 3)
  sir.phd3[i] <- tr.cor(sir5b[,1:3], phdb[,1:3], 3)
  save.cov3[i] <- tr.cor(save5b[,1:3], cov3b[,1:3], 3)
  save.phd3[i] <- tr.cor(save5b[,1:3], phdb[,1:3], 3)
  phd.cov3[i] <- tr.cor(phdb[,1:3], cov3b[,1:3], 3)
 }
```

```
## averages of the trace correlations for each pair under d=1,2,3
round(apply(cbind(sir.cov, sir.save, sir.phd, save.cov, save.phd, phd.cov),
      2, mean), 3)
round(apply(cbind(sir.cov2, sir.save2, sir.phd2, save.cov2, save.phd2, phd.cov2),
      2, mean), 3)
round(apply(cbind(sir.cov3, sir.save3, sir.phd3, save.cov3, save.phd3, phd.cov3),
      2, mean), 3)

 sir.cov  sir.save   sir.phd  save.cov  save.phd   phd.cov
   0.580     0.224     0.476     0.317     0.451     0.616
 sir.cov2 sir.save2  sir.phd2 save.cov2 save.phd2  phd.cov2
   0.822     0.397     0.656     0.486     0.606     0.801
 sir.cov3 sir.save3  sir.phd3 save.cov3 save.phd3  phd.cov3
   0.773     0.517     0.665     0.577     0.679     0.753
```

In the model, the true structural dimension is equal to two, and $\mathcal{S}_{y|\mathbf{Z}}$ is spanned by the columns of $\boldsymbol{\eta} = \{(1,0,\ldots,0),(0,1,0,\ldots,0)\}^{\mathrm{T}}$. For $d = 1,2,3$, the averages of the trace correlations are computed for the six pairs. According to the results, the averages are maximized under the true structural dimension $d = 2$ for each pair, as expected. In addition, the maximum trace correlation is attained at the pair of SIR and cov2 under $d = 2$.

Based on this discussion, the selection algorithm can be developed as follows:

Algorithm

1. Fix the maximum value $d_{\max}$ of d. The value should be less than or equal to $\min\{k,(h_{\mathrm{SIR}}-1),(p-1)\}$, where k stands for the maximum polynomial in cov$k$ and h_{SIR} is the number slices in SIR.
2. Compute r_{tr} between the basis estimates from each pair for $m = 1,\ldots,d_{\max}$. For $d = 1$ or 2, the cov2 is commonly used, and the cov$d$ is employed for $d \geq 3$.
3. Choose a pair of the methods to provide the maximum r_{tr} in Step 2. The pair chosen in this step will be called the *initial pair*.
4. Remove the initial pair and all of the other pairs not containing one of the methods of the initial pair for all values of d. After completing this step, the surviving pairs must contain one method of the initial pair.
5. Search a second pair that has the highest r_{tr} among all of the remaining pairs. This pair will be called the *final pair*.
6. The common method in the initial and final pairs is the representative SDR method to the data.

Steps 4–5 are required to select one of the methods of the initial pair in Step 2. This approach of selecting SDR methods through the proposed algorithm is called *basis-adaptive selection* (BAS). The BAS is data-driven and not necessarily limited in the four SDR methods of SIR, SAVE, pHd and cov$k$, as one can extend it to other SDR methods.

The bas1 function

The `bas1` function runs the BAS algorithm for SIR, SAVE, pHd and cov$k$. The function requires **dr** and has the following arguments:

```
bas1(formula, data, nsir=5, nsave=4, k=4, plot=TRUE)
```

The arguments of `nsir`, `nsave` and `k` determine the numbers of slices for SIR and SAVE as well as the moment for cov$k$, respectively. The default values are 5, 4 and 4, in order. If `plot=TRUE`, the function `bas1` returns a scatter plot of the trace correlations against the various choices of d, up to min(`nsir`-1, `k`) for the six pairs of the four methods in **dr**.

The values returned by the `bas1` function are `selection`, `sir`, `save`, `phd` and `covk`. Their descriptions are as follows:

- `selection`: a selected method among SIR, SAVE, pHd and cov$k$ by the BAS algorithm
- `sir`: the SIR application object with the number of slices equal to `nsir`
- `save`: the SAVE application object with the number of slices equal to `nsave`
- `pHd`: the pHd application object
- `covk`: a list type object by the cov$k$ application objects with the kth polynomial

Soil evaporation data: Revisited

We revisit the soil evaporation data. Here, the application of the BAS is more extensively studied under the cross-combinations of four or five numbers of slices for SIR and SAVE and the third or forth order of polynomials for cov$k$. We define "BASijk" for $i = 3, 4, 5$, $j = 4, 5$ and $k = 2, \ldots, i$. For example, in BAS453, the numbers of slice for SIR and SAVE are four and five, respectively, and the order of polynomial for cov$k$ is three.

```
## BAS application
BAS342 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=3, nsave=4, k=2, plot=FALSE)
BAS343 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=3, nsave=4, k=3, plot=FALSE)
BAS352 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=3, nsave=5, k=2, plot=FALSE)
BAS353 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=3, nsave=5, k=3, plot=FALSE)

BAS443 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=4, nsave=4, k=3, plot=FALSE)
BAS444 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=4, nsave=4, k=4, plot=FALSE)
BAS453 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=4, nsave=5, k=3, plot=FALSE)
BAS454 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=4, nsave=5, k=4, plot=FALSE)

BAS544 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=5, nsave=4, k=4, plot=FALSE)
BAS545 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=5, nsave=4, k=5, plot=FALSE)
BAS554 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=5, nsave=5, k=4, plot=FALSE)
BAS555 <- bas1(Evap~Avat+Rat+Avst+Rst, data=evaporat, nsir=5, nsave=5, k=5, plot=FALSE)

## Selection results
BAS342$selection; BAS343$selection; BAS352$selection; BAS353$selection
BAS443$selection; BAS444$selection; BAS453$selection; BAS454$selection
BAS544$selection; BAS545$selection; BAS554$selection; BAS555$selection

## dimension test
set.seed(100); round(dr.permutation.test(BAS342$covk$cov2, npermute=1000)$summary, 3)
set.seed(100); round(dr.permutation.test(BAS343$covk$cov3, npermute=1000)$summary, 3)
round(dr.test(BAS443$phd, numdir=4), 3)
round(dr.test(BAS544$sir), 3)
```

Both BAS342 and BAS352 recommended cov2, and cov3 was the selection of BAS343 and BAS353. For the cases of BAS4jk and BAS5jk, pHd and SIR were recommended regardless of the values of j and k. These selection results are summarized in Table 2. From Table 2, it can be seen that the selection results are different from the numbers of slices in SIR. For example, if the numbers of slices in SIR was 3, BAS recommended cov$k$. For $i = 4$ and 5, BAS chose pHd and SIR, respectively. Thus, additional work needs to be done in order to choose between cov$k$, pHd and SIR in the soil evaporation data.

For this purpose, we considered how reasonably the methods recommended in Table 2 estimated the true dimension of the central subspace. The dimension estimation results are already summarized in Table 1, and the nominal level 5% was used. First, the two methods of cov2 and cov3 were inspected. According to Table 1, cov2 and cov3 determine that $d \geq 2$ and $d \geq 3$, respectively. This indicates that the order of polynomial in cov$k$ should be bigger than or equal to 4 for further dimension determination. However, in the case $k = 4$, cov$k$ yields $\hat{d} \geq 4$, so the dimension reduction is meaningless because $p = 4$. Therefore, we conclude that cov$k$ would not be recommended one for this data. Next we consider pHd, which infers that $\hat{d} = 0$ according to Table 1. Therefore, the pHd should also be ruled out for the final choice. In Table 1, the SIR with 5 slices determines that $\hat{d} = 2$, which is the most reasonable among the three recommendations. Thus, one may continue the regression analysis with the two-dimensional sufficient predictors from the SIR results.

Table 2: Method selection results by BAS in soil evaporate data

	BAS3•2	BAS3•3	BAS4••	BAS5••
Recommendation	cov2	cov3	pHd	SIR

Numerical studies

Predictors $\mathbf{X} = (x_1, \ldots, x_{10})^{\mathrm{T}}$ and a random error ε were independently generated from $N(0,1)$. Under these variable configurations, the following four models were considered:

- Model 1: $y = x_1 + \varepsilon$
- Model 2: $y = x_1^2 + \varepsilon$
- Model 3: $y = x_1 + x_1 x_2 + \varepsilon$
- Model 4: $y = 1 + x_1 + \exp(x_2)\varepsilon$

In Models 1 and 2, the column of $\boldsymbol{\eta} = (1, 0, \ldots, 0)^{\mathrm{T}}$ spans $\mathcal{S}_{y|\mathbf{X}}$, so $d = 1$. The structural dimension of Models 3 and 4 is two, and $\mathcal{S}_{y|\mathbf{X}}$ is spanned by the columns of $\boldsymbol{\eta} = \{(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0)\}^{\mathrm{T}}$. The four models are commonly used in Cook and Weisberg (1991); Li (1991, 1992); Yin and Cook (2002). The desired SDR methods for each model are as follows: Model 1: SIR and cov*k*; Model 2: SAVE and pHd; Model 3: cov*k* and pHd; Model 4: SIR and cov*k*.

Each model was iterated 500 times with $n = 100$ and $n = 200$. Throughout all numerical studies, five slices were commonly used for SIR and SAVE, and four was the maximum polynomial in cov*k*.

As a summary, the selection percentages of the methods by BAS are reported in Table 3.

Table 3: Percentages of the selection of a method among SIR, SAVE, pHd, and cov*k* by BAS

	$n = 100$				$n = 200$			
	SIR	SAVE	pHd	cov*k*	SIR	SAVE	pHd	cov*k*
Model 1	41.2	0.6	0.2	58.0	49.4	0.0	0.0	50.6
Model 2	1.0	17.8	78.4	2.8	0.0	22.8	77.2	0.0
Model 3	6.4	2.0	25.6	66.0	3.6	1.2	25.0	70.2
Model 4	6.4	0.6	6.4	86.6	26.8	1.2	1.4	70.6

According to Table 3, the BAS selects SIR and cov*k* 99.8% of the time for Model 1 and SAVE and pHd 97.8% of the time for Model 2 with $n = 100$, respectively. These results are consistent with the discussion given in the previous section regarding the development of the algorithm. In Model 1, cov*k* is preferred to SIR with $n = 100$, but the two are almost equally selected with $n = 200$. For Model 2, the pHd is recommended more than SAVE with $n = 100$ and $n = 200$. In Model 3, the pHd and the cov*k* are two dominant methods according to BAS, although the cov*k* is selected more frequently with $n = 100$. For Model 4, the cov*k* is recommended most frequently, but the selection percentages of SIR rapidly grow with $n = 200$.

```
set.seed(5); sel1 <- sel2 <- sel3 <- sel4 <- sel12 <- sel22 <- sel32 <- sel42 <- NULL

## starting loop
for (i in 1:500){

## model construction for n=100
  X <- matrix(rnorm(100 * 10), c(100, 10))
  y1 <- X[,1] + rnorm(100)
  y2 <- X[,1]^2 + rnorm(100)
  y3 <- X[,1] + X[,1] * X[,2] + rnorm(100)
  y4 <- 1 + X[,1] + exp(X[,2]) * rnorm(100)

## model construction for n=200
  X2 <- matrix(rnorm(200 * 10), c(200,10))
  y12 <- X2[,1] + rnorm(200)
  y22 <- X2[,1]^2 + rnorm(200)
  y32 <- X2[,1] + X2[,1] * X2[,2] + rnorm(200)
  y42 <- 1 + X2[,1] + exp(X2[,2]) * rnorm(200)

## selection by BAS
  sel1[i] <- bas1(y1~X, plot=FALSE)$selection
  sel2[i]<-bas1(y2~X, plot=FALSE)$selection
  sel3[i] <- bas1(y3~X, plot=FALSE)$selection
  sel4[i]<-bas1(y4~X, plot=FALSE)$selection
  sel12[i] <- bas1(y12~X2, plot=FALSE)$selection
  sel22[i]<-bas1(y22~X2, plot=FALSE)$selection
  sel32[i] <- bas1(y32~X2, plot=FALSE)$selection
  sel42[i]<-bas1(y42~X2, plot=FALSE)$selection
}

## computing the selection percentages for model 1
# n=100
c(length(which(sel1=="sir")), length(which(sel1=="save")),
  length(which(sel1=="phd")), length(which(sel1=="covk"))) / 500
# n=200
c(length(which(sel12=="sir")), length(which(sel12=="save")),
  length(which(sel12=="phd")), length(which(sel12=="covk"))) / 500

## computing the selection percentages for model 2
# n=100
c(length(which(sel2=="sir")), length(which(sel2=="save")),
  length(which(sel2=="phd")), length(which(sel2=="covk"))) / 500
# n=200
c(length(which(sel22=="sir")), length(which(sel22=="save")),
  length(which(sel22=="phd")), length(which(sel22=="covk"))) / 500

## computing the selection percentages for model 3
# n=100
c(length(which(sel3=="sir")), length(which(sel3=="save")),
  length(which(sel3=="phd")), length(which(sel3=="covk"))) / 500
# n=200
c(length(which(sel32=="sir")), length(which(sel32=="save")),
  length(which(sel32=="phd")), length(which(sel32=="covk"))) / 500

## computing the selection percentages for model 4
# n=100
c(length(which(sel4=="sir")), length(which(sel4=="save")),
  length(which(sel4=="phd")), length(which(sel4=="covk"))) / 500
# n=200
c(length(which(sel42=="sir")), length(which(sel42=="save")),
  length(which(sel42=="phd")), length(which(sel42=="covk"))) / 500
```

Summary

Sufficient dimension reduction (SDR) is a useful dimension reduction method in regression. The popularly used SDR methods among the others include sliced inverse regression (Li, 1991), sliced average variance estimation (Cook and Weisberg, 1991), principal Hessian directions (Li, 1992) and covariance method (Yin and Cook, 2002). Currently, the **dr**-package is the only one to cover the four sufficient dimension reduction methods in R. However, users were left without practical guidelines as to which SDR method should be chosen. To remedy this, we developed the basis-adaptive selection (BAS) algorithm to recommend a SDR method in **dr** by maximizing a trace correlation (Hooper, 1959). A real data example and numerical studies confirm its potential usefulness in practice.

The BAS algorithm requires the two parts. The first is the basis estimates of the dimension reduction subspace, and the second is a quantity to measure the distances between the subspaces spanned by the columns of the estimates. For non-linear feature extractions through different kernel methods, there is no reason why the BAS algorithm cannot be applied if some quantity to measure the distance between the non-linear subspaces defined in different kernels is feasible.

If the sliced inverse regression applications with various numbers of slices replaces the other three methods in the BAS, the BAS algorithm can be utilized to find a good number of slices for this method.

The code for BAS is available through the personal webpage of the author: `http://home.ewha.ac.kr/~yjkstat/bas.R`.

Acknowledgments

For the corresponding author Jae Keun Yoo, this work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Korean Ministry of Education (NRF-2017R1A2B1004909).

Bibliography

R. D. Cook. *Regression Graphics: Idea for Studying Regressions through Graphics*. John Wiley & Sons, 1998a. [p124]

R. D. Cook. Principal hessian directions revisited. *Journal of the American Statistical Association*, 93(441): 84–94, 1998b. URL http://dx.doi.org/10.1080/01621459.1998.10474090. [p124]

R. D. Cook and S. Weisberg. Comment: Sliced inverse regression for dimension reduction. *Journal of the American Statistical Association*, 86(414):328–332, 1991. URL http://dx.doi.org/10.1080/01621459.1991.10475036. [p124, 130, 131]

J. Hooper. Simultaneous equations and canonical correlation theory. *Econometrika*, 27(2):245–256, 1959. URL http://dx.doi.org/10.2307/1909445. [p125, 127, 131]

K. C. Li. Sliced inverse regression for dimension reduction. *Journal of the American Statistical Association*, 86(414):326–342, 1991. URL http://doi.org/10.1080/01621459.1991.10475035. [p124, 130, 131]

K. C. Li. On principal hessian directions for data visualization and dimension reduction: Another application of stein's lemma. *Journal of the American Statistical Association*, 87(420):1025–1039, 1992. URL http://dx.doi.org/10.1080/01621459.1992.10476258. [p124, 130, 131]

S. Weisberg. Dimension reduction regression in R. *Journal of Statistical Software*, 7(1):1–22, 2002. URL http://dx.doi.org/10.18637/jss.v007.i01. [p124]

X. Yin and R. D. Cook. Dimension reduction for the conditional kth moment in regression. *Journal of Royal Statistical Society Series B*, 64(2):159–175, 2002. URL http://dx.doi.org/10.1111/1467-9868.00330. [p124, 125, 130, 131]

Jae Keun Yoo
Department of Statistics, Ewha Womans University
Seoul 03760
Republic of Korea
peter.yoo@ewha.ac.kr

SARIMA Analysis and Automated Model Reports with BETS, an R Package

by Talitha F. Speranza, Pedro C. Ferreira, and Jonatha A. da Costa

Abstract This article aims to demonstrate how the powerful features of the R package **BETS** can be applied to SARIMA time series analysis. **BETS** provides not only thousands of Brazilian economic time series from different institutions, but also a range of analytical tools, and educational resources. In particular, **BETS** is capable of generating automated model reports for any given time series. These reports rely on a single function call and are able to build three types of models (SARIMA being one of them). The functions need few inputs and output rich content. The output varies according to the inputs and usually consists of a summary of the series properties, step-by-step explanations on how the model was developed, predictions made by the model, and a file containing these predictions. This work focuses on this feature and several other **BETS** functions that are designed to help in modeling time series. We present them in a thorough case study: the SARIMA approach to model and forecast the Brazilian production of intermediate goods index series.

Introduction

The **BETS** (Ferreira et al., 2017) package (an abbreviation for Brazilian Economic Time Series) for R (R Core Team, 2017) allows easy access to the most important Brazilian economic time series and a range of powerful tools for analyzing and visualizing not only these data, but also external series. It provides a much-needed single access point to the many Brazilian series through a simple, flexible and robust interface. The package now contains more than 18,000 Brazilian economic series and contains an integrated modelling and learning environment. In addition, some functions include the option of generating explanatory outputs that encourage and help users to expand their knowledge.

BETS relies on several well-established R packages. In order to work with time series, **BETS** functions build upon packages such as **forecast** (Hyndman, 2015), **mFilter** (Balcilar, 2016), **urca** (Pfaff et al., 2016) and **seasonal** (Sax, 2016). Access to external APIs to obtain data is handled by the web scraping and HTTP tools available through the **httr** (Wickham, 2017) and **rvest** (Wickham, 2016) packages, whereas **BETS** own databases are queried using the package **RMySQL** (Ooms et al., 2016).

This article seeks to demonstrate many of these features through a practical case. We create a SARIMA model (Box and Jenkins, 1970) for the Brazilian production of intermediate goods index (BPIGI) series, step by step, using mostly functions from **BETS**. Then, we show that **BETS** is capable of using **rmarkdown** (Allaire, 2017) to generate a fully automated report that includes a virtually identical SARIMA analysis. The report needs few inputs, runs for any given time series and has the advantage of outputting several informative comments, explaining to the user how and why it achieved the given results. The outputs, of course, vary according to the inputs. Reports provide a link to a file containing the series under analysis and the predictions calculated by the automated model.

Automated reports can also produce GRNN[1] and exponential smoothing models, but here we focus on the previously specified methodology[2]. In the next section, we briefly present the SARIMA approach and show how to use **BETS** functions to conduct such an analysis in a typical case: predicting the future levels of the BPIGI. Then, in Section 3, we introduce the automated model reports and illustrate its capabilities by applying it to the series under study using a range of other inputs. Section 4 concludes this article.

SARIMA Analysis: A Case Study

In this section we develop a case study in which we use a handful of **BETS** functions to model and forecast values of the Brazilian production of intemediate goods index, following the SARIMA (Box & Jenkins) approach. Before starting the analysis we will briefly look at the methodology to be used. For a more comprehensive treatment of the topic, see Box and Jenkins (1970) and Ferreira et al. (2016)[3].

[1]General Regression Neural Network, as proposed by Specht (1991)

[2]Examples on how to use GRNN and exponential smoothing reports are available in the documentation for the function `report()`.

[3]For a hands-on introduction, see the tutorial of Arima models offered by Duke university at `https://people.duke.edu/~rnau/411arim.htm`

Box & Jenkins Methodology

The Box & Jenkins methodology allows the prediction of future values of a series based only on past and present values of the same series. These univariate models are known as SARIMA, an abbreviation for Seasonal Autoregressive Integrated Moving Average, and have the following form:

$$\Phi_P(B)\phi_p(B)\nabla^d\nabla^D Z_t = \Theta_Q(B)\theta_q(B)a_t, \tag{1}$$

where

- B is the lag operator, i.e., for all $t > 1$, $BZ_t = Z_{t-1}$
- P, Q, p and q are the orders of the polynomials
- $\Phi_P(B)$ is the seasonal autoregressive polynomial
- $\phi_p(B)$ is the autoregressive polynomial
- $\nabla^d = (1-B)^d$ is the difference operator and d is the number of unit roots
- $\nabla^D = (1-B^s)^D$ is the difference operator at the seasonal frequency s and D is the number of seasonal unit roots
- Z_t is the series being studied
- $\Theta_Q(B)$ is the seasonal moving average polynomial
- $\theta_q(B)$ is the moving average polynomial
- a_t is the random error.

In its original form, which will be used here, the Box & Jenkins methodology consists of three iterative stages:

(i) Identification and selection of the models: the series is checked for stationarity and the appropriate corrections are made for non-stationary cases, by differencing the series `d` times in its own frequency and `D` times in the seasonal frequency. The orders of the polynomials ϕ, Φ, θ, and Θ are identified using the autocorrelation function (ACF) and partial autocorrelation function (PACF) of the series.

(ii) Estimation of the model's parameters using maximum likelihood or dynamic regression.

(iii) Confirmation of model compliance using statistical tests.

The ACF, used in stage (i), can be defined as the correlation of a series with a lagged copy of itself as a function of lags. Formally, we can write the ACF of a time series s with realizations s_i for $i \in [1, N]$ and mean $\bar{s}$ as

$$ACF(s,l) = \frac{\sum_{i=1}^{N-l}(s_i - \bar{s})(s_{i+l} - \bar{s})}{\sum_{i=1}^{N}(s_i - \bar{s})^2}. \tag{2}$$

Another tool for phase (i), the PACF also gives the correlation of a series with its own lagged values, but controlling for all lower lags. Let $P_{t,l}(x)$ denote the projection of x_t onto the space spanned by $x_{t+1}, \ldots, x_{t+l-1}$. We can define the PACF as

$$PACF(s,l) = \begin{cases} Cor(s_i, s_{i+1}) \text{ if } l = 1, \\ Cor(s_{t+l} - P_{t,l}(s_{t+l}), \ldots, s_t - P_{t,l}(s_t)) \text{ if } l \geq 1. \end{cases} \tag{3}$$

The PACF is used to find `p` (the lag beyond which the PACF becomes zero is the indicated number of autoregressive terms), while the ACF is used to find `q` (the cut in the ACF points to the number of moving average terms). Seasonal polynomial orders, `P` and `Q`, are identified in a similar fashion, but by inspecting high values at fixed intervals. In this case, the intervals become the orders.

If the model does not pass phase (iii), the procedure starts again at step (i). Otherwise, the model is ready to be used to make forecasts. In the next section we will look at each of these stages in more detail and say more about the methodology as we go through the case study.

Some preliminary topics

The first step is to find the series in the **BETS** database. This can be done with the `BETSsearch()` function. It performs flexible searches on the metadata table that stores series characteristics. Since metadata is accessed through SQL queries, searchers are fast and the package performs well in any environment. `BETSsearch()` is a crucial feature of **BETS**, so we provide a thorough description in this section.

Accepted parameters, expected data types, and definitions are the following:

- `description` - A `character`. A search *string* to look for matching series descriptions.
- `src` - A `character`. The source of the data.
- `periodicity` - A `character`. The frequency with which the series is observed.
- `unit` - A `character`. The unit in which the data were measured.
- `code` - An `integer`. The unique code for the series in the **BETS** database.
- `view` - A `boolean`. By default, `TRUE`. If `FALSE`, the results will be shown directly on the R console.
- `lang` - A `character`. Search language. By default, 'pt', for Portuguese. A search can also be performed in English by changing the value to 'en'.

To refine the search, there are syntax rules for the parameter `description`:

1. To look for alternative words, separate them by blank spaces. Example: `description = 'core ipca'` means that the description of the series should contain the words *core* **and** *ipca*.
2. To search for complete expressions, put them inside single quotes (' '). Example: `description = "index and 'core ipca'"` means that the description of the series should contain the expression *core ipca* **and** the word *index*.
3. To exclude words from the search, insert a ∼ before each word. Example: `description = "ipca ∼ core"` means that the description of the series should contain the word *ipca* and should **not** contain the word *core*.
4. To exclude a whole expression from a search, place them inside single quotes (' ') and insert a ∼ before each of them. Example: `description = "index ∼ 'core ipca'"` means that the description of the series should contain *index* and should **not** contain *core ipca*.
5. It is possible to search for or exclude certain words as long as these rules are obeyed.
6. No blank space is required after the exclusion sign (∼), but it is required after each expression or word.

Finally, the command to find information about the series, which we will use throughout this article, is shown below, along with the corresponding output.

```
# Searching for the production of intermediate goods index series
# excluding the term 'imports' from the search

results <- BETSsearch(description = "'intermediate goods' ~ imports", view = F, lang = "en")
results

#> # A tibble: 3 x 7
#>   code  description      unit  periodicity start
#>   <chr> <chr>            <chr> <chr>       <chr>
#> 1 11068 Intermediate go? Index M           31/0?
#> 2 1334  Production indi? Index M           31/0?
#> 3 21864 Physical Produc? Index M           01/0?
#> # ... with 2 more variables: last_value <chr>,
#>   source <chr>
```

The series we are looking for is the third (code 21864). We now load it using the function `BETSget()` and store some values for later comparison with the model's forecasts. We will also produce a graph (Figure 1) to help formulate hypotheses about the behavior of the underlying stochastic process. The professional looking chart is created with `chart()`, which has the nice property of displaying useful information about the series, such as the last value and monthly/interannual variations.

```
# Get the series with code 21864 (BPIGI, IBGE)
data <- BETSget(21864)

# Keep last values for comparison with the forecasts
data_test <- window(data, start = c(2015,11), end = c(2016,4), frequency = 12)
data <- window(data, start = c(2002,1), end = c(2015,10), frequency = 12)

# Graph of the series
```

```
params <- list(
    title = "Intermediate Goods Production",
    subtitle = "Index",
    colors = "royalblue")
chart(ts = data, style = "normal", open = T, params = params, file = "igp.png")
```

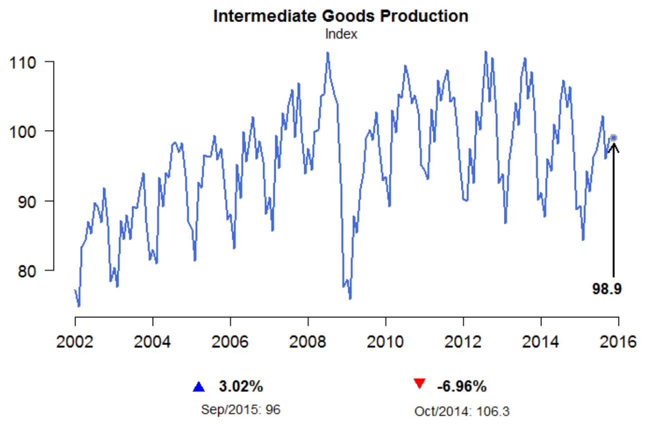


Figure 1: Graph of the BPIGI, made with function `chart()`. Indicators below the x coordinates represent index variation over a month and over 12 months, respectively.

Four characteristics can be observed. First, the series is homoskedastic, with monthly seasonality. A further striking feature is the structural break in November 2008, when the international financial crisis occurred and investor confidence plummeted. This break had a direct impact on the next important characteristic of the series, the trend, which was initially clearly one of growth, albeit not at explosive rates. From November 2008 onward, however, production appeared to remain unchanged or even to fall. Since the main goal of this article is to present **BETS**′ features, rather than performing an overcautious time series analysis, we decided to work with fewer observations, thus avoiding problems with the structural break. Only the values from January 2009 on were used. This did not imply poorer results. In fact, discarding older values yields better predictions, at least within the SARIMA framework.

```
data <- window(data, start = c(2009,1), end = c(2015,10), frequency = 12)
```

In the following sections we create a model for the chosen series following the previously defined modelling steps.

I. Identification

Stationarity Tests

This subsection covers a crucial step in the Box & Jenkins approach: checking if there are unit roots in the seasonal and non-seasonal autoregressive model polynomials and determining how many of these roots there are. If there are no unit roots, the series is stationary. In case we find unit roots, it is possible to obtain stationarity by differencing the original series. Then, the order of the polynomials can be identified using the ACF and the PACF funcitons.

The `ur_test()` function performs unit root tests. The user can choose among *Augmented Dickey Fuller* (ADF, Dickey and Fuller (1979)), *Phillips-Perron* (PP, Phillips and Perron (1988)), and *KPSS* (Kwiatkowski et al., 1992) tests. The `ur_test()` function was built around functions from the **urca** package. Relative to those, the advantage of `ur_test()` is the output, which is designed to quickly see the test's results and all the needed information. The output is an object with two fields: (i) a table showing test statistics, critical values and whether the null hypothesis is rejected or not, and (ii) a vector containing the residuals of the test's equation. This equation is shown below.

$$\Delta y_t = \phi_1 + \tau_1 t + \tau_2 y_{t-1} + \delta_1 \Delta y_{t-1} + \cdots + \delta_{p-1} \Delta y_{t-p+1} + \varepsilon_t \tag{4}$$

The test statistics in the output table refer to the coefficients ϕ (drift), τ_1 (deterministic trend) and τ_2 (unit root). Inclusion of the constant (drift) and deterministic trend is optional. To control the test parameters, ur_test() accepts the same parameters as urca's ur.df(), as well as the desired significance level.

```
df <- ur_test(y = data, type = "drift", lags = 12,
              selectlags = "BIC", level = "5pct")
# Show the result of the tests
df$results

##       statistic crit.val rej.H0
## tau2 -0.9673939    -2.89     no
## phi1  0.7526983     4.71    yes
```

Therefore, for the series in levels, the null hypothesis that there is a unit root cannot be rejected at a 95% confidence level, as the test statistic tau2 is greater than the critical value. We will now apply the diff function to the series repeatedly, to determine whether the differenced series has a unit root. This way we find out how many unit roots there are.

```
ns_roots <- 0
d_ts <- data

# Dickey-Fuller tests loop
# Execution is interrupted when the null hypothesis cannot be rejected
while(df$results[1,"statistic"] > df$results[1,"crit.val"]) {
    ns_roots <- ns_roots + 1
    d_ts <- diff(d_ts)
    df <- ur_test(y = d_ts, type = "none", lags = 12,
                  selectlags = "BIC", level = "5pct")
 }
ns_roots
#> [1] 1
```

Hence, for the series in first differences, the hypothesis that there is a unit root is rejected at the 5% significance level. We now use the nsdiffs function from the **forecast** package to perform the Osborn-Chui-Smith-Birchenhall test (Osborn et al., 1988) and identify unit roots at the seasonal frequency (in our case, monthly).

```
library(forecast)

# OCSB tests for unit roots at the seasonal frequency
nsdiffs(data, test = "ocsb")

#> [1] 1
```

The results indicate that there is a seasonal unit root and that seasonal differencing is therefore required:

```
d.data <- diff(diff(data), lag = 12, differences = 1)
```

Autocorrelation Functions

The previous conclusions are corroborated by the autocorrelation function of the series in differences (from now on, $\nabla\nabla^{12}Z_t$). It shows that statistically significant correlations, i.e., correlations outside the confidence interval, are not persistent for lags corresponding to multiples of 12 or values close to these multiples. This indicates the absence of a seasonal unit root.

The **BETS** function that we use to draw the correlograms is corrgram(). Unlike its main alternative, the **forecast** package's Acf() function, corrgram() returns an attractive graph and provides the option of calculating the confidence intervals using the method proposed by Bartlett (Bartlett, 1946). The greatest advantage it offers, however, cannot be shown here as it depends on Javascript. If the style parameter is set to 'plotly', an interactive graph is produced, displaying all the values of interest (autocorrelations, lags and confidence intervals) on mouse hover, as well as providing zooming, panning, and the option to save the graph in *png* format.

```
# Correlogram of differenced series
corrgram(d.data, lag.max = 48, mode = "bartlett", style="normal")
```

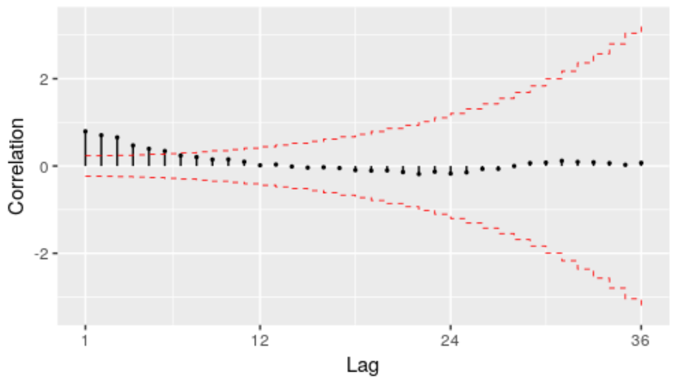


Figure 2: Autocorrelation function of $\nabla\nabla^{12}Z_t$

This correlogram, however, is not sufficient for us to propose a model for the series. We will therefore produce a graph of the partial autocorrelation function (PACF) of $\nabla\nabla^{12}Z_t$. `corrgram()` can also be used for this purpose.

```
# Partial autocorrelation function of diff(data)
corrgram(d.data, lag.max = 48, type = "partial", style="normal")
```

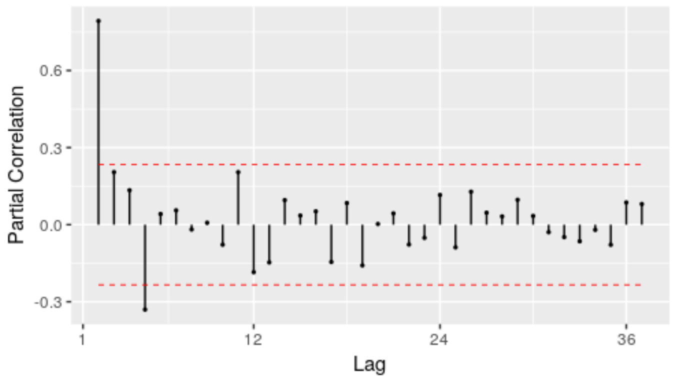


Figure 3: Partial autocorrelation function of $\nabla\nabla^{12}Z_t$

The ACF in Figure 2 and PACF in Figure 3 could have been generated by a *SARIMA(1,0,0)(0,0,0)* process. The explanation is straightforward: the ACF exhibits exponential decay and the PACF has a sharp cutoff after the first lag. This is a textbook example of an AR(1) process. Therefore, the first model proposed for Z_t will be a *SARIMA(1,1,0)(0,1,0)[12]* model.

II. Estimation

To estimate the coefficients of the *SARIMA(1,1,0)(0,1,0)[12]* model, we will use the `Arima()` function from the **forecast** package. The *t* tests will be performed using the `t_test()` function, which receives an object of type `"arima"` or `"Arima"`, an integer representing the number of exogenous variables in the model and an integer for the desired significance level. It returns a data frame containing information related to the test (estimated coefficients, standard errors, test statistics, critical values and the results of the test).

```
# Estimation of the model parameters
model1 <- Arima(data, order = c(1,1,0),
                seasonal = list(order = c(0,1,0), period = 12))

# t-test with the estimated coefficients
# 1% significance level
t_test(model1, alpha = 0.01)

#>        Coeffs Std.Errors        t Crit.Values Rej.H0
#> ar1 -0.386501  0.1099599 3.514925    2.639505   TRUE
```

We conclude from column Rej.H0 that the AR(1) coefficient, when estimated by maximum likelihood, is statistically significant at a 99% confidence level .

III. Diagnostic Tests

The aim of the diagnostic tests is to determine whether the chosen model is suitable. Here, two well-known tools will be used: analysis of standardized residuals and the Llung-Box test (Ljung and Box, 1978).

The graph of the standardized residuals will be produced with the std_resid() function, which was implemented specifically for this purpose.

```
# Graph of the standardized residuals
resids <- std_resid(model1, alpha = 0.01)

# Highlight outlier
points(2013 + 1/4, resids[52], col = "red")
```

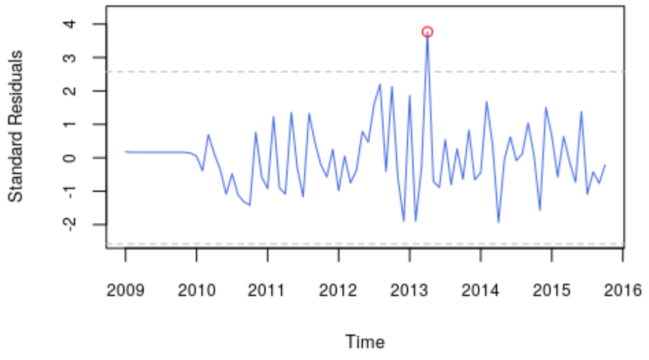


Figure 4: Standard residuals from the first proposed model.

We can see that there is a prominent and statistically significant outlier in April 2013. A second model including a dummy (a time series whose values can be either 0 or 1, to control for the structural break) is therefore proposed. The dummy is defined as follows:

$$D1_t = \begin{cases} 1, t = \text{April 2013} \\ 0, \text{otherwise} \end{cases} \tag{5}$$

This dummy can be created using the dummy() function, as shown below. The start and end parameters indicate time period covered by the dummy. The from and to fields indicate the interval within which the dummy should take a value of 1.

```
dummy1 <- dummy(start = c(2009,1), end = c(2015,10),
                year = 2013, month = 4)
```

Estimation of this model by maximum likelihood resulted in coefficients that are statistically different from 0 at a 1% significance level. The graph of the standardized residuals of the new model (Figure 5) also shows that the decision to include D_t was correct as there is no more evidence of a structural break.

```
# Estimation of the parameters of the model with a dummy
model2 <- Arima(data, order = c(1,1,0),
                seasonal = list(order = c(0,1,0), period = 12), xreg = dummy1)

# t-test with the estimated coefficients
# 1% significance level
t_test(model2, alpha = 0.01)

#>            Coeffs Std.Errors        t Crit.Values Rej.H0
#> ar1    -0.4063976  0.1095272 3.710472    2.639505   TRUE
#> dummy   5.2072304  1.3552891 3.842155    2.639505   TRUE
```

```
resids <- std_resid(model2, alpha = 0.01)

# Highlight November 2008
points(2013 + 1/4, resids[52], col = "red")
```

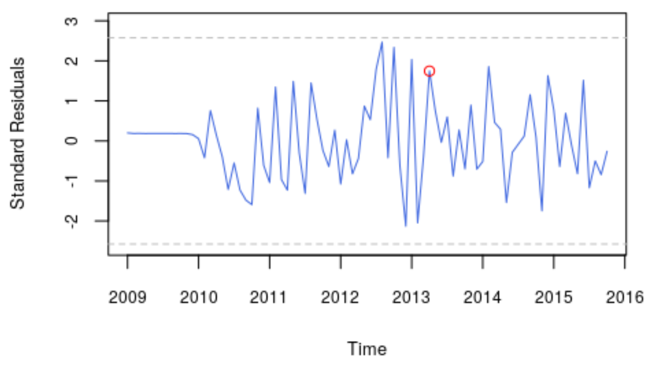


Figure 5: Standard residuals of the model proposed after the detection of a structural break

We spotted some other possible outliers in the standard residuals graph. Hence we introduce two more dummies to improve the model and then estimate a third set of coefficients.

$$D2_t = \begin{cases} 1, t = \text{December 2012} \\ 0, \text{otherwise} \end{cases} \tag{6}$$

$$D3_t = \begin{cases} 1, t = \text{January 2013 or February 2013} \\ 0, \text{otherwise} \end{cases} \tag{7}$$

```
dummy2 <- dummy(start = c(2009,1), end = c(2015,10), year = 2012, month = 12)
dummy3 <- dummy(start = c(2009,1), end = c(2015,10), year = 2013, month = c(1,2))
dummy <- cbind(dummy1,dummy2,dummy3)

model3 <- Arima(data, order = c(1,1,0),
                seasonal = list(order = c(0,1,0), period = 12), xreg = dummy)
```

One possible way of evaluating the quality of the models is to calculate the value of an information criterion. This is already contained in the object returned by `Arima()`.

```
# Show BIC for the two estimated models
model1$bic
#> [1] 335.7308
model3$bic
#> [1] 334.3813
```

Note also that the Bayesian Information Criterion (`BIC`, Schwarz (1978)) for the model with the dummies is smaller. Hence, based on this criterion as well, the model with the dummy should be preferred over the previous one.

The Ljung-Box test for the chosen model can be performed with the `Box.test()` function in the **stats** package. We will follow Rob Hyndman's suggestion[4] and use $min(2m, T/5)$ lags, where m is the period of seasonality (12 here) and T is the length of the time series (82), which yields 16 in our case.

```
# Ljung-Box test on the residuals of the model with a dummy
Box.test(resid(model3), type = "Ljung-Box",lag = 16)

#>          Box-Ljung test

#> data:  resid(model2)
#> X-squared = 27.073, df = 16,
#> p-value = 0.04067
```

[4]The discussion can be found at https://robjhyndman.com/hyndsight/ljung-box-test/.

The p-value of 0.041 is low and indicates there is no statistical evidence to reject the null hypothesis of no autocorrelation in the residuals. We could have added one MA term to double the p-value and generate such evidence, but we decided not to, for two reasons: (i) the law of parsimony; (ii) adding an extra term would be an *ad hoc* solution - we have no indication from the ACF and PACF that this term should exist; and (iii) there is no reason to be overcautious and present an extensive analysis, because our intention is to exhibit **BETS**' features.

IV. Forecasts

BETS provides a convenient way of making forecasts with SARIMA, GRNN, and exponential smoothing models. The `predict()` function receives the parameters from the `forecast()` function (from the package of the same name) and returns not only the objects containing the information related to the forecasts, but also produces a graph of the series including the predicted values. Displaying the information in this way is important to get a better idea of how suitable the model is. The actual values for the forecasting period can also be shown if desired.

We now call `predict()` to generate the forecasts for the proposed model. The parameters `object` (object of type `arima` or `Arima`), `h` (forecast horizon), and `xreg` (the dummy for the forecasting period) are inherited from the `forecast()` function. The others are from `predict()` itself and control features of the graph, except for `actual`, which is the the series of effective values observed during the forecasting period.

```
new_dummy <- dummy(start = start(data_test), end = end(data_test))
new_dummy <- cbind(new_dummy, new_dummy, new_dummy)

preds <- predict(object = model3, xreg = new_dummy,
                 actual = data_test, xlim = c(2010, 2016.2),
                 ylab = "R$ Millions",
                 style = "normal", legend.pos = "bottomleft")
```

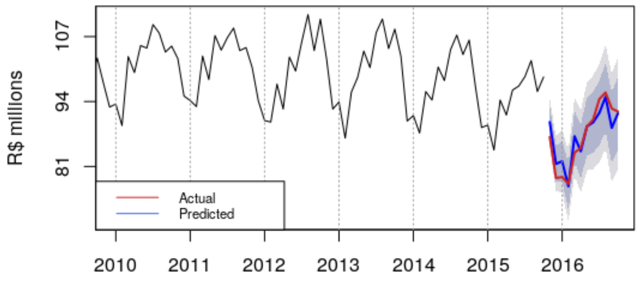


Figure 6: Graph of the proposed SARIMA model forecasts, an output of predict

The areas in dark blue and light blue around the forecasts are the 85% and 95% confidence intervals, respectively. It appears that the forecasts were satisfactory, since in general the actual values are inside the confidence interval.

To make this statement more meaningful, we can check various measures of fit by consulting the accuracy field contained within the predictions object (`preds`).

```
preds$accuracy

#>                      ME     RMSE  MAE        MPE    MAPE      ACF1
#> Test set -0.08333333 2.252776 1.85 -0.2144377 2.14463 0.3403901
#>          Theil's U
#> Test set 0.5675735
```

The other field in this object, `predictions`, contains the object returned by `forecast()` (or by another forecasting function, such as `grnn.test()` or `predict()`, depending on the model).

Using the automated report for SARIMA modeling

The `report()` function performs the Box & Jenkins modeling for any series chosen by the user and generates reports with the results and explanatory comments. Besides, it allows the forecasts of the model to be saved in a data file and accepts parameters in order to increase model flexibility. Each input series produce a different output, as expected. The `report()` prototype is as follows:

```
report(mode = "SARIMA", ts = 21864, parameters = NULL,
       report.file = NA, series.saveas = "none")
```

If the user changes the value of the parameter `series.saveas()` from 'none' to a valid format, the series and the model forecasts are saved in a file. `series.saveas` accepts all the formats `save()` functions can handle (namely, '.sas', '.dta' and '.spss') plus '.csv', which is very convenient. The argument `ts` is just as flexible, and is capable of receiving a list containing `ts` objects, codes of series within the **BETS** database, or a combination of both. A report is generated for each element on this list. For instance, the following piece of code would work:

```
series <- list(21863, BETSget(21864))

parameters <- list(cf.lags= 25, n.ahead = 15 )

report(ts = series, parameters = parameters)
```

The list of reports' specific arguments are passed through `parameters`, whose accepted fields vary according to the type of report (defined via `mode`). In the case of SARIMA models, these fields are presented in table 1, all of them being optional.

Name	Type	Description
`cf.lags`	`integer`	Maximum number of lags to show on the ACFs and PACFs
`n.ahead`	`integer`	Prevision horizon (number of steps ahead)
`inf.crit`	`character`	Information criterion to be used in model selection (BIC or AIC).
`dummy`	`ts` object	A dummy regressor. Must also cover the forecasting period.
`ur.test`	`list`	Parameters of `ur_test`
`arch.test`	`list`	Parameters of `arch_test`
`box.test`	`list`	Parameters of `Box.test`, from package **stats**

Table 1: Possible fields of `parameters`, an argument of `report` function, when `mode` is set to SARIMA

To model the BPIGI series in a similar way to what we modeled in the last section, we resort to the call below:

```
params = list(
  cf.lags = 48,
  n.ahead = 12,
  box.test = list(lag = 16),
  dummy = dummy(start= c(2009,1) , end = c(2016,4) , year = 2013, month = 4),
  ur.test =  list(mode = "ADF", type = "none", lags = 12,
                  selectlags = "BIC", level = "5pct")
)

report(ts = window(BETSget(21864), start= c(2009,1) , end = c(2015,10)),
       parameters = params,
       series.saveas = "csv")
```

The report is rendered in the form of an *html* file, which opens automatically in the user's default internet browser. This whole file is shown in Figure 7. It consists of the following:

1. The information about the series as it is found in the **BETS** metadata table. In our example call, we use a custom series, so it exhibits more general information.

Fitted SARIMA Model

BETS Package

2017-09-04

User-Defined Parameters

Parameter	Value	Variable
Series Code	None	ts
Maximum Lag	48	af.lags
Prevision Horizon	12	n.ahead
Unit Root Test	ADF, drift, 11, BIC, 5pct	ur.test
ARCH Test	12, FALSE, 0.05	arch.test
Box Test	2	box.test
Dummy	TRUE	dummy

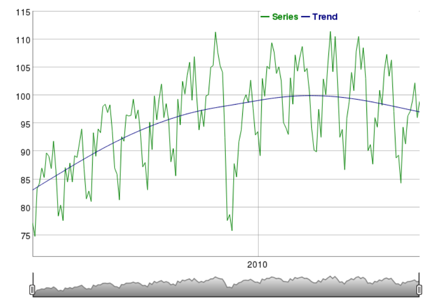
Information About the Series

```
data <- ts
```

Code	Description	Periodicity	Start	Source	Unit
None		12	2002.1	Custom	

Graph

```
## library(mFilter)
trend = fitted(hpfilter(data))

library(dygraphs)
dygraph(cbind(Series = data, Trend = trend), main = info[,"Description"]) %>%
  dyRangeSelector(strokeColor = "gray", fillColor = "gray") %>%
    dyAxis("y", label = info[,"Unit"])
```



Unit Root Tests

Augmented Dickey-Fuller

```
 test.params = append(list(y = data), ur.test)
 df = do.call(BETS.ur_test,test.params)
 df$results
```

```
##      statistic crit.val rej.H0
## tau2 -2.420907    -2.88     no
## phi1  3.531397     4.63    yes
```

For the chosen confidence interval, the test statistic is greater than the critical value. We therefore conclude that there must be a unit root.

Now, we are going to repeatedly apply `diff` to the series and check if the diferenced series has a unit root.

```
 ns_roots = 0
 d_ts = data

 while(df$results[1,"statistic"] > df$results[1,"crit.val"]){
  ns_roots = ns_roots + 1
  d_ts = diff(d_ts)
  test.params = append(list(y = d_ts), ur.test)
  df = do.call(BETS.ur_test,test.params)
  print(df$results)
}
```

```
##      statistic crit.val rej.H0
## tau2 -3.030349    -2.88    yes
## phi1  4.616128     4.63    yes
```

These tests found that there must be a total of 1 unit root(s)

Osborn-Chui-Smith-Birchenhall

This test will be performed for lag 12, that is, the frequency of the series.

```
library(forecast)
s_roots = nsdiffs(data)
print(s_roots)
```

```
## [1] 1
```

This result holds for a 5% signficance level and means that, according to the OCSB test, there must be a total of 1 seasonal unit root(s)

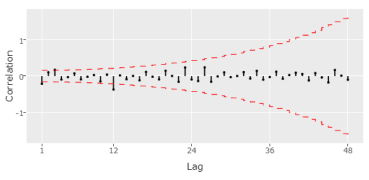
Auto-Correlation Functions

ACF and PACF - After Differencing

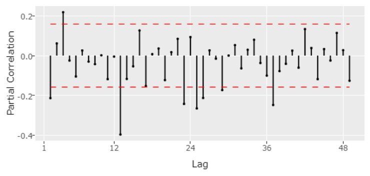
As we saw earlier, this series probably has 1 non-seasonal unit root(s) and 1 seasonal unit root(s) . It means we have look into the correlograms of the differenced series.

```
d_ts <- diff(d_ts, lag = frequency(data), differences = s_roots)
```

```
BETS.corrgram(d_ts, lag.max = af.lags, mode = "bartlett", knit = T)
```



```
BETS.corrgram(d_ts, lag.max = af.lags, mode = "simple", type = "partial", knit = T)
```



Model Identification and Estimation

The correlograms from last section gives us enough information to try to identify the underlying SARIMA model parameters. We can confirm our guess by running the `auto.arima` function from the package `forecast` . By default, this function uses the AICc (Akaike Information Criterion with Finite Sample Correction) for model selection. Here, we are going to use BIC (Bayesian Information Criterion), in which the penalty term for the number of parameters in the model is larger than in AIC.

Since a dummy has to be included, we are going to separate it in two samples, one to build the model and other to make the forecasts.

```
m.dummy = window(dummy, end = end(data))
f.dummy = tail(dummy, n.ahead)
```

```
model <- auto.arima(data, ic = tolower(inf.crit), test = tolower(ur.test$mode),
              max.d = ns_roots, max.D = s_roots, xreg = m.dummy)
summary(model)
```

```
## Series: data
## Regression with ARIMA(2,0,0)(1,0,0)[12] errors
##
## Coefficients:
##          ar1     ar2    sar1  intercept  m.dummy
##       0.7001  0.1859  0.8411    90.6785   5.7587
## s.e.  0.0758  0.0760  0.0369     8.4818   1.4790
##
## sigma^2 estimated as 7.468:  log likelihood=-408.26
## AIC=828.52   AICc=829.05   BIC=847.19
##
## Training set error measures:
##                    ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 0.157744 2.691243 2.107117 0.1049055 2.221923 0.4921826
##                    ACF1
## Training set 0.01353842
```

We see that, according to BIC, the best model is a SARIMA(2,0,0)(1,0,0)[12]. Nevertheless, this is not the end. We still have to test for heteroskedasticity in the residuals. We can use an ARCH test with this purpose.

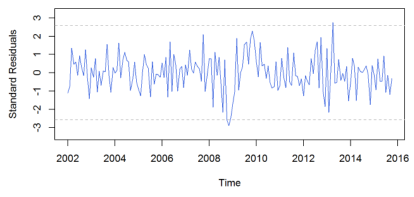
```
arch.params <- append(list(x = resid(model)), arch.test)
at <- do.call(BETS.arch_test, arch.params)
at
```

```
##   statistic      p.value   htk
## 1   34.7783 0.0005079266 FALSE
```

The p.value of 0 is smaller than the significance level of 0.05. We therefore conclude that the residuals are not heteroskedastic.

The next function outputs the model's standardized residuals. If they are all inside the confidence interval, it means the behaviour of the series was well captured by the model. If few residuals are outside the confidence interval, using a dummy to handle structural breaks should be considered. But if most residuals are outside the interval, then SARIMA might not be the appropriate choice.

```
rsd <- BETS.std_resid(model, alpha = 0.01)
```



As a final step in model evaluation, we are going to apply the test to check for autocorrelation in the residuals.

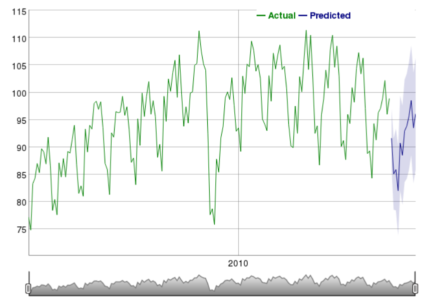
```
test.params <- append(list(x = resid(model)), box.test)
bt = do.call(Box.test,test.params)
bt$p.value
```

```
## [1] 0.2843234
```

The p-value is greater than 0.05, which means there is not enough statistical evidence to reject the null hypothesis of no autocorrelation in the residuals. This is a desirable result.

Forecasts

```
BETS.predict(model, h=n.ahead, xreg = f.dummy,
         main = info[,"Description"], ylab = info[,"Unit"], knit = T)
```

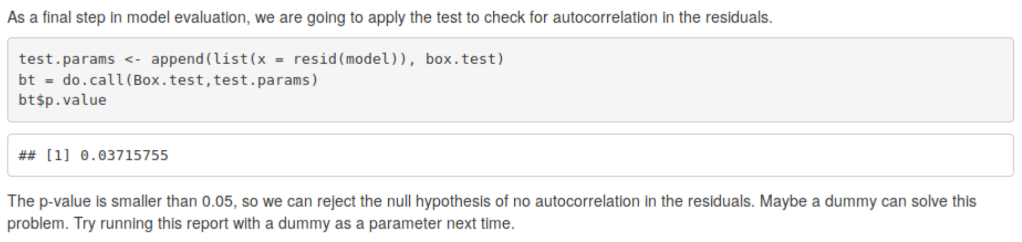


The whole series and the model predictions are available at THIS LINK

Figure 7: The output file of `report()` function for the series under study.

2. The graph of the series along with its trend, which is extracted with an HP Filter (Hodrick and Prescott, 1997). The graph is produced with the **dygraphs** package (Vanderkam et al., 2016) and enables the user to define windows and examine the series' values.
3. The steps involved in the identification of a possible model: unit root tests (at the moment, ACF (Dickey and Fuller, 1979), PP (Phillips and Perron, 1988) or KPSS (Kwiatkowski et al., 1992) for non-seasonal unit roots, and OCSB (Osborn et al., 1988) test for seasonal unit roots) and correlograms of the original series (if no unit root is found), or the differenced series (if one or more unit roots are found).
4. Estimation of the parameters and display of the result from the automatic model selection performed by the `auto.arima()` function (from the **forecast** package).
5. The ARCH test (Engle, 1982) for heteroskedasticity in the residuals. If residuals are found to be heteroskedastic, the report performs a log transformation on the original series and run steps 3 and 4 again for the transformed series.
6. The graph of the standard residuals.
7. The Ljung-Box (Ljung and Box, 1978) test for autocorrelation in the residuals. If the null hypothesis of no autocorrelation is rejected, the user will be advised to use a dummy or another type of model.
8. The n-steps-ahead forecasts and a graph of the original series with the forecasted values and confidence intervals, also produced with `dygraphs` (this output is an option of `predict()`).
9. A link to the file containing the original data and the forecasts.

One example of how the output could change in response to different inputs is obtained by removing the dummy from the last example. The call without a dummy renders a report that, after detecting autocorrelation in the residuals, suggests the usage of a dummy to solve the problem (Figure 8). It does indeed solve it, as we demonstrated.


As a final step in model evaluation, we are going to apply the test to check for autocorrelation in the residuals.

```
test.params <- append(list(x = resid(model)), box.test)
bt = do.call(Box.test,test.params)
bt$p.value
```

```
## [1] 0.03715755
```

The p-value is smaller than 0.05, so we can reject the null hypothesis of no autocorrelation in the residuals. Maybe a dummy can solve this problem. Try running this report with a dummy as a parameter next time.

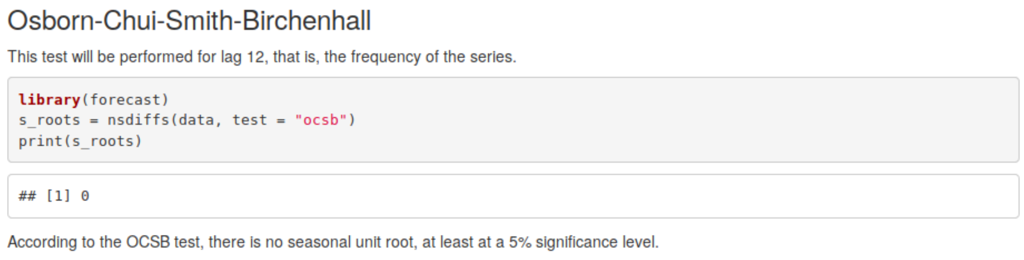
Figure 8: Snippet of the report generated without a dummy.

We now provide one more call and output snippets from the `report()` function, aiming at illustrating its capabilities. First, suppose we need to model the Brazilian production of capital goods (BPCG) series. Its unique identifier in the **BETS** database is 21863, so this will be the value of parameter `ts`. The next piece of code builds a SARIMA report using the same dummy we needed in the case study, but ARCH test and Box test parameters are different.[5]

```
dum <- dummy(start= c(2002,1) , end = c(2017,12) , from = c(2008,7) , to = c(2008,11))
series <- window(BETSget(21863), start = c(2002,1), end = c(2017,6))

params <- list(
    cf.lags = 36,
    n.ahead = 6,
    dummy = dum,
    arch.test = list(lags = 12, alpha = 0.03),
    box.test = list(type = "Box-Pierce")
)
report(ts = series, parameters = params)
```

Figures 9 to 11 show snippets of the report that was generated by the last call and demonstrate that it presents different results when compared to the one produced previously. OCSB test, for instance, did not find any seasonal unit root, thereby displaying a different comment (Figure 9).

[5]This time, we employ the Box-Pierce test (Box and Pierce, 1970)



Osborn-Chui-Smith-Birchenhall

This test will be performed for lag 12, that is, the frequency of the series.

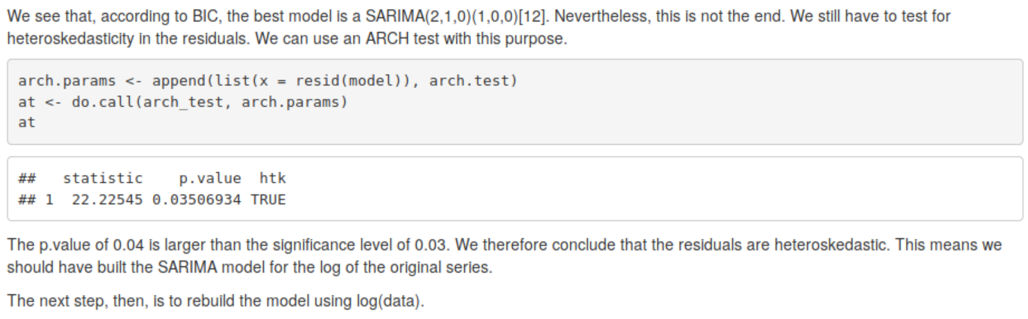
```
library(forecast)
s_roots = nsdiffs(data, test = "ocsb")
print(s_roots)
```

```
## [1] 0
```

According to the OCSB test, there is no seasonal unit root, at least at a 5% significance level.

Figure 9: First snippet of the example report with the BPCG series

In addition, residuals were found to be heteroskedastic, which forced the model to be estimated a second time, using the log transformation of the original series (Figure 10).



We see that, according to BIC, the best model is a SARIMA(2,1,0)(1,0,0)[12]. Nevertheless, this is not the end. We still have to test for heteroskedasticity in the residuals. We can use an ARCH test with this purpose.

```
arch.params <- append(list(x = resid(model)), arch.test)
at <- do.call(arch_test, arch.params)
at
```

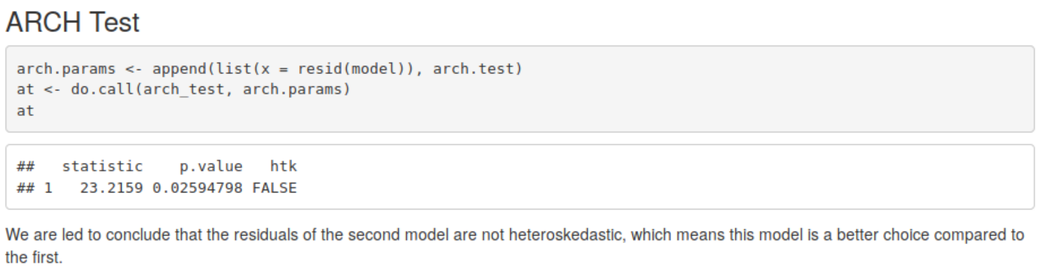
```
##   statistic    p.value  htk
## 1  22.22545 0.03506934 TRUE
```

The p.value of 0.04 is larger than the significance level of 0.03. We therefore conclude that the residuals are heteroskedastic. This means we should have built the SARIMA model for the log of the original series.

The next step, then, is to rebuild the model using log(data).

Figure 10: Second snippet of the example report with the BPCG series

Finally, the second model was tested and considered better than the first (Figure 11) and no autocorrelation in the residuals were found.



ARCH Test

```
arch.params <- append(list(x = resid(model)), arch.test)
at <- do.call(arch_test, arch.params)
at
```

```
##   statistic    p.value   htk
## 1   23.2159 0.02594798 FALSE
```

We are led to conclude that the residuals of the second model are not heteroskedastic, which means this model is a better choice compared to the first.

Figure 11: Third snippet of the example report with the BPCG series

Note that running a report saves a lot of time and might be insightful. However, performing the analysis in detail, using individual **BETS** functions rather than the batch of functions included in the reports, may provide more flexibility. Since reports use `auto.arima()`, they might not find models that are in line with a more sofisticated analysis. For example, the SARIMA model calculated by `auto.arima()` in the first example (Figure 7) is not the one we proposed in the case study. This happened because the function did not find a non-seasonal unit root, since it does not test for more than 2 lags, and we tested for 6. For now, we cannot include the option of testing for more lags, the reason being that `auto.arima()` does not accept it.

Conclusion

We have shown in this article that the approach adopted in **BETS** is a novel one as it allows users not only to quickly and easily access thousands of Brazilian economic time series, but also to use and analyze these series in many different ways. Its automated reports are comprehensive, fully customizable and three methods can be readily applied: SARIMA models, GRNNs and exponential smoothing models. We showed case studies to examine the use of the first, but only mentioned the latter two. Nevertheless, the user can benefit from the extensive documentation shipped with the package, where example code to generate GRNN and exponential smoothing reports are exhibited and discussed.

In the future we hope to expand these dynamic reports and provide new options for users. One such option has already been taken into account in the design of the package: a display mode that does not include explanations about the methodology, so that the information is provided in a more technical, succinct manner. In addition to developing new modeling methods, such as *Multi-Layer Perceptrons*, fuzzy logic, Box & Jenkins with transfer functions, and multivariate models, we plan to refine the analyses themselves.

Bibliography

J. J. Allaire. *Rmarkdown: Dynamic Documents for R*, 2017. URL `http://cran.r-project.org/package=rmarkdown`. [p133]

M. Balcilar. *Mfilter: Miscellaneous Time Series Filters*, 2016. URL `http://cran.r-project.org/package=mfilter`. [p133]

M. S. Bartlett. On the theoretical specification and sampling properties of autocorrelated time series. *Supplement to the Journal of the Royal Statistical Society*, 1946. [p137]

G. E. P. Box and G. M. Jenkins. *Time Series Analysis Forecasting and Control*. Holden Day, San Francisco, 1970. [p133]

G. E. P. Box and D. A. Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association*, 65(332):1509–1526, 1970. [p144]

D. A. Dickey and W. A. Fuller. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, 74, 1979. [p136, 144]

R. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica*, 50(4):987–1007, 1982. [p144]

P. C. Ferreira, D. Mattos, D. Almeida, I. de Oliveira, and R. Pereira. *Análise De Séries Temporais Usando o R*. FGV/IBRE, Rio de Janeiro, RJ, 2016. [p133]

P. C. Ferreira, T. F. Speranza, J. C. Azevedo, F. O. Teixeira, and D. Marcolino. *BETS: Brazilian Economic Time Series*, 2017. URL `http://cran.r-project.org/package=BETS`. [p133]

R. J. Hodrick and E. C. Prescott. Postwar U.S. business cycles: An empirical investigation. *Journal of Money, Credit, and Banking*, 1997. [p144]

R. J. Hyndman. *Forecast: Forecasting Functions for Time Series and Linear Models*, 2015. URL `http://cran.r-project.org/package=forecast`. [p133]

D. Kwiatkowski, P. C. B. Phillips, P. Schmidt, and Y. Shin. Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of Econometrics*, 1992. [p136, 144]

G. M. Ljung and G. E. P. Box. On a measure of a lack of fit in time series models. *Biometrika*, 65, 1978. [p139, 144]

J. Ooms, D. James, S. DebRoy, H. Wickham, and J. Horner. *RMySQL: Database Interface and MySQL Driver for R*, 2016. URL `http://cran.r-project.org/package=RMySQL`. [p133]

D. R. Osborn, A. P. L. Chui, J. Smith, and C. R. Birchenhall. Seasonality and the order of integration for consumption. *Oxford Bulletin of Economics and Statistics*, 1988. [p137, 144]

B. Pfaff, E. Zivot, and M. Stigler. *Urca: Unit Root and Cointegration Tests for Time Series Data*, 2016. URL `http://cran.r-project.org/package=urca`. [p133]

P. C. Phillips and P. Perron. Testing for a unit root in time series regression. *Biometrika*, 1988. [p136, 144]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. URL `https://www.R-project.org/`. [p133, 364]

C. Sax. *Seasonal: R Interface to X-13-ARIMA-SEATS*, 2016. URL `https://cran.r-project.org/web/packages/seasonal/index.html`. [p133]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6, 1978. [p140]

D. F. Specht. A general regression neural network. *IEEE Transactions on Neural Networks*, 1991. [p133]

D. Vanderkam, J. J. Allaire, J. Owen, D. Gromer, P. Shevtsov, and B. Thieurmel. *Dygraphs: Interface to 'Dygraphs' Interactive Time Series Charting Library*, 2016. URL `https://cran.r-project.org/web/packages/dygraphs/index.html`. [p144]

H. Wickham. *Rvest: Easily Harvest (Scrape) Web Pages*, 2016. URL `http://cran.r-project.org/package=rvest`. [p133]

H. Wickham. *Httr: Tools for Working with URLs and HTTP*, 2017. URL `http://cran.r-project.org/package=httr`. [p133]

Pedro C. Ferreira
Instituto Brasileiro de Economia (IBRE)
Fundação Getúlio Vargas (FGV)
`pedro.guilherme@fgv.br`

Talitha F. Speranza
Instituto Brasileiro de Economia (IBRE)
Fundação Getúlio Vargas (FGV)
`talitha.speranza@fgv.br`

Jonatha A. da Costa
Instituto Brasileiro de Economia (IBRE)
Fundação Getúlio Vargas (FGV)
`jonatha.costa@fgv.br`

fICA: FastICA Algorithms and Their Improved Variants

by Jari Miettinen, Klaus Nordhausen, and Sara Taskinen

Abstract In independent component analysis (ICA) one searches for mutually independent nongaussian latent variables when the components of the multivariate data are assumed to be linear combinations of them. Arguably, the most popular method to perform ICA is FastICA. There are two classical versions, the deflation-based FastICA where the components are found one by one, and the symmetric FastICA where the components are found simultaneously. These methods have been implemented previously in two R packages, **fastICA** and **ica**. We present the R package **fICA** and compare it to the other packages. Additional features in **fICA** include optimization of the extraction order in the deflation-based version, possibility to use any nonlinearity function, and improvement to convergence of the deflation-based algorithm. The usage of the package is demonstrated by applying it to the real ECG data of a pregnant woman.

Introduction

The goal of independent component analysis is to transform a multivariate dataset so that the resulting components are as independent as possible. The idea is to separate latent components from the dataset which is assumed to consist of linear mixtures of them.

The basic independent component (IC) model is thus written as

$$\mathbf{x}_i = A\mathbf{z}_i,\ i = 1,\ldots,n, \tag{1}$$

where $X = (\mathbf{x}_1,\ldots,\mathbf{x}_n)$ is a centered $p \times n$ data matrix, A is a $p \times p$ mixing matrix, and $\mathbf{z}_1,\ldots,\mathbf{z}_n$ are realizations of a zero mean random p-vector $\mathbf{z} = (z_1,\ldots,z_p)^T$ with mutually independent components and at most one Gaussian component. The aim of ICA is then to estimate the unmixing matrix $W = A^{-1}$ and/or Z from the data matrix X alone (Hyvärinen et al., 2001; Cichocki and Amari, 2002; Comon and Jutten, 2010; Nordhausen and Oja, 2018). Indeed, ensuring independence and nongaussianity of the components of $\mathbf{z}$ is sufficient for their consistent estimation, but only up to signs and scales (Comon, 1994). The scales of the independent components are often fixed by the assumption $\text{Cov}(\mathbf{z}) = I_p$, and we may proceed from Model 1 to

$$\mathbf{x}_{st} = U^T\mathbf{z}, \tag{2}$$

where $\mathbf{x}_{st} = \text{Cov}(\mathbf{x})^{-1/2}(\mathbf{x} - \mathbb{E}(\mathbf{x}))$ and U is an orthogonal $p \times p$ matrix. This means the problem can be reduced to that of finding an orthogonal matrix and the final unmixing matrix estimate is then $W = U\text{Cov}(\mathbf{x})^{-1/2}$.

ICA has become a widely used tool in various fields, including brain imaging, image and audio signal processing, and financial time series analysis. Despite the vast amount of literature on ICA methodology, the classical FastICA (Hyvärinen and Oja, 1997; Hyvärinen, 1999) methods are still the most popular tools with which to estimate the independent components. FastICA functions are available for several programming languages such as Matlab, in package FastICA (Gävert et al., 2005); R, in packages **fastICA** (Marchini et al., 2013) and **ica** (Helwig, 2015); C++, as part of package IT++ (Cayre and Furon, 2004); and Python, as part of packages MDP (Zito et al., 2008) and Scikit-learn (Pedregosa et al., 2011).

Here we introduce the R package **fICA** (Miettinen et al., 2017b). In addition to the classical FastICA methods, **fICA** includes functions for three recently proposed improved variants of FastICA, which are presented in the following section. Then we discuss how the **fICA** package differs from the packages **fastICA** and **ica**, and finally, we give some examples on the usage of the **fICA** package.

FastICA estimators in the fICA package

The **fICA** package includes implementations of the classical FastICA estimators as well as three improved variants, which we will now describe in detail. For other variants of FastICA, see Koldovský and Tichavský (2015).

All the methods maximize the nongaussianity of the components of $U\mathbf{x}_{st}$, when the nongaussianity of a univariate random variable x is measured by $|\mathbb{E}[G(x)]|$ with some twice continuously differentiable and nonquadratic function G which satisfies $\mathbb{E}[G(y)] = 0$ for the standard Gaussian random variable y. The established and most popular options for G are

$$\begin{aligned}&\textit{pow3}:\ G(x) = (x^4 - 3)/4,\\ &\textit{tanh}:\ G(x) = log(cosh(x)) - c_t\ ,\\ &\textit{gaus}:\ G(x) = -\exp(-x^2/2) - c_g\ ,\end{aligned}$$

where $c_t = \mathbb{E}[\log(\cosh(y))] \approx 0.375$ and $c_g = \mathbb{E}[-\exp(-y^2/2)] \approx -0.707$. The names *pow3*, *tanh*, and *gaus* originate from the first derivatives of the functions, the so called nonlinearities, $g(x) = x^3$, $g(x) = \tanh(x)$, and $g(x) = x\exp(-x^2/2)$. The choice *pow3* always yields consistent estimates, whereas *tanh* and *gaus* do not. However, the examples (Miettinen et al., 2017c; Wei, 2014) of such distributions of the independent components where *tanh* and *gaus* fail are quite artificial, and we agree with Hyvärinen (1999) stating that *tanh* and *gaus* work with most of the reasonable distributions. For further discussions concerning possible nonlinearities and their properties, see Dermoune and Wei (2013) and Virta and Nordhausen (2017).

Deflation-based FastICA and its variants

The first FastICA paper (Hyvärinen and Oja, 1997) used *pow3* to find the independent components one after the other. In this *deflation-based FastICA*, the kth row of $U = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p)^T$ maximizes

$$|\,\mathbb{E}[G(\boldsymbol{u}_k^T \boldsymbol{x}_{st})]|$$

under the constraints $\boldsymbol{u}_k^T\boldsymbol{u}_k = 1$ and $\boldsymbol{u}_j^T\boldsymbol{u}_k = 0$ for $j = 1, \ldots, k-1$. A modified Newton-Raphson algorithm iterates the following steps until convergence:

$$\begin{aligned}\boldsymbol{u}_k &\leftarrow \mathbb{E}[g(\boldsymbol{u}_k^T\boldsymbol{x}_{st})\boldsymbol{x}_{st}] - \mathbb{E}[g'(\boldsymbol{u}_k^T\boldsymbol{x}_{st})]\boldsymbol{u}_k\\ \boldsymbol{u}_k &\leftarrow \left(\boldsymbol{I}_p - \sum_{l=1}^{k-1}\boldsymbol{u}_l\boldsymbol{u}_l^T\right)\boldsymbol{u}_k\\ \boldsymbol{u}_k &\leftarrow ||\boldsymbol{u}_k||^{-1}\boldsymbol{u}_k\end{aligned}$$

where the last two steps perform the Gram-Schmidt orthonormalization.

In addition to the global maximum, the objective function has several local maxima to which the FastICA algorithm may converge. The order in which the rows of U are found depends on their initial values. The extraction order affects the estimation performance, and to our knowledge, **fICA** is the only R package which provides a function to estimate the rows of U in such order that the objective function is maximized globally at each stage. This extraction order is usually better than a random order, even though not always optimal.

The statistical properties including asymptotic normality of the deflation-based FastICA estimator were studied rigorously in Ollila (2009, 2010); Nordhausen et al. (2011). The derivation of the asymptotic variances of the unmixing matrix estimate lead to *reloaded FastICA* (Nordhausen et al., 2011) which first computes an initial estimate $\widehat{Z}$ of the sources applying some simple IC method such as FOBI (Cardoso, 1989) or k-JADE (Miettinen et al., 2013). For all components of $\widehat{Z}$, one then computes certain quantities from which the optimal extraction order can be deduced.

The *adaptive deflation-based FastICA* (Miettinen et al., 2014) differs from the other estimators here, as instead of one nonlinearity it uses a candidate set of multiple nonlinearities, from which the best one is chosen for each component. Again, an initial estimate of the sources is computed and then the same quantities as in reloaded FastICA are obtained, but now for for each candidate in the set of nonlinearities. In addition to identifying the optimal extraction order, the optimal nonlinearity for each independent component is also chosen separately. The default set of nonlinearities for the function `adapt_fICA` in the R package **fICA** includes *pow3*, *tanh*, *gaus*, and the following eleven options:

$$\begin{array}{ll} g(x) = x^3 & g(x) = (x-0.2)_+^2 + (x+0.2)_-^2\\ g(x) = tanh(x) & g(x) = (x-0.4)_+^2 + (x+0.4)_-^2\\ g(x) = xexp(-x^2/2) & g(x) = (x-0.6)_+^2 + (x+0.6)_-^2\\ g(x) = (x+0.6)_-^2 & g(x) = (x-0.8)_+^2 + (x+0.8)_-^2\\ g(x) = (x-0.6)_+^2 & g(x) = (x-1.2)_+^2 + (x+1.2)_-^2\\ g(x) = (x)_+^2 + (x)_-^2 & g(x) = (x-1.4)_+^2 + (x+1.4)_-^2\\ g(x) = (x-1)_+^2 + (x+1)_-^2 & g(x) = (x-1.6)_+^2 + (x+1.6)_-^2 \end{array}$$

where $(x)_+ = x$ if $x > 0$ and 0 otherwise, and $(x)_- = x$ if $x < 0$ and 0 otherwise.

Symmetric and squared symmetric FastICA

The *symmetric FastICA* (Hyvärinen, 1999) estimator maximizes

$$\sum_{j=1}^{p} | \mathbb{E}[G(\boldsymbol{u}_j^T \boldsymbol{x}_{st}]|$$

under the orthogonality constraint $UU^T = I_p$. Now the rows of $U = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_p)^T$ are estimated in parallel, and the steps of the iterative algorithm are

$$\begin{aligned} &\boldsymbol{u}_k \leftarrow \mathbb{E}[g(\boldsymbol{u}_k^T \boldsymbol{x}_{st})\boldsymbol{x}_{st}] - \mathbb{E}[g'(\boldsymbol{u}_k^T \boldsymbol{x}_{st})]\boldsymbol{u}_k, \ \ k = 1, \ldots, p \\ &U \leftarrow (UU^T)^{-1/2}U. \end{aligned}$$

The first update step is similar to that of the deflation-based version. In a way, the orthogonalization step can be seen as taking an average over the vectors of the first step, while in the Gram-Schmidt orthogonalization the errors in estimating the first rows of U will remain throughout the estimation. It is said that the error accumulates in the deflation-based approach. The symmetric FastICA is usually considered superior to the deflation-based FastICA, but there are also cases where the accumulation is preferable to the averaging. This occurs when some independent components are easier to find than the others. Statistical properties of symmetric FastICA are given in Miettinen et al. (2015); Wei (2015); Miettinen et al. (2017c).

The *squared symmetric FastICA* (Miettinen et al., 2017c) estimator maximizes

$$\sum_{j=1}^{p} (\mathbb{E}[G(\boldsymbol{u}_j^T \boldsymbol{x}_{st}])^2$$

under the orthogonality constraint $UU^T = I_p$. The first step of the algorithm

$$\begin{aligned} &\boldsymbol{u}_k \leftarrow \mathbb{E}[G(\boldsymbol{u}_k^T \boldsymbol{x}_{st})](\mathbb{E}[g(\boldsymbol{u}_k^T \boldsymbol{x}_{st})\boldsymbol{x}_{st}] - \mathbb{E}[g'(\boldsymbol{u}_k^T \boldsymbol{x}_{st})]\boldsymbol{u}_k), \ \ k = 1, \ldots, p, \\ &U \leftarrow (UU^T)^{-1/2}U \end{aligned}$$

is that of the classical symmetric version multiplied by $\mathbb{E}[G(\boldsymbol{u}_k^T \boldsymbol{x}_{st})]$. Hence, the squared symmetric version puts more weight on the rows which correspond to more nongaussian components, which most often, but not always, is advantageous.

For both of the symmetric versions, the initial value of U is not important when the sample size is large, as the algorithms converge always to the global maxima. With small sample sizes the initial value plays a role.

Comparison of the packages

The FastICA algorithms sometimes fail to converge when the sample size is small. However, this is not obvious if one uses the R package **fastICA** or **ica** since they do not give errors or warnings if the algorithm does not converge, but simply return the estimate after the maximum number of iterations has been reached. In the **fastICA** package, the information about convergence can be obtained, but since it is not given in the default setup, many of the users will not notice it. Similarly, in the **ica** package the number of iterations is included in the output and could be used as an indicator for convergence, but convergence problems might still go unnoticed. Growing the number of iterations does usually not help because if the algorithm has not converged after 2000 iterations, it is often repeating a loop of values. In contrast, the functions in the R package **fICA** give an error message if the algorithm does not converge. When computing the symmetric estimators, the user can choose the number of random orthogonal matrices which are generated for the initial values of the algorithms. If this number is at least two, the function returns the estimate which yields the highest value of the objective function and gives a warning if none of the algorithm runs converge. To avoid the repeating loop in the deflation-based case, **fICA** alternates the step size. The update of the vector $\boldsymbol{u}_k$ is of the form

$$\boldsymbol{u}_k \leftarrow w\boldsymbol{u}_k + (1-w)\left(\mathbb{E}[g(\boldsymbol{u}_k^T \boldsymbol{x}_{st})\boldsymbol{x}_{st}] - \mathbb{E}[g'(\boldsymbol{u}_k^T \boldsymbol{x}_{st})]\boldsymbol{u}_k\right),$$

where w is zero for most iterations and takes small nonzero values with some selected iteration numbers with an irregular pattern.

To sum up the key differences of **fICA** to the **fastICA** and **ica** packages, we present Table 1.

The number of nonlinearities in **fICA** is basically unlimited. In addition to the already implemented suggestions, the user can provide their own functions by specifying g and its derivative. We provide

Table 1: The properties of the three R packages which include the FastICA algorithm.

	fICA	**fastICA**	**ica**
Symmetric FastICA	YES	YES	YES
Squared symm. FastICA	YES	NO	NO
Deflation-based FastICA	YES	YES	YES
Optimized extraction order	YES	NO	NO
Adaptive defl. FastICA	YES	NO	NO
Error if no convergence	YES	NO	NO
Number of nonlinearities	∞	2	3

an example of the user-specified option in the next section.

Examples

To illustrate the use of the **fICA** package, we give the following two examples.

Electrocardiography data

First, we analyze an electrocardiography recording (ECG) dataset `foetal_ecg.dat` (de Lathauwer et al., 2000), which can be downloaded from the supplementary files of Miettinen et al. (2017d). An ECG measures the electrical potential on the body surface. This recording is from a pregnant woman and consists of eight signals from five sensors on the stomach and three on the chest. The lengths of the signals are 2500 time points.

The signals are mixtures of fetus' and mother's heart beats and possibly some artifacts. The goal is to separate these sources using independent component analysis.

```
> library(fICA)
> library(JADE)
> options(digits = 4)
>
> dataset <- matrix(scan("https://www.jstatsoft.org/index.php/jss/article/
+ downloadSuppFile/v076i02/foetal_ecg.dat"), 2500, 9, byrow = TRUE)
> X <- dataset[ , 2:9]
> plot.ts(X, nc = 1, main = "Data")
```

After downloading the data and plotting it in Figure 1, we see that the main feature in each signal are 14 peaks, which are presumably produced by mother's heart beats. Next, we show how to apply the function `adapt_fICA` with a modified set of nonlinearities. We demonstrate how to use the pre-programmed *pow3* , *tanh*, and *gaus* options in the packages, and demonstrate how to use a user-specified function by providing g and its derivative for another classical nonlinearity called *skew* given by $g(x) = x^2$.

```
> g <- function(x){ x^2 }
> dg <- function(x){ 2*x }
> gf_new <- c(gf[1:3], g)
> dgf_new <- c(dgf[1:3], dg)
> gnames_new <- c(gnames[1:3], "skew")
>
> res <- adapt_fICA(X, gs = gf_new, dgs = dgf_new, name = gnames_new, kj = 1)
>
> res
W :
         [,1]      [,2]      [,3]      [,4]      [,5]       [,6]       [,7]
[1,] -0.04135  0.059521  0.004006  0.001754 -0.010123  0.0076023  0.0001551
[2,]  0.01731 -0.023294  0.002298  0.002663  0.008367  0.0022727 -0.0075911
[3,]  0.09352  0.071699 -0.106091 -0.004342  0.175369 -0.0030657 -0.0063677
[4,] -0.05358  0.106633  0.076979  0.044062 -0.024241 -0.0261067 -0.0522030
[5,]  0.15805  0.039095  0.233377 -0.041272 -0.169486  0.0104065 -0.0116267
[6,] -0.06605  0.114817  0.201392 -0.021387  0.234210 -0.0106505  0.0254791
[7,]  0.07817 -0.001272  0.146718  0.220175 -0.180189 -0.0008764  0.0035999
```

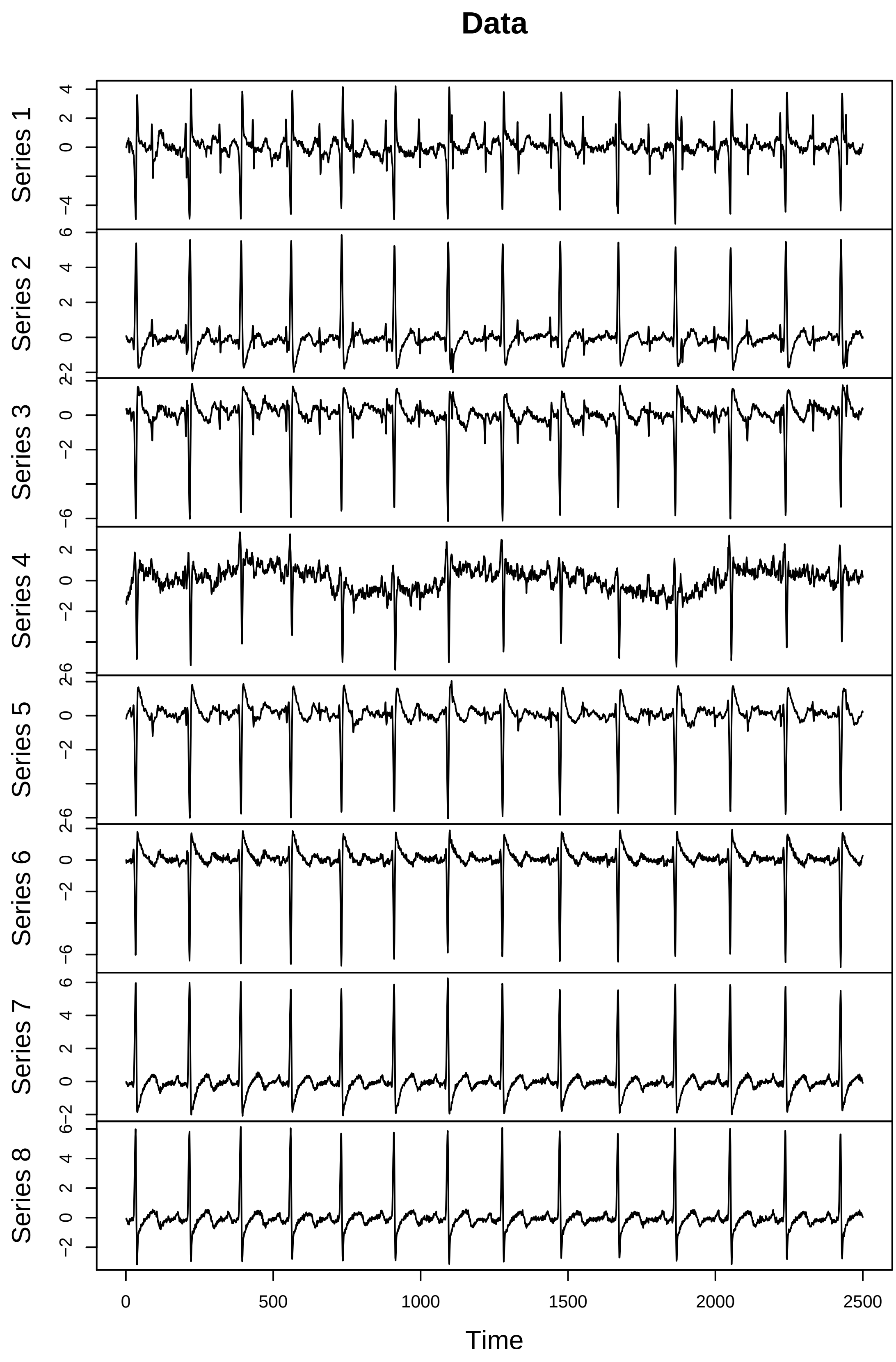


Figure 1: The ECG signals of a pregnant woman.

```
[8,] -0.25492  0.313232  0.049493  0.089799  0.028816  0.0232881 -0.0535985
          [,8]
[1,] -0.009234
[2,]  0.009301
[3,]  0.013034
[4,]  0.022396
[5,]  0.028662
[6,] -0.010261
[7,] -0.009922
[8,]  0.034743

gs :
[1] "pow3" "tanh" "gaus" "skew"

used_gs :
[1] "gaus" "gaus" "gaus" "tanh" "tanh" "tanh" "tanh"

alphas :
     comp 1 comp 2  comp 3 comp 4 comp 5 comp 6 comp 7  comp 8
pow3 0.7445 0.6119  1.4060  1.994  5.014  9.628  23.28 50087.8
tanh 0.2433 0.2569  0.7769  1.374  3.189  8.400  21.53   343.3
gaus 0.2058 0.2336  0.7336  1.458  3.351 10.829  22.87   215.1
skew 0.5457 0.4126 11.8980  7.091 12.867  9.118  25.00   108.7

init_est :
[1] "1-JADE"
```

In the output, we have the 8×8 unmixing matrix, the names of the available nonlinearities, the nonlinearities used, the criterion values for the selection of the extraction order and the nonlinearities, and the name of the initial estimator. Note that the used_gs output does not give a nonlinearity for the last component since the last component is fixed by the seven previous components due to the orthogonality constraint.

```
> plot.ts(bss.components(res), nc=1, main="Estimated components")
```

Figure 2 plots the estimated independent components. Clearly, the first two components are related to mother's heart beat and the third one is related to fetus' heart beat. The third row of the unmixing matrix estimate shows which data signals contribute to the estimate of the fetal heart beat. Not surprisingly, the nonzero coefficients correspond to sensors 1, 2, 3, and 5, which are located on the stomach area.

Simulations

In this simulation example, we also take a look at the related R package **BSSasymp** (Miettinen et al., 2017a), which computes asymptotic covariance matrices of several mixing and unmixing matrix estimates, including the FastICA variants discussed in this paper and implemented in **fICA**. The $p = 3$ independent components in the simulation setup are generated from a t_9-distribution, an exponential distribution with rate 1, and the normal distribution. Each density is standardized so that the expected value is 0 and the variance is 1. In each of the 1000 repetitions, the sample size is 5000, the mixing matrix is the same randomly generated 3×3 matrix, and we collect the unmixing matrix estimates given by (1) the reloaded deflation-based FastICA using *tanh* from **fICA**; (2) deflation-based FastICA using *tanh* from **fastICA**; and (3) deflation-based FastICA using *tanh* from **ica**. The goal of the simulation is to show that the extraction order of the components indeed matters.

```
> library(fICA)
> library(fastICA)
> library(ica)
> library(BSSasymp)
> library(JADE)
>
> options(digits = 4)
> set.seed(1145)
>
> rort <- function(p){qr.Q(qr(matrix(rnorm(p * p), p)))}
>
```

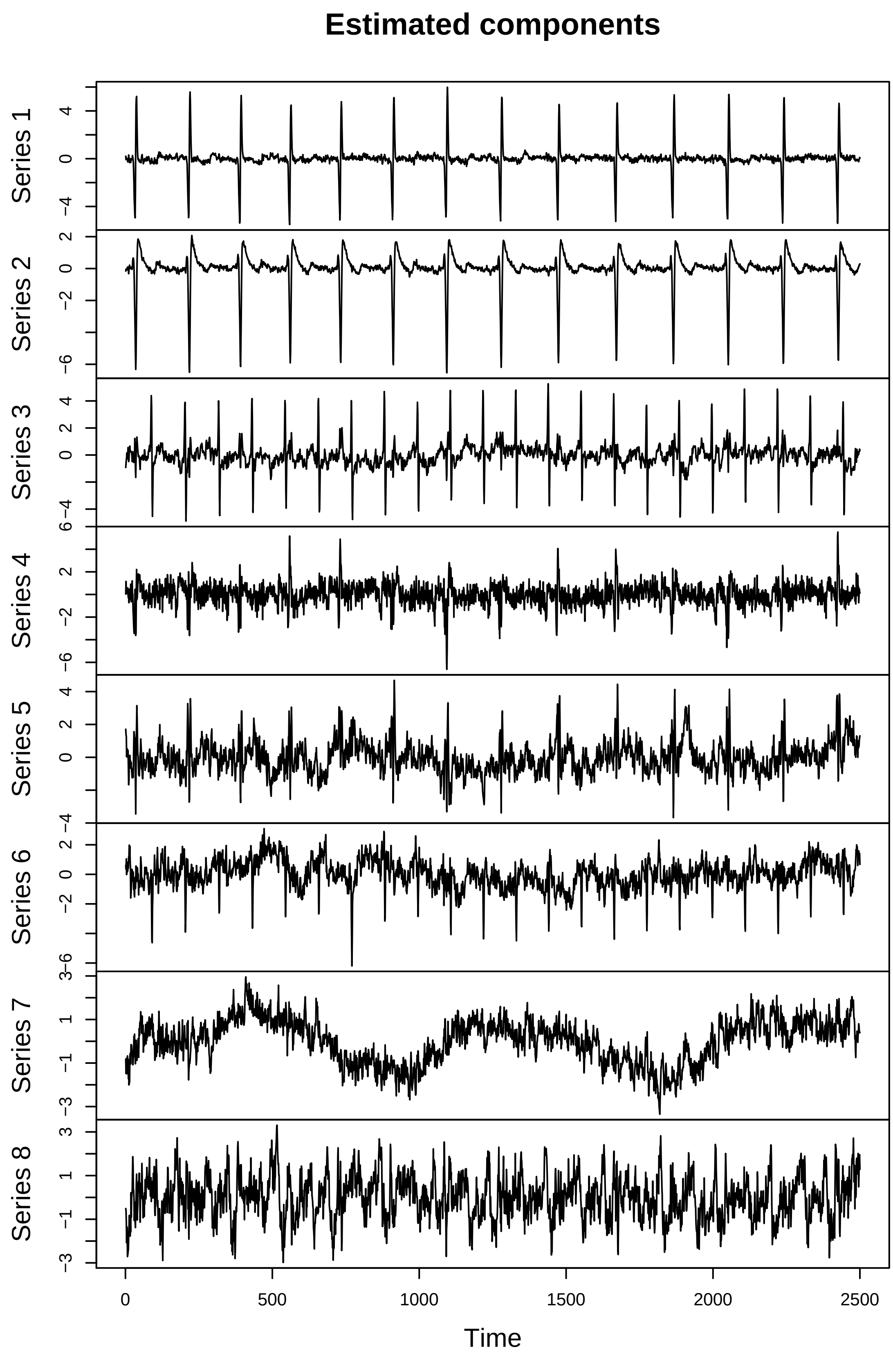


Figure 2: The estimated independent components from the ECG data.

```
>
> n <- 5000
> p <- 3
> A <- matrix(rnorm(p^2), p, p)
>
> Ws1 <- Ws2 <- Ws3 <- vector("list", 1000)
>
> for(i in 1:1000){
+   Z <- cbind(rt(n, 9) / sqrt(9/7), rexp(n, 1) - 1, rnorm(n))
+
+   X <- tcrossprod(Z,A)
+
+   init = rort(p)
+
+   Ws1[[i]] <- reloaded_fICA(X, g = "tanh")$W
+   res <- fastICA(X, n.comp = 3, alg.typ = "deflation" )
+   Ws2[[i]] <- t(res$W) %*% t(res$K)
+   Ws3[[i]] <- icafast(X, nc = 3, alg = "def", Rmat = init)$W
+ }
```

To obtain the theoretical asymptotic variances, the density functions of the independent components and their supports are required. Function `ASCOV_FastICAdefl` computes the values for deflation-based variants of FastICA. However, when the initial matrix is random as it is in **fastICA** and **ica**, there are no asymptotic results. Therefore, we only consider the reloaded FastICA for which the asymptotic variances are derived using `ASCOV_FastICAdefl` with `method = "adapt"` and only one nonlinearity. The output component `$COV_W` gives the asymptotic covariance matrix of the unmixing matrix estimate with the variances on the diagonal. The variances can be estimated using the function `ASCOV_FastICAdefl_est` and the data matrix. Here we estimate the variances from the last dataset generated above. In this case, the estimated variances are slightly higher than the theoretical values. Finally, we compute the variances from the simulations, and notice that they are quite close to the theoretical and the estimated variances.

```
 > f1 <- function(x){ dt(x*sqrt(9/7),9)*sqrt(9/7) }
 > f2 <- function(x){ dexp(x+1,1) }
 > f3 <- function(x){ dnorm(x) }
 > supp <- matrix(c(-Inf, -1, -Inf, Inf, Inf, Inf), p, 2)
 >
 > COV <- ASCOV_FastICAdefl(c(f1,f2,f3), gf[2], dgf[2], Gf[2],
 + method = "adapt", name = c("tanh"), supp = supp, A = A)$COV_W
 > matrix(diag(COV), p, p)
       [,1]  [,2]   [,3]
[1,]  6.974 1.486  4.157
[2,] 27.813 5.163 10.895
[3,]  4.703 1.533  9.879
> COV_est <- ASCOV_FastICAdefl_est(X, gf[2], dgf[2], Gf[2],
+ method="adapt", name=c("tanh"))$COV_W
> matrix(diag(COV_est), p, p)*n
       [,1]  [,2]   [,3]
[1,]  7.219 1.369  4.356
[2,] 39.321 6.399 12.983
[3,]  5.714 1.552 16.177
> apply(simplify2array(Ws1), c(1,2), var)*n
       [,1]  [,2]   [,3]
[1,]  7.128 1.582  4.132
[2,] 29.009 5.334 11.498
[3,]  4.939 1.528 10.639
```

The minimum distance index (Ilmonen et al., 2010) can be used to quantify the separation performance of an unmixing matrix $\widehat{W}$ in a simulation study where the mixing matrix is known. It is defined as

$$\widehat{D} = \frac{1}{\sqrt{p-1}} \inf_{C \in \mathcal{C}} \|C\widehat{W}A - I_p\|,$$

where $\|\cdot\|$ is the Frobenius norm and

$$\mathcal{C} = \{C \ : \ \text{each row and column of } C \text{ has exactly one non-zero element}\}.$$

The minimum distance index takes values between 0 and 1, where zero means perfect separation. The function `ASCOV_FastICAdefl` gives the limiting expected value of $\widehat{D}^2 n(p-1)$, which is independent of the mixing matrix A. Below, the expected value is computed for the optimal extraction order (first the exponential, then the t_9-distribution, and the Gaussian component last), and for the other reasonable order (first the t_9-distribution, then exponential, and the Gaussian last). If the Gaussian component is found first or second, the limiting expected value goes to infinity. We also compute the averages of $\widehat{D}^2 n(p-1)$ over the 1000 repetitions for the three estimates. The average of the reloaded FastICA is close to the expected value of the optimal order, while the other two estimates with a random initial value have larger averages, indicating that the components are often found in non-optimal order.

```
> EMD_opt <- ASCOV_FastICAdefl(c(f1,f2,f3), gf[2], dgf[2], Gf[2],
+ method = "adapt", name = c("tanh"), supp = supp, A = A)$EMD
> EMD_opt
[1] 44.74
> EMD2 <- ASCOV_FastICAdefl(c(f1,f2,f3), gf[2], dgf[2], Gf[2],
+ method = "regular", name = c("tanh"), supp = supp, A = A)$EMD
> EMD2
[1] 67.66
> MD1 <- unlist(lapply(Ws1, MD, A=A))
> MD2 <- unlist(lapply(Ws2, MD, A=A))
> MD3 <- unlist(lapply(Ws3, MD, A=A))
>
> mean(MD1^2)*n*(p-1) # fICA
[1] 46.74
> mean(MD2^2)*n*(p-1) # fastICA
[1] 67.87
> mean(MD3^2)*n*(p-1) # ica
[1] 69.34
```

This simulation shows the effect of the extraction order in deflation-based fastICA. To date, only **fICA** has offered tools to find the optimal extraction order.

Summary

FastICA is the most widely used method to carry out independent component analysis. The R package **fICA** contributes to existing methods in two ways. First, it introduces the reloaded and adaptive deflation-based FastICA and the squared symmetric FastICA. Second, it improves on the existing R functions for classical FastICA algorithms by allowing user-specified nonlinearities and enhancing the convergence of the deflation-based FastICA. We gave a short review of the FastICA estimators, and examples on how to use them with the **fICA** package.

Acknowledgements

We would like to thank the anonymous reviewer for comments. The work of Klaus Nordhausen and Sara Taskinen was supported by CRoNoS COST Action IC1408, COST (European Cooperation in Science and Technology).

Bibliography

J. F. Cardoso. Source separation using higher order moments. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 2109–2112, 1989. doi: 10.1109/ICASSP.1989.266878. [p149]

F. Cayre and T. Furon. *Implementation of FastICA (Independent Component Analysis) for IT++*, 2004. TEMICS Project INRIA/Rennes (IRISA) Campus Universitaire de Beaulieu 35402. [p148]

A. Cichocki and S. Amari. *Adaptive Blind Signal and Image Processing: Learning Algorithms and Applications.* John Wiley & Sons, Cichester, 2002. doi: 10.1002/0470845899. [p148]

P. Comon. Independent component analysis - a new concept? *Signal Processing*, 36:287–314, 1994. doi: 10.1016/0165-1684(94)90029-9. [p148]

P. Comon and C. Jutten. *Handbook of Blind Source Separation. Independent Component Analysis and Applications*. Academic Press, Amsterdam, 2010. [p148]

L. de Lathauwer, B. de Moor, and J. Vandewalle. Fetal electrocardiogram extraction by blind source subspace separation. *IEEE Transactions on Biomedical Engineering*, 47(5):567–572, 2000. doi: 10.1109/10.841326. [p151]

A. Dermoune and T. Wei. FastICA algorithm: Five criteria for the optimal choice of the nonlinearity function. *IEEE Transactions on Signal Processing*, 61(8):2078–2087, 2013. doi: 10.1109/TSP.2013.2243440. [p149]

H. Gävert, J. Hurri, J. Särelä, and A. Hyvärinen. *The FastICA package for MATLAB*, 2005. Lab. of Computer and Information Science, Helsinki University of Technology. [p148]

N. E. Helwig. **ica***: Independent Component Analysis and Projection Pursuit*, 2015. URL `http://CRAN.R-project.org/package=ica`. R package version 1.0-1. [p148]

A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10:626–634, 1999. doi: 10.1109/72.761722. [p148, 149, 150]

A. Hyvärinen and E. Oja. A fast fixed-point algorithm for independent component analyis. *Neural Computation*, 9:1483–1492, 1997. doi: 10.1162/neco.1997.9.7.1483. [p148, 149]

A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. John Wiley & Sons, New York, USA, 2001. [p148]

P. Ilmonen, K. Nordhausen, H. Oja, and E. Ollila. A new performance index for ICA: Properties computation and asymptotic analysis. In V. Vigneron, V. Zarzoso, E. Moreau, R. Gribonval, and E. Vincent, editors, *"Latent Variable Analysis and Signal Separation", LNCS*, volume 6365, pages 229–236, Heidelberg, 2010. Springer. doi: 10.1007/978-3-642-15995-4_29. [p155]

Z. Koldovský and P. Tichavský. Improved variants of the FastICA algorithm. *Advances in Independent Component Analysis and Learning Machines*, 53:53–74, 2015. doi: 10.1016/B978-0-12-802806-3.00002-6. [p148]

J. L. Marchini, C. Heaton, and B. D. Ripley. **fastICA***: FastICA Algorithms to Perform ICA and Projection Pursuit*, 2013. URL `http://CRAN.R-project.org/package=fastICA`. R package version 1.2-0. [p148]

J. Miettinen, K. Nordhausen, H. Oja, and S. Taskinen. Fast equivariant JADE. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 6153–6157, 2013. doi: 10.1109/ICASSP.2013.6638847. [p149]

J. Miettinen, K. Nordhausen, H. Oja, and S. Taskinen. Deflation-based FastICA with adaptive choices of nonlinearities. *IEEE Transactions on Signal Processing*, 62:5716–5724, 2014. doi: 10.1109/TSP.2014.2356442. [p149]

J. Miettinen, S. Taskinen, K. Nordhausen, and H. Oja. Fourth moments and independent component analysis. *Statistical Science*, 30:372–390, 2015. doi: 10.1214/15-STS520. [p150]

J. Miettinen, K. Nordhausen, H. Oja, and S. Taskinen. **BSSasymp***: Asymptotic Covariance Matrices of Some BSS Mixing and Unmixing Matrix Estimates*, 2017a. URL `http://CRAN.R-project.org/package=BSSasymp`. R package version 1.2-1. [p153]

J. Miettinen, K. Nordhausen, H. Oja, and S. Taskinen. **fICA***: Classical, Reloaded and Adaptive FastICA Algorithms*, 2017b. URL `http://CRAN.R-project.org/package=fICA`. R package version 1.1-0. [p148]

J. Miettinen, K. Nordhausen, H. Oja, S. Taskinen, and J. Virta. The squared symmetric FastICA estimator. *Signal Processing*, 131:402–411, 2017c. doi: 10.1016/j.sigpro.2016.08.028. [p149, 150]

J. Miettinen, K. Nordhausen, and S. Taskinen. Blind source separation based on joint diagonalization in R: The packages **JADE** and **BSSasymp**. *Journal of Statistical Software*, 76:1–31, 2017d. doi: 10.18637/jss.v076.i02. [p151]

K. Nordhausen and H. Oja. Independent component analysis: A statistical perspective. *Wiley Interdisciplinary Reviews: Computational Statistics*, page e1440, 2018. doi: 10.1002/wics.1440. [p148]

K. Nordhausen, P. Ilmonen, A. Mandal, and H. Oja. Deflation-based FastICA reloaded. In *European Signal Processing Conference*, pages 1854–1858, 2011. [p149]

E. Ollila. On the robustness of the deflation-based FastICA estimator. In *IEEE Workshop on Statistical Signal Processing*, pages 673–676, 2009. doi: 10.1109/SSP.2009.5278485. [p149]

E. Ollila. The deflation-based FastICA estimator: statistical analysis revisited. *IEEE Transactions on Signal Processing*, 58:1527–1541, 2010. doi: 10.1109/TSP.2009.2036072. [p149]

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, ..., and J. Vanderplas. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2011, 2011. [p148]

J. Virta and K. Nordhausen. On the optimal non-linearities for Gaussian mixtures in FastICA. In P. Tichavský, M. Babaie-Zadeh, O. J. J. Michel, and N. Thirion-Moreau, editors, *Latent Variable Analysis and Signal Separation: 13th International Conference, LVA/ICA 2017, Grenoble, France, February 21-23, 2017, Proceedings*, pages 427–437, Cham, 2017. Springer International Publishing. doi: 10.1007/978-3-319-53547-0_40. [p149]

T. Wei. On the spurious solutions of the FastICA algorithm. In *IEEE Workshop on Statistical Signal Processing*, pages 161–164, 2014. doi: 10.1109/SSP.2014.6884600. [p149]

T. Wei. A convergence and asymptotic analysis of the generalized symmetric FastICA algorithm. *IEEE Transactions on Signal Processing*, 63(24):6445–6458, 2015. doi: 10.1109/TSP.2015.2468686. [p150]

T. Zito, N. Wilbert, L. Wiskott, and P. Berkes. Modular toolkit for data processing (mdp): a python data processing frame work. *Frontiers in Neuroinformatics*, 2, 2008. doi: 10.3389/neuro.11.008.2008. [p148]

Jari Miettinen
Department of Signal Processing and Acoustics
P.O.Box 15400, 20014 Aalto University
Finland
jari.p.miettinen@aalto.fi

Klaus Nordhausen
Institute of Statistics & Mathematical Methods in Economics
Vienna University of Technology Wiedner Hauptstr. 7, 1040 Vienna
Austria
klaus.nordhausen@tuwien.ac.at

Sara Taskinen
Department of Mathematics and Statistics
P.O.Box 35 (MaD), 40014 University of Jyväskylä
Finland
sara.l.taskinen@jyu.fi

Profile Likelihood Estimation of the Correlation Coefficient in the Presence of Left, Right or Interval Censoring and Missing Data

by Yanming Li, Brenda W. Gillespie, Kerby Shedden, John A. Gillespie

Abstract We discuss implementation of a profile likelihood method for estimating a Pearson correlation coefficient from bivariate data with censoring and/or missing values. The method is implemented in an R package **clikcorr** which calculates maximum likelihood estimates of the correlation coefficient when the data are modeled with either a Gaussian or a Student *t*-distribution, in the presence of left, right, or interval censored and/or missing data. The R package includes functions for conducting inference and also provides graphical functions for visualizing the censored data scatter plot and profile log likelihood function. The performance of **clikcorr** in a variety of circumstances is evaluated through extensive simulation studies. We illustrate the package using two dioxin exposure datasets.

Introduction

Partially observed data present a challenge in statistical estimation. Simple approaches like complete case analysis allow traditional estimators to be used, but sacrifice precision and may introduce bias. More sophisticated approaches that incorporate information from partially observed cases have the potential to achieve greater power, but can be much more difficult to implement. As a case in point, the Pearson correlation coefficient is easily estimated by the familiar moment-based formula,

$$r = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{[\Sigma_{i=1}^{n}(x_i - \bar{x})^2 \, \Sigma_{i=1}^{n}(y_i - \bar{y})^2]^{1/2}} \tag{1}$$

and under certain conditions, uncertainty in the estimate can be assessed using Fisher's variance stabilizing transformation (Fisher, 1915). However, it is not obvious how this approach can be extended to accommodate partially observed data.

Partially observed bivariate data (x, y) can take several forms. If either x or y is completely missing, the observation provides no direct information about the correlation coefficient, but is informative about the marginal distributions. If x and/or y is censored, but neither is completely missing, then the observation provides information about the correlation parameter as well as about the marginal distributions.

Here we discuss a likelihood-based method for estimating the correlation coefficient from partially-observed bivariate data, and present an R package implementing the method, **clikcorr** ("Censored data *lik*elihood based *corr*elation estimation") (Li et al., 2016). **clikcorr** calculates maximum likelihood estimates of the parameters in a model, in particular the correlation coefficient, in the presence of left, right, and interval censored data and missing values. **clikcorr** also has functions to help visualize bivariate datasets that contain censored and/or missing values.

The package emphasizes the role of the profile likelihood function for the correlation parameter. It provides functions to visualize the profile likelihood function, and uses likelihood ratio test inversion to provide confidence intervals for the correlation coefficient. This circumvents difficulties that arise when using Wald-type confidence intervals for a parameter that lies in a bounded domain, and that may have an asymmetric sampling distribution.

This project was motivated by the need to calculate correlation coefficients with left-censored chemical assay data. Such data often arise in the context of toxicology, chemistry, environmental science and medical laboratory applications, where some measurements fall below a limit of detection (LOD) (Jaspers et al., 2013). The limit of detection must be known or approximated. Estimating the correlation coefficient from bivariate censored data has been studied (Li et al., 2005), but issues in both methodology and software remain.

The paper is organized as follows. Section 24.1 develops model-based frameworks for correlation estimation based on Gaussian and Student-t models. Section 24.2 describes **clikcorr**, including the data input format, the output format, and program features such as the plotting functions. Section 24.3 presents simulations to address the performance of **clikcorr** under a range of scenarios, focusing on run time and coverage probability of the confidence intervals. In Section 24.4 we use **clikcorr** to analyze dioxin data from both Jaspers et al. (2013) and the National Health and Nutrition Examination Survey (NHANES). Some further details on using the R package **clikcorr** are provided in Section 24.5

Models for correlation parameter estimation

Working in a model-based framework of a bivariate distribution with a correlation parameter opens up several paths for handling partially observed data. A likelihood for the observed data can be calculated by integrating a "complete data" likelihood over all data configurations that are consistent with the observed data. The resulting "observed data likelihood" is seen as preserving full information from the data, and can be used with any standard likelihood-based inference approach, including maximum likelihood (ML) estimation, Wald-type inference, likelihood ratio testing, and profile likelihood analysis.

The bivariate Gaussian distribution is a natural starting point for model-based estimation of a correlation coefficient. The distribution has five parameters: two marginal means (μ_X, μ_Y), two marginal standard deviations (σ_X, σ_Y), and the correlation coefficient ρ_{XY}. The joint density function of the bivariate Gaussian distribution for standardized data ($\mu_x = \mu_y = 0$, $\sigma_x = \sigma_y = 1$) is

$$f_0(x,y) = \frac{1}{2\pi\sqrt{1-\rho_{XY}^2}} \exp\left(-\frac{x^2+y^2-2\rho_{XY}xy}{2(1-\rho_{XY}^2)}\right). \tag{2}$$

When the data are complete, the maximum likelihood estimates of the parameters in the bivariate Gaussian distribution are the sample means ($\bar{x}$ and $\bar{y}$), the sample standard deviations (scaled by $\sqrt{(n-1)/n}$), and the sample correlation coefficient, respectively. These are all closed-form estimators, with the correlation coefficient estimator r given by (1).

The fact that the Gaussian distribution maximum likelihood estimates coincide with the standard method of moments estimates implies some desirable properties. In particular, in the complete data setting, the Gaussian ML estimates will be consistent whenever the law of large numbers can be invoked to give consistency of the sample moments, regardless of whether the data are actually Gaussian. This makes it an appealing foundation for developing an approach that accommodates partially observed cases.

However, inference based on the Gaussian distribution may be misleading if the true distribution is far from Gaussian. In the complete data setting, traditional maximum likelihood (ML) techniques such as likelihood ratio testing and Wald-type inference are seldom performed, with approaches based on Fisher's (Fisher, 1915) variance stabilizing transformation being much more common. An approach to estimation and inference for the correlation parameter in the setting of partially observed data based on Fisher's transformation has been developed (Li et al., 2005). All these approaches may produce misleading results if the data are strongly non-Gaussian, or if the conditions under which Fisher's transformation actually stabilizes the variance do not hold.

The approach we implement here is fully model-based and does not rely on variance stabilizing transformations. Instead, we use standard maximum-likelihood approaches for estimation and inference. The model-based framework discussed here can be used with any model for bivariate data that includes an explicit correlation parameter (including any "elliptic distribution" with density of the form $c \cdot \phi((z-\mu)'\Sigma^{-1}(z-\mu))$). Most such models will involve additional nuisance parameters describing the marginal distributions. Likelihood ratio testing and profile likelihood analysis are especially useful in settings where several nuisance parameters must be estimated along with the parameter of interest.

To explore the sensitivity of the estimates and inferences to the model specification, we also implemented our approach for the setting where the data follow a bivariate Student-t distribution. The density for this distribution when x and y have zero mean, unit variance and degree of freedom d is

$$f_0(x,y;d) = \frac{1}{2\pi\sqrt{1-\rho_{XY}^2}} \left(1+\frac{x^2+y^2-2\rho_{XY}xy}{d(1-\rho_{XY}^2)}\right)^{(d/2+1)}. \tag{3}$$

Note that the parameter ρ_{XY} retains the "product moment" interpretation of the Pearson correlation coefficient – that is, $\rho_{XY} = E[xy]/(\mathrm{SD}(x)\cdot\mathrm{SD}(y))$.

Partially observed data

Our main goal here is to accommodate partially observed data. In the presence of left-, right-, and/or interval-censored data, the univariate likelihood function can be written in terms of density functions (for exact observed values), cumulative distribution functions (CDF, $F_\theta(x) = \mathbb{P}(X \le x)$, for left-censored values), survival functions ($S_\theta(x) = \mathbb{P}(X > x)$, for right-censored values) and differences between cumulative distribution functions ($F_\theta(b) - F_\theta(a)$, for interval-censored values in an interval $(a, b]$). Here θ represents a generic vector of parameters including the correlation coefficient of interest along with any nuisance parameters.

We adopt a convention to treat all three censoring types and exact observed values in a unified representation, with all being special cases of interval censoring. Specifically, every case is known to lie in an interval $(x_i^{lower}, x_i^{upper}]$, with $-\infty < x_i^{lower} = x_i = x_i^{upper} < \infty$ for exact observed cases, $-\infty = x_i^{lower} < x_i^{upper} < \infty$ for left censored cases, $-\infty < x_i^{lower} < x_i^{upper} < \infty$ for interval censored cases, and $-\infty < x_i^{lower} < x_i^{upper} = \infty$ for right censored cases. In addition, we define $-\infty = x_i^{lower} < x_i^{upper} = \infty$ for missing cases; in this case the likelihood contribution is $F(\infty) - F(-\infty) = 1 - 0 = 1$, equivalent to omitting the missing values. However, this convention will be useful to handle missing x or y in the bivariate case. Similar unified representations can be also found in Allison (2010) and Giolo (2004). For example, for k_1 left-censored values with LODs $x_i^{upper}, i = 1, ..., k_1$; followed by $k_2 - k_1$ interval-censored values with censoring interval bounds $(x_i^{lower}, x_i^{upper}], i = k_1 + 1, ..., k_2$; $k_3 - k_2$ right-censored at values $x_i^{lower}, i = k_2 + 1, ..., k_3$; and $n - k_3$ exact values $x_i, i = k_3 + 1, ..., n$, the univariate likelihood function would be

$$L(\theta) = \prod_{i=1}^{k_1} F_\theta(x_i^{upper}) \cdot \prod_{i=k_1+1}^{k_2} [F_\theta(x_i^{upper}) - F_\theta(x_i^{lower})] \cdot \prod_{i=k_2+1}^{k_3} [1 - F_\theta(x_i^{lower})] \cdot \prod_{i=k_3+1}^{n} f_\theta(x_i). \quad (4)$$

For the bivariate setting of (X, Y), extending the likelihood function to accommodate censored data is complicated by the number of types of data pairs to consider. Each of the X and Y variables can be one of the following cases: completely observed, left censored, interval censored, right censored and missing. This yields $5^2 = 25$ types of data pairs. For example, when there are only left-censored and complete data, four cases must be considered: (1) x and y both complete, (2) x left-censored and y complete, (3) x complete and y left-censored, and (4) both x and y left-censored. Each factor in the likelihood (4) is one of these four cases. For case 1, the factor is $f(x_i, y_i)$, as in the complete data situation. For case 2, the factor is $F_{X|Y}(x_i|y_i) \cdot f(y_i)$, where $F_{X|Y}(x|y_i)$ is the conditional distribution function of X, given y_i. For case 3, the factor is $F_{Y|X}(y_i|x_i) \cdot f(x_i)$, where $F_{Y|X}(y|x_i)$ is the conditional distribution function of Y, given x_i. For case 4, the factor is $F(x_i, y_i) = P(X \le x_i, Y \le y_i)$, i.e., the bivariate cumulative distribution function evaluated at (x_i, y_i).

In the Gaussian model, for cases 2 and 3 we can take advantage of the fact that the conditional distributions are also Gaussian, with means and variances that are easily calculated (Rao, 1973). Thus we calculate $F_{X|Y}(x_i|y_i) \cdot f(y_i)$ rather than $\mathbb{P}(X \le x_i, Y = y_i)$, because the former eases the calculation by reducing the dimension of the distribution functions from bivariate to univariate. This method can be applied to right-censored data in a similar way. In case 4, because $F(x_i, y_i)$ does not have a closed form expression, it must be evaluated by integration of the joint density over a 2-dimensional region in the bivariate domain. Efficient methods for this calculation are provided by the `mvtnorm` R package.

Besides censoring, missing data can also be present in bivariate data (Little and Rubin, 2002). When both members of a pair are missing, the likelihood contribution factor is 1, and these observations can be omitted from the analysis. When one member of a pair is missing, the observed member of the pair can be used to improve estimation of parameters describing its marginal distribution.

The likelihood function $L(\theta)$ is the product of n factors, one per observation pair, with the form of each term depending on whether members of the pair are observed, censored or missing. Table 1 lists all the possible cases of such pairs and the corresponding factor of the likelihood function in the bivariate normal case. We use the conditional distribution whenever possible to make the calculations easier.

Alternative models

When data arise from an underlying heavy tailed bivariate distribution, estimates based on the bivariate normal model may yield confidence intervals with poor coverage probabilities. This is especially a concern when partially observed data are present. We provide the bivariate Student-t model as an alternative approach to estimation in this setting.

The bivariate t distribution does not have the useful property held by the bivariate normal distribution that the conditional distributions have the same distributional form as the marginal distributions. Consequently, we do not have a simple expression for the conditional distributions of the bivariate t distribution. Instead, we construct the likelihood through numerical integration over certain regions of the bivariate domain for each likelihood factor. Specifically, the likelihood factors

Pair type	Factor to the likelihood	Pair type	Factor to the likelihood
X o., Y o.	$f(x_i, y_i)$	*X o., Y lc.*	$F_{Y\|X}(y_i^u\|x_i) * f_X(x_i)$
X o., Y rc.	$[1 - F_{Y\|X}(y_i^l\|x_i)] * f_X(x_i)$	*X o., Y ic.*	$[F_{Y\|X}(y_i^u\|x_i) - F_{Y\|X}(y_i^l\|x_i)] * f_X(x_i)$
X o., Y m.	$f_X(x_i)$	*X lc., Y o.*	$F_{X\|Y}(x_i^u\|y_i) * f_Y(y_i)$
X lc., Y lc.	$F(x_i^u, y_i^u)$	*X lc., Y rc.*	$\mathbb{P}(X \leq x_i^u, Y > y_i^l)$
X lc., Y ic.	$\mathbb{P}(X \leq x_i^u, y_i^l < Y \leq y_i^u)$	*X lc., Y m.*	$F_X(x_i^u)$
X rc., Y o.	$[1 - F_{X\|Y}(x_i^l\|y_i)] * f_Y(y_i)$	*X rc., Y lc.*	$\mathbb{P}(X > x_i^l, Y \leq y_i^u)$
X rc., Y rc.	$\mathbb{P}(X > x_i^l, Y > y_i^l)$	*X rc., Y ic.*	$\mathbb{P}(X > x_i^l, y_i^l < Y \leq y_i^u)$
X rc., Y m.	$1 - F_X(x_i^l)$	*X ic., Y o.*	$[F_{X\|Y}(x_i^u\|y_i) - F_{X\|Y}(x_i^l\|y_i)] * f_Y(y_i)$
X ic., Y lc.	$\mathbb{P}(x_i^l < X \leq x_i^u, Y \leq y_i^u)$	*X ic., Y rc.*	$\mathbb{P}(x_i^l < X \leq x_i^u, Y > y_i^l)$
X ic., Y ic.	$\mathbb{P}(x_i^l < X \leq x_i^u, y_i^l < Y \leq y_i^u)$	*X ic., Y m.*	$F_X(x_i^u) - F_X(x_i^l)$
X m., Y o.	$f_Y(y_i)$	*X m., Y lc.*	$F_Y(y_i^u)$
X m., Y rc.	$1 - F_Y(y_i^l)$	*X m., Y ic.*	$F_Y(y_i^u) - F_Y(y_i^l)$
X m., Y m.	1		

Table 1: Contributing factors to the likelihood function for each case of observed, censored or missing values in pairs from the bivariate normal distribution. "o." = observed; "lc." = left censored; "rc." =right censored; "ic." = interval censored; "m." = missing. $x_i^l = x_i^{lower}$; $x_i^u = x_i^{upper}$; $y_i^l = y_i^{lower}$; $y_i^u = y_i^{upper}$.

are given by

$$\int_{x_i^{lower}}^{x_i^{upper}} \int_{y_i^{lower}}^{y_i^{upper}} f(\mu, \nu) d\mu d\nu, \tag{5}$$

where $f(x, y)$ is the bivariate t density, and $(x_i^{lower}, x_i^{upper})$ are the censoring intervals as defined in Section 24.1.2. Table 2 lists the likelihood contribution for each case. We use the `mvtnorm` package in R to approximate these values.

Pair type	Factor to the likelihood	Pair type	Factor to the likelihood
X o., Y o.	$f(x_i, y_i)$	*X o., Y lc.*	$\int_{-\infty}^{y_i^u} f(x_i, \nu) d\nu$
X o., Y rc.	$\int_{y_i^l}^{\infty} f(x_i, \nu) d\nu$	*X o., Y ic.*	$\int_{y_i^l}^{y_i^u} f(x_i, \nu) d\nu$
X o., Y m.	$f_X(x_i)$	*X lc., Y o.*	$\int_{-\infty}^{x_i^u} f(\mu, y_i) d\mu$
X lc., Y lc.	$\int_{-\infty}^{y_i^u} \int_{-\infty}^{x_i^u} f(\mu, \nu) d\mu d\nu$	*X lc., Y rc.*	$\int_{y_i^l}^{\infty} \int_{-\infty}^{x_i^u} f(\mu, \nu) d\mu d\nu$
X lc., Y ic.	$\int_{y_i^l}^{y_i^u} \int_{-\infty}^{x_i^u} f(\mu, \nu) d\mu d\nu$	*X lc., Y m.*	$F_X(x_i^u)$
X rc., Y o.	$\int_{x_i^l}^{\infty} f(\mu, y_i) d\mu$	*X rc., Y lc.*	$\int_{-\infty}^{y_i^u} \int_{x_i^l}^{\infty} f(\mu, \nu) d\mu d\nu$
X rc., Y rc.	$\int_{y_i^l}^{\infty} \int_{x_i^u}^{\infty} f(\mu, \nu) d\mu d\nu$	*X rc., Y ic.*	$\int_{y_i^l}^{y_i^u} \int_{x_i^l}^{\infty} f(\mu, \nu) d\mu d\nu$
X rc., Y m.	$1 - F_X(x_i^l)$	*X ic., Y o.*	$\int_{x_i^l}^{x_i^u} f(\mu, y_i) d\mu$
X ic., Y lc.	$\int_{-\infty}^{y_i^u} \int_{x_i^l}^{x_i^u} f(\mu, \nu) d\mu d\nu$	*X ic., Y rc.*	$\int_{y_i^l}^{\infty} \int_{x_i^l}^{x_i^u} f(\mu, \nu) d\mu d\nu$
X ic., Y ic.	$\int_{y_i^l}^{y_i^u} \int_{x_i^l}^{x_i^u} f(\mu, \nu) d\mu d\nu$	*X ic., Y m.*	$F_X(x_i^u) - F_X(x_i^l)$
X m., Y o.	$f_Y(y_i)$	*X m., Y lc.*	$F_Y(y_i^u)$
X m., Y rc.	$1 - F_Y(y_i^l)$	*X m., Y ic.*	$F_Y(y_i^u) - F_Y(y_i^l)$
X m., Y m.	1		

Table 2: Contributing factors to the likelihood function for each case of observed, censored or missing values in pairs from the bivariate t-distribution. "o." = observed; "lc." = left censored; "rc." =right censored; "ic." = interval censored; "m." = missing. $x_i^l = x_i^{lower}$; $x_i^u = x_i^{upper}$; $y_i^l = y_i^{lower}$; $y_i^u = y_i^{upper}$.

Inference

For large n, the likelihood ratio (LR) test for the null hypothesis $\rho_{XY} = \rho_0$ is obtained by evaluating the log profile likelihood function at the maximum likelihood estimator (MLE) and at a given value

ρ_0, taking twice the (positive) difference between them, and comparing to a chi-squared distribution with 1 degree of freedom (Murphy and van der Vaart, 2000). The p-value for the test is the chi-squared probability of a value as or more extreme as the one obtained. A profile likelihood-based confidence interval for ρ_{XY} is obtained by inverting the likelihood ratio test, i.e., by finding the values (ρ_L, ρ_U) above and below the profile MLE for which the LR test would first reject (at significance level α) the null hypothesis that $\rho_{XY} = \rho_L$, and similarly ρ_U. Specifically, the confidence interval with coverage probability $1 - \alpha$ is constructed to be the set

$$\left\{\rho : -2[\ell(\rho) - \max_{\rho} \ell(\rho)] < q_{\chi_1^2}(1-\alpha)\right\},$$

where $\ell(\rho) = \max_{\mu_X,\mu_Y,\sigma_X,\sigma_Y} \ell(\mu_X, \mu_Y, \sigma_X, \sigma_Y, \rho)$ is the profile log likelihood for ρ and $q_{\chi_1^2}(1-\alpha)$ is the $1-\alpha$ upper critical value of the chi-squared distribution with 1 degree of freedom (Venzon and Moolgavkar, 1988; Stryhn and Christensen, 2003).

Profile likelihood based confidence intervals have some advantages over other methods such as the commonly used Wald-type confidence intervals obtained from the limiting distribution of the MLE. Wald-type intervals may give poor performance when the parameter is close to the boundary of its domain, in which case the parameter estimate assumes a skewed distribution (Stryhn and Christensen, 2003). The profile likelihood based approach is robust to this skewness by providing an asymmetric confidence interval estimate. Zhou et al. (2012) compared confidence interval estimates for a logistic regression setting using (1) the Wald method, (2) the Bootstrap method and (3) the profile likelihood based method. In our setting, the profile likelihood based method is more computationally efficient than the bootstrap method and is more robust to cases of parameter estimates with markedly skewed distributions than the Wald method.

An R package, clikcorr

We implemented an R package **clikcorr** (Li et al., 2016) to facilitate inference on the correlation coefficient from bivariate data in the presence of censored and/or missing values. The package is able to handle all combinations of observed, censored (left-, right-, or interval-censored) or missing data on each of the bivariate components. Most of the currently available software for estimating correlation coefficient between bivariate censored data focus on the left-truncated and right-censored data (Li et al., 2005; Schemper et al., 2013). To the best of our knowledge, **clikcorr** is the first available software that can handle data with a complex mixture types of censoring and missing. The package also has graphical functions to help visualize bivariate censored data and the shape of the profile log likelihood function.

We note the large set of routines for left-censored data given in the NADA (Non-detects and Data Analysis) package (Helsel, 2012, 2005). NADA currently has no function for the Pearson correlation coefficient with left-censored data, although it does include a function for Kendall's τ. The implementation described here is not currently included in NADA.

clikcorr requires a special input data format. Each input variable is represented by two columns for the interval bounds discussed in section 24.1.3, with $\pm\infty$ replaced by "NA" values. The same input data format is also adopted in the `LIFEREG` procedure of `SAS` software (Allison, 2010) and in the "`Surv`" class of the **survival** package (Therneau, 2015) in R when `type=interval2`. Table 3 gives an example dataset formatted for input into **clikcorr**, with a pair of variables, Variable1 and Variable2, each given with lower and upper interval endpoints. For the first sample ID, both variables are exactly observed at values 10.9 and 37.6 respectively. For the second to the fifth samples, Variable1 is exactly observed but Variable2 is, in order, left-censored at 7.6, right censored at 26.7, interval-censored at $(11.7, 20.9)$ and missing. For sample ID 6, both variables are left censored, and for sample ID 7, both variables are interval censored. **clikcorr** functions `rightLeftC2DF` and `intervalC2DF` can transform input data from "`Surv`" class formats into the **clikcorr** required input format. `rightLeftC2DF` transforms the right- or left- censored data in the "`Surv`" class and "`intervalC2DF`" transforms interval-censored data (either `interval` or `interval2`) in the "`Surv`" class.

clikcorr has three main functions, one estimating function and two graphical functions. The estimating function calculates the correlation coefficient estimate, its confidence interval and the p-value from the likelihood ratio test of the null hypothesis that the underlying true correlation coefficient is zero. The user can specify either a bivariate normal (the default) or a bivariate t distribution to use for inference.

Below is an example of calling the estimation function and the output assuming (by default) the bivariate normal distribution. Here `ND` is an example dataset contained in the **clikcorr** package. See Li et al. (2016) for details about the `ND` dataset. (`"t1_OCDD"`, `"t1_HxCDF_234678"`) and (`"t2_OCDD"`, `"t2_HxCDF_234678"`) are column names for the lower and upper interval bounds for input variables

Sample ID	Lower bound of the 1st variable	Upper bound of the 1st variable	Lower bound of the 2nd variable	Upper bound of the 2nd variable
1	10.9	10.9	37.6	37.6
2	12.8	12.8	NA	7.6
3	8.7	8.7	26.7	NA
4	13.2	13.2	11.7	20.9
5	9.8	9.8	NA	NA
6	NA	10.0	NA	6.9
7	11.4	16.3	10.8	18.7

Table 3: An example input data frame for **clikcorr** with two input variables.

"OCDD" and "HxCDF_234678", respectively . cp is the user specified coverage probability (confidence level) of the confidence interval, with the default set to 0.95. The print method of "clikcorr" class outputs the coefficients matrix, which contains the estimated correlation coefficient coefficients, the lower and upper bounds for the confidence interval CI.lower and CI.upper, and the p-value, p.value, from the likelihood ratio test of the null hypothesis that $\rho_{XY} = 0$.

```
R> data(ND)
R> logND <- log(ND)
R> clikcorr(data = logND, lower1 = "t1_OCDD", upper1 = "t2_OCDD",
   + lower2 = "t1_HxCDF_234678", upper2 = "t2_HxCDF_234678", cp=.95)

        Call:
        clikcorr.default(data = logND, lower1 = "t1_OCDD", upper1 = "t2_OCDD",
        lower2 = "t1_HxCDF_234678", upper2 = "t2_HxCDF_234678", cp = 0.95)

        coefficients  95%CI.lower 95 %CI.upper      p.value
         0.472657898 -0.006538307  0.769620179  0.053083585
```

The summary method further outputs the vector of estimated means $Mean, the estimated variance-covariance matrix $Cov and the log likelihood value at the MLE $Loglike.

```
R> obj <- clikcorr(data=logND, lower1="t1_OCDD", upper1="t2_OCDD",
   + lower2="t1_HxCDF_234678", upper2="t2_HxCDF_234678", cp=.95)
R> summary(obj)

        $call
        clikcorr.default(data = logND, lower1 = "t1_OCDD", upper1 = "t2_OCDD",
            lower2 = "t1_HxCDF_234678", upper2 = "t2_HxCDF_234678", cp = 0.95)

        $coefficients
        coefficients  95%CI.lower  95%CI.upper      p.value
         0.472657898 -0.006538307  0.769620179  0.053083585

        $mean
        [1]  5.3706036 -0.4655811

        $Cov
                  [,1]      [,2]
        [1,] 0.5938656 0.3100190
        [2,] 0.3100190 0.7244269

        $loglik
        [1] -37.97112

        attr(,"class")
        [1] "summary.clikcorr"
```

As we will illustrate in a later section, when data are generated from a heavy tailed distribution, inference assuming the bivariate normal distribution can result in poor performance. The option "dist=t" can be used to specify a bivariate t distribution. The option "df" specifies the degrees of freedom (d.f.) of the bivariate t distribution and by default is set to 4. Smaller d.f. give t distributions with heavier tails; larger d.f. (e.g., >30) give *t* distributions that are similar to the normal distribution. Below is an example of calling the estimation function using dataset ND, and the output assuming the bivariate *t* distribution. The variable names are the same as given above.

```
R> obj <- clikcorr(ND, lower1="t1_OCDD", upper1="t2_OCDD", lower2="t1_HxCDF_234678",
   + upper2="t2_HxCDF_234678", dist="t", df=10)
R> summary(obj)

        $call
        clikcorr.default(data = logND, lower1 = "t1_OCDD", upper1 = "t2_OCDD",
            lower2 = "t1_HxCDF_234678", upper2 = "t2_HxCDF_234678", dist = "t",
            df = 10, nlm = TRUE)

        $coefficients
        coefficients  95%CI.lower  95%CI.upper      p.value
          0.77229746  -0.09201948   0.72893732   0.10812589

        $mean
        [1]  5.5240929 -0.3338772

        $Cov
                  [,1]      [,2]
        [1,] 0.5996139 0.4886447
        [2,] 0.4886447 0.6676448

        $loglik
        [1] -104.0848

        attr(,"class")
        [1] "summary.clikcorr"
```

clikcorr also provides two graphical functions. Visualizing the data and the analytic outputs is often useful, and is sometimes invaluable for understanding the results. A barrier to graphing censored data is finding a way to plot the range of possible data values. The function splot provides an effective way of visualizing bivariate censored and/or missing data. With splot, censored values can be identified with different colors or plotting symbols, depending on the data types, as described in sections 24.1.2 and 24.1.3. For example, the plotting symbol could be an arrow showing the direction of uncertainty. Since the ND dataset in **clikcorr** package only contains right censored data points, to demonstrate the fact that **clikcorr** can provide a visual representation for all types of censored and/or missing data, in the following examples, we illustrate the **clikcorr** plotting functions using a simulated dataset. Denote by SimDat an input data matrix, generated from a tri-variate normal distribution with pairwise correlation coefficient 0.5 and with missing values and all types of censoring randomly introduced. Let "L1" and "U1" be column names for the lower and upper interval bounds for input variable 1. Similar notations are also adopted for variable 2 and variable 3. A single scatterplot of Variable 2 versus Variable 1, or ("L2", "U2") versus ("L1", "U1"), would be plotted by calling splot2(SimDat, c("L1","L2"), c("U1","U2"). The following syntax calls splot and produces a scatterplot matrix with all combinations of three input variables, as shown in Figure 1. Note that a censored observation has its position indicated by the tail of an arrow and the head of the arrow is pointed to the censoring direction.

```
R> splot(SimDat, c("L1", "L2", "L3"), c("U1", "U2", "U3"))
```

The S3 function plot of the clikcorr class produces a plot of the profile log likelihood function. The following syntax calls the plot function using the bivariate normal and the bivariate $t-$distributions respectively. The output profile plots are given in Figure 2, and illustrate that specifying the appropriate distribution is important for both optimum point and confidence interval estimation.

```
R> modN <- clikcorr(SimDat, "L1", "U1", "L2", "U2", cp=.95)
R> plot(modN)
R> modT <- clikcorr(SimDat, "L1", "U1", "L2", "U2", dist="t", df=3)
R> plot(modT)
```

The **clikcorr** main estimating function requires that the following other R functions be loaded: the R package **mvtnorm** (Genz and Bretz, 2009; Genz et al., 2012; Hothorn et al., 2013), as well as function "optim", and function "integrate" when using the bivariate $t-$distribution.

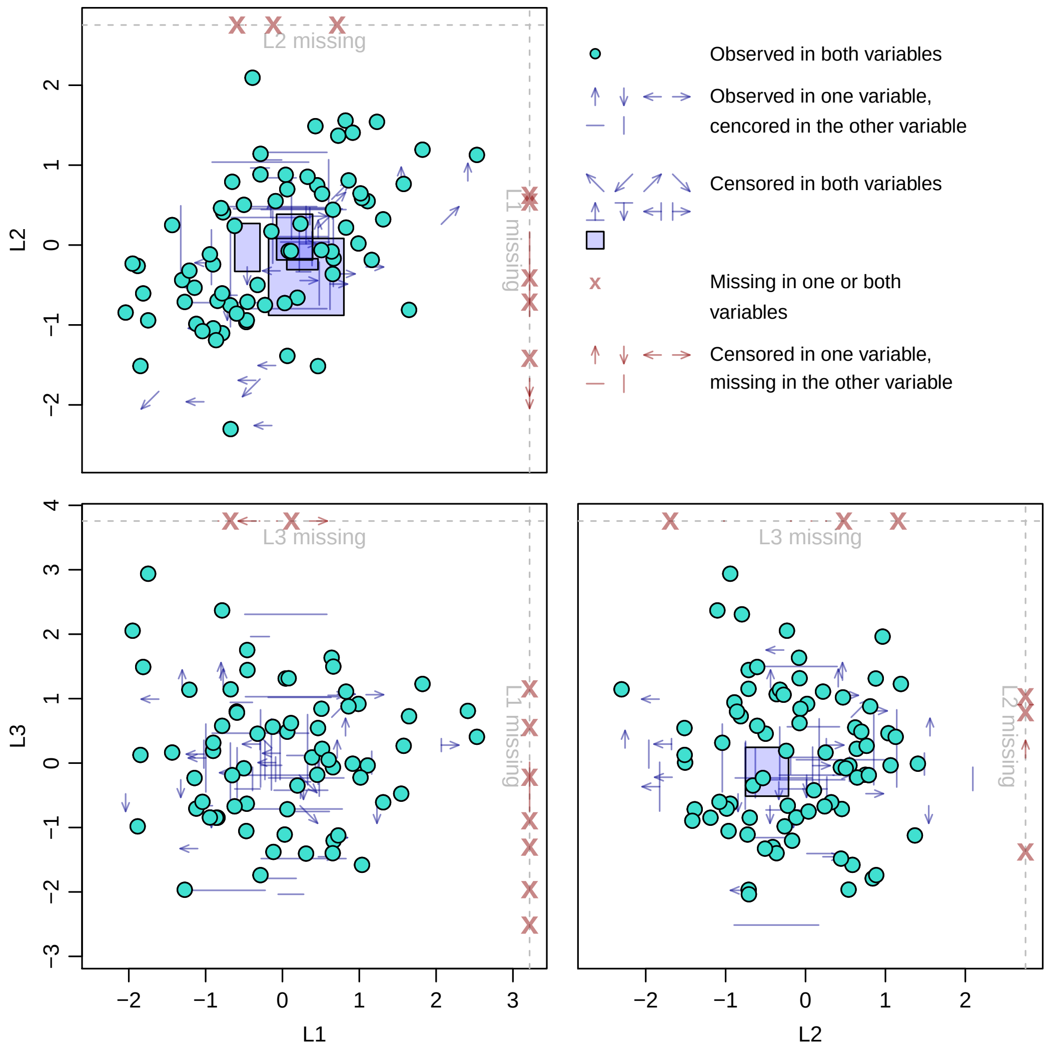


Figure 1: Example scatter plots from a simulated data frame.

Simulation studies

Extensive simulations have been done to demonstrate different aspects of the performance of **clikcorr** under various settings of sample size, percent censoring, and underlying distribution. We show these results in the following subsections.

Coverage probability of the confidence interval

First, we investigate the coverage probability of the confidence intervals with two types of right censoring, termed "fixed point" and "random", as described further below. Data were generated from the bivariate normal distribution, and inference assumed the same distribution.

In the fixed point right censoring setting, all observations were censored at the same value. Different fixed points were calculated to give expected censoring rates of 0%, 25% and 75%. For the random right censoring setting, values were censored at randomly generated thresholds, which were generated from normal distributions with means set at different values to produce data with specific expected censoring proportions. Tables 4 and 5 give the simulated coverage probabilities for different settings of sample sizes and censoring rates. In the completely observed cases in Table 4, coverage probabilities calculated using the Fisher transformation rather than the profile likelihood approach are given in parentheses for comparison.

It can be seen from Tables 4 and 5 that **clikcorr** gives satisfying confidence interval coverage probabilities in both the fixed and random censoring settings. In the completely observed cases, coverage probabilities for both profile likelihood and Fisher transformation based confidence intervals

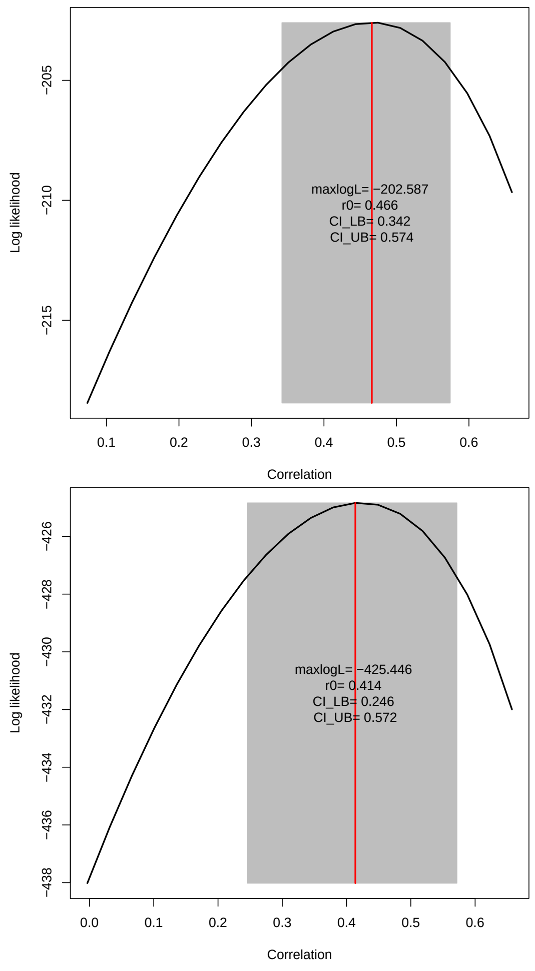


Figure 2: Example of profile likelihood plots. Input data are simulated from the bivariate normal distribution with true $\rho_{XY} = 0.5$. The upper panel shows the profile likelihood using the bivariate normal distribution; the lower panel shows the profile likelihood using the bivariate $t-$distribution.

are very close to the nominal level of 0.95

Run time, bias and mean squared error (MSE)

Table 6 shows the run time of **clikcorr** under different settings of sample sizes and censoring proportion configurations, using the bivariate normal distribution for both data generation and inference. The censoring proportion configurations, for example "$X15\%, XY15\%, Y15\%$" means that 15% of the observations are censored only for X, 15% are censored for both X and Y, and 15% are censored only for Y. It can be seen that longer run time is needed for larger sample size, larger proportion with both X and Y being censored, or larger magnitude of the correlation. A typical run time ranges from a few seconds to a few minutes.

For the bivariate t distributions, the run time is generally slower than for the bivariate normal distributions in the same setting. The reason for this is that the bivariate t setting requires numerical integration to calculate the joint distribution over a 2-dimensional domain as illustrated in (5) in the case of either X or Y censored, whereas the analogous calculation for the bivariate normal setting has a closed form.

Table 7 shows the estimates of the bias and MSE for inference on ρ using the bivariate normal distribution for data generation and inference, and with random right censoring. Essentially, **clikcorr** provides an unbiased estimator of the correlation coefficient.

Gaussian versus Student t models

When the underlying marginal distributions are heavy tailed rather than normal, basing inference on the bivariate normal can result in poor confidence interval coverage probabilities. This is illustrated in Table 8, where the data were generated from correlated marginal *t* distributions and the correlation coefficients were estimated using the bivariate normal distribution. When the degrees of freedom of the underlying marginal *t* distributions are small ($\sim$ 3 to 5) or the true underlying correlation coefficient is very high ($\sim$ 0.9), the confidence intervals have poor coverage probabilities ($\sim$ 60% to 80%).

The coverage probabilities within the parentheses in Table 8 are for confidence intervals constructed using the bivariate *t* distributions. Compared to inference based on the bivariate normal distribution, using the bivariate *t* distribution provides much better coverage probabilities in settings with very heavy-tailed distributions and high correlations.

The **clikcorr** package does not provide a way to test whether the data are drawn from a heavy tailed distribution or not, nor does it estimate the best value of d.f. to use. The users could refer to Bryson (1974); Resnick (2007) or use QQ plots to guide their decision on which model to use.

Dioxin exposure datasets

In this section, we used **clikcorr** to analyze two real world datasets and compared the results to conventional approaches.

Owl feather data

The first dataset was analyzed in (Jaspers et al., 2013) which used an early version of **clikcorr** to estimate correlations for left-censored (below detection) data. The data contain Perfluorooctane sulfonate (PFOS) substances in feathers and soft tissues (liver, muscle, adipose tissue and preen glands) of barn owl road-kill victims collected in the province of Antwerp, Belgium. Figure 3 gives an example scatter plot (upper panel) and an example plot of the profile log likelihood of the correlation between PFOS in liver tissue and feathers using **clikcorr**. Figure 4 shows the scatter plot matrix across tail feather and the tissues generated by **clikcorr**. The asymmetrical shape of the profile likelihood reflects the asymmetrical sampling distribution, and leads to a confidence interval that is longer to the left than to the right of the point estimate.

NHANES data

In the second analysis using **clikcorr**, we analyzed dioxin data from the 2002-2003 National Health and Nutrition Examination Survey (NHANES). NHANES is one of the most widely-used datasets describing the health and nutritional status of people residing in the US. The data we analyzed contained 22 chemicals, including 7 dioxins, 9 furans and 6 polychlorinated biphenyls (PCBs) across 1643 individuals. Correlations between these chemicals are common, and are of interest for several reasons. First, dioxins with high correlations measured in the environment may arise from the same or similar sources. In addition, levels of two chemicals in the body that have high correlations may have similar pathways for entering the body. For two chemicals with high correlations, it may be sufficient to monitor one of them (to reduce cost), as was the goal in Jaspers' barn owl example. Finally, exposure to pairs of correlated chemicals may confound an investigation of health effects, since distinguishing the individual effects of each chemical would be difficult. Therefore estimating correlations between those compounds is of scientific interest. The chemicals in our dioxin data are all subject to detection limits, and have a wide range of percentage of left-censored values. It is important to incorporate information from partially observed cases in the data to provide accurate estimation and inference results on the correlations between the compounds.

The pairwise correlation coefficients calculated from **clikcorr** are given in upper-right triangle of Table 9. For each pair, the MLE of the correlation coefficient and associated LRT *p*-value (in parentheses) are calculated assuming the bivariate normal distribution. The highly correlated pairs (with estimated correlation coefficient $\geq$ .80) are in boldface type. Chemicals within each group (dioxin, furan or PCB) tend to be more highly correlated than chemicals from different groups. Abbreviated names of each chemical were used in Table 9. The full names and the detail information about each chemical can be found at Agency for Toxic Substances and Disease Registry (ATSDR) (`http://www.atsdr.cdc.gov/substances`), which is part of the U.S. Centers for Disease Control and Prevention. The chemical indices and names are corresponding as following: for dioxind, d1=*TCDD*; d2=*PeCDD*; d3=*HxCDD123478*; d4=*HxCDD123678*; d5=*HxCDD123789*;

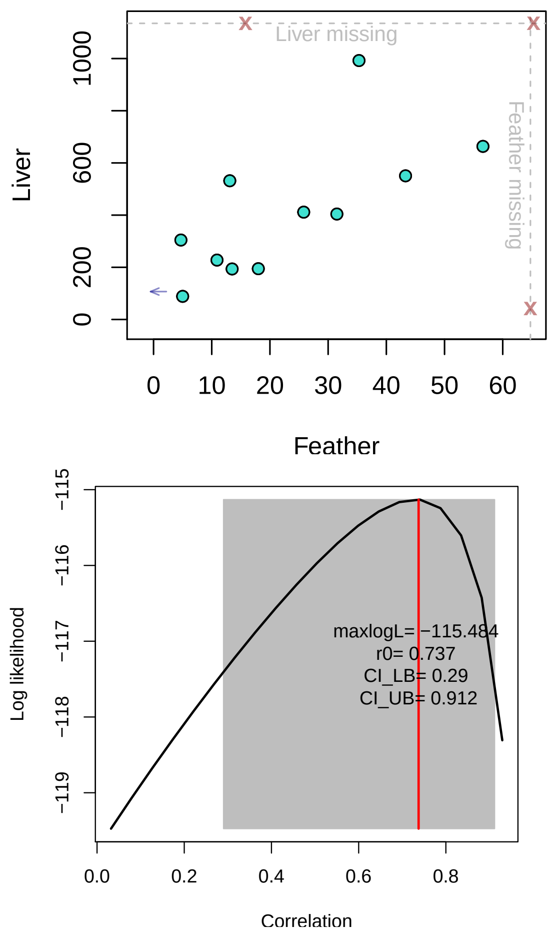


Figure 3: Output graphics from analyzing the owl feather data using **clikcorr**. Upper panel: scatter plot of PFOS in liver tissue vs. feathers; Lower panel: profile log likelihood plot for the correlation coefficient between PFOS concentration in liver and in feathers.

d6=*HpCDD*; d7=*OCDD*. For furan, f1=*TCDF*; f2=*PeCDF12378*; f3=*PeCDF23478*; f4=*HxCDF123478*; f5=*HxCDF123678*; f6=*HxCDF234678*; f7=*HpCDF1234678*; f8=*HpCDF1234789*; f9=*OCDF*. For PCB, p1=*PCB105*; p2=*PCB118*; p3=*PCB156*; p4=*PCB157*; p5=*PCB167*; p6=*PCB189*. p-values less than 0.01 are recoded as 0's in Table 9.

The pairs of chemicals with the highest correlation in each group are shown in Figure 6. Both the MLE of the correlation coefficient calculated from **clikcorr** $\hat{\rho}_{clikcorr}$, and the Pearson correlation coefficients calculated from only the complete cases $\hat{\rho}_{complete}$, are given, along with the complete case sample size, $n_{complete}$. When the proportion of pairs with censored values is low, $\hat{\rho}_{clikcorr}$ and $\hat{\rho}_{complete}$ are very similar; on the other hand, when the proportion of pairs with censored values (in either or both variables) is high, the two correlation coefficients are different since $\hat{\rho}_{clikcorr}$ can utilize the extra information from the censored data.

Some chemicals in the dataset appear to have highly skewed marginal distributions. It is common to apply a symmetrization transformation in such a setting. The correlation coefficient between transformed values estimates a different population parameter, but this parameter may be of as much or greater inference than the correlation computed on the original scale. To demonstrate the effect of such a transformation, we log-transformed the data for each chemical and recalculated the correlation coefficient using the bivariate normal model. The estimation and inference results from the transformed data are given in the lower-left triangle in Table 9.

Using the bivariate normal model for transformed data is an example of using bivariate Gaussian copulas to allow a wide range of distributions to be analyzed using software developed for Gaussian models. A general approach of this type would be to transform X using $F^{-1} \circ \hat{F}_x$, where $\hat{F}_x$ is the empirical CDF of X and F is the standard Gaussian CDF (Y would be transformed analogously). This approach is quiet flexible, and would allow **clikcorr** to be used for correlation parameter inference in a wide range of settings. One limitation of this approach is that the marginal transformations

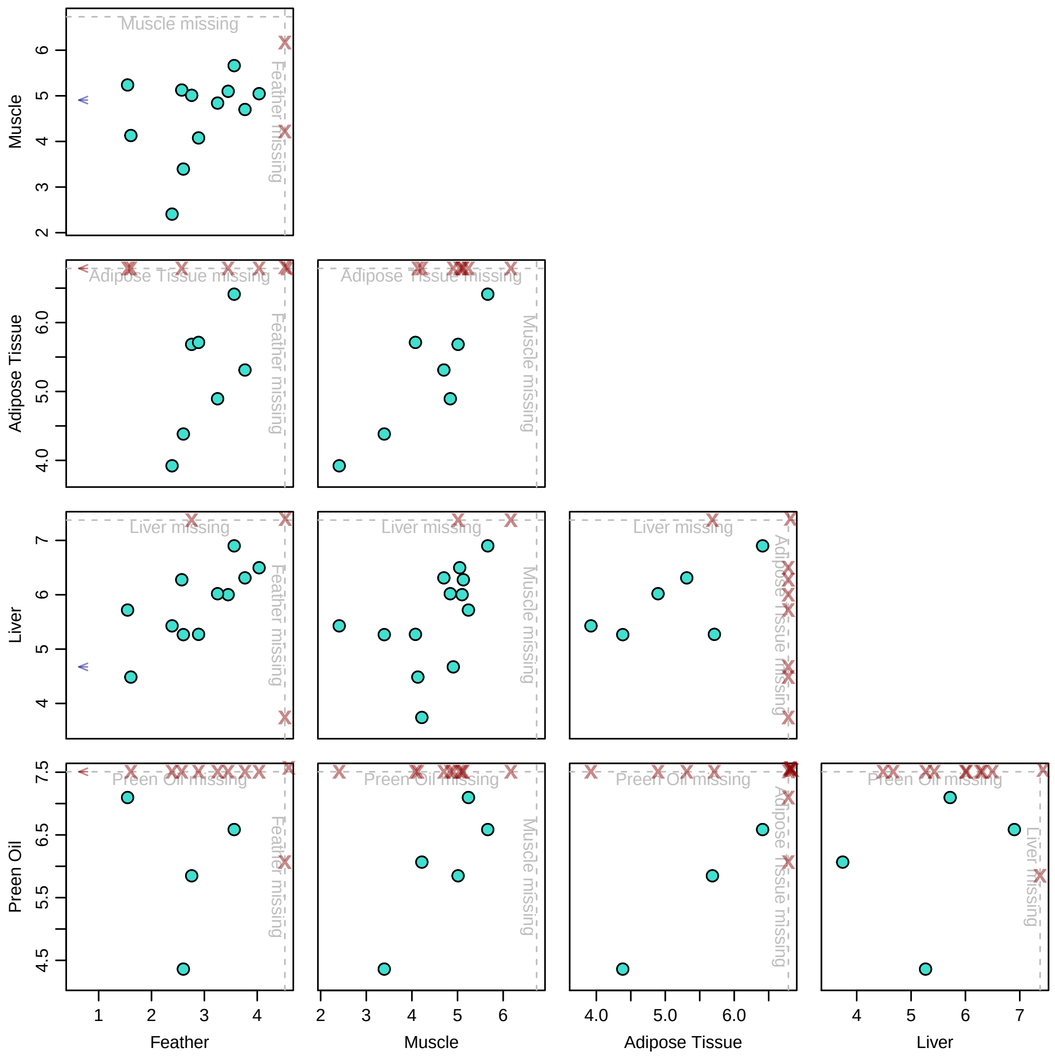


Figure 4: Scatter plot matrix from analyzing the owl feather data using **clikcorr**.

are estimated from the data, but the uncertainty in this estimation is not propagated through to the estimation of the correlation parameter. This would mainly be a concern with small datasets.

Figure 5 demonstrates the difference between using data on the original and the log-transformed scales. When the observed marginal distributions are highly skewed, assuming a bivariate normal true distribution could lead to false results. For example, the correlation effect could be leveraged away by the "outlier" points on tail of either marginal distributions. As in Figure 5 (c), the original values between chemicals *OCDD*(d7) and *HxCDF234678* (f6) were not significantly correlated due to the divergent effect of the points on the marginal tails. While after the log transformation, the marginal sample distributions are more normally shaped and thus make the bivariate normal assumption more likely to be true, and the two chemicals are significantly correlated with a moderate correlation coefficient.

Further details on clikcorr

This section gives technical details regarding function maximization and gives some examples of additional plotting options.

We note that **clikcorr** depends on the R function `optim`, which is used to find the maximum of the log likelihood, or profile log likelihood. For some starting values of μ_X, μ_Y, σ_X, σ_Y and ρ_{XY}, the procedure gives an error message "`function cannot be evaluated at initial parameters`". In such cases, manually adjusting the initial parameter values is recommended.

Figure 7 illustrates two example profile log likelihood functions for parameters of the bivariate distribution of PFOS in Tail feather and Adipose tissue in the owl feather dataset. Each plot shows the log likelihood for certain parameters with the other parameters fixed. In the upper panel, the log likelihood of $(\log(\sigma_X^2), \log(\sigma_Y^2))$ remains at $-\infty$ over the lower left triangular region of the plot. So if the starting values of σ_X and σ_Y are set such that the pair $(\log(\sigma_X^2), \log(\sigma_Y^2))$ falls in that region, then the value of the joint log likelihood will be trapped in the region and the above error message will show up. In the lower panel graph, the log likelihood of the covariance drops steeply when the covariance parameter value is greater than 2000. Therefore, if the starting value of ρ_{XY} is set such that the starting covariance is much larger than 2000, say 5000, then the profile log likelihood might underflow the limit of a floating number adopted in the "`optim`" function and give the above error message.

It is worth the effort to make the input starting parameter values as close to the true parameters as possible, if these are known. By default, **clikcorr** uses the sample means, the log transform of sample variances and the sample correlation based on the complete observations as the starting values. It also allows the user to specify the starting values and pass them to the "`optim`" function by using the option `sv`. This option is especially useful when the number of exact observations is small. For example, if a dataset contains only one complete observation, then the starting values of $(\log(\sigma_X^2), \log(\sigma_Y^2))$ can not be effectively estimated (both will be estimated as $-\infty$). In such cases, it is suggested that users make their own reasonable guesses on the starting parameter values rather than using the default. An example is given in the following syntax:

```
R> clikcorr(owlData, "feather_L", "feather_U", "AT_L", "AT_U", cp=.95, dist=t,
   + df=5, sv=c(23, 240, 6, 1300, 12))
```

The default optimization method for searching for the MLE in **clikcorr** is the default method used in "`optim`", which is the Nelder-Mead method (R Core Team, 2013b). **clikcorr** also allows the user to use other optimization methods implemented in "`optim`" or use the non-linear minimization function "`nlm`" (R Core Team, 2013a). For example:

```
R> clikcorr(owlData, "feather_L", "feather_U", "AT_L", "AT_U", cp=.95, dist="t",
   + df=3, method="BFGS")
R> clikcorr(owlData, "feather_L", "feather_U", "AT_L", "AT_U", cp=.95, dist="t",
   + df=3, nlm=TRUE)
```

We note here that the default optimization method in "`optim`" function does not use derivative (or gradient) information but rather does a greedy simplex search on the function values, which makes it work relatively slowly. In future work, we plan to implement some independent gradient-directed types of optimization algorithms into **clikcorr** to hasten the convergence speed. It would be especially helpful in searching for the confidence interval bounds of the correlation coefficient.

clikcorr also allows R plotting options such as "xlab", "xlim", "bg" etc., in the "`splot`" and "`plot`" functions. For example, in Figure 6 the individual panels are generated respectively by

```
R> splot2(NHANESD, "t1_TCDD", "t2_TCDD", "t1_PeCDD", "t2_PeCDD", xlab="d1=TCDD",
   + ylab="d2=PeCDD", bg="#FFB90F", cex.lab=2.0)

R> splot2(NHANESD, "t1_HxCDF_123478", "t2_HxCDF_123478", "t1_HxCDF_123678",
   + "t2_HxCDF_123678", xlab="f4=HxCDF123478", ylab="f5=HxCDF123678", bg="#FFB90F",
   + cex.lab=2.0)

R> splot2(NHANESD, "t1_PCB_156", "t2_PCB_156", "t1_PCB_157", "t2_PCB_157",
   + xlab="p3=PCB156", ylab="p4=PCB157", bg="#FFB90F", cex.lab=2.0)

R> splot2(NHANESD, "t1_PCB_105", "t2_PCB_105", "t1_PCB_118", "t2_PCB_118",
   + xlab="p1=PCB105", ylab="p2=PCB106", bg="#FFB90F", cex.lab=2.0).
```

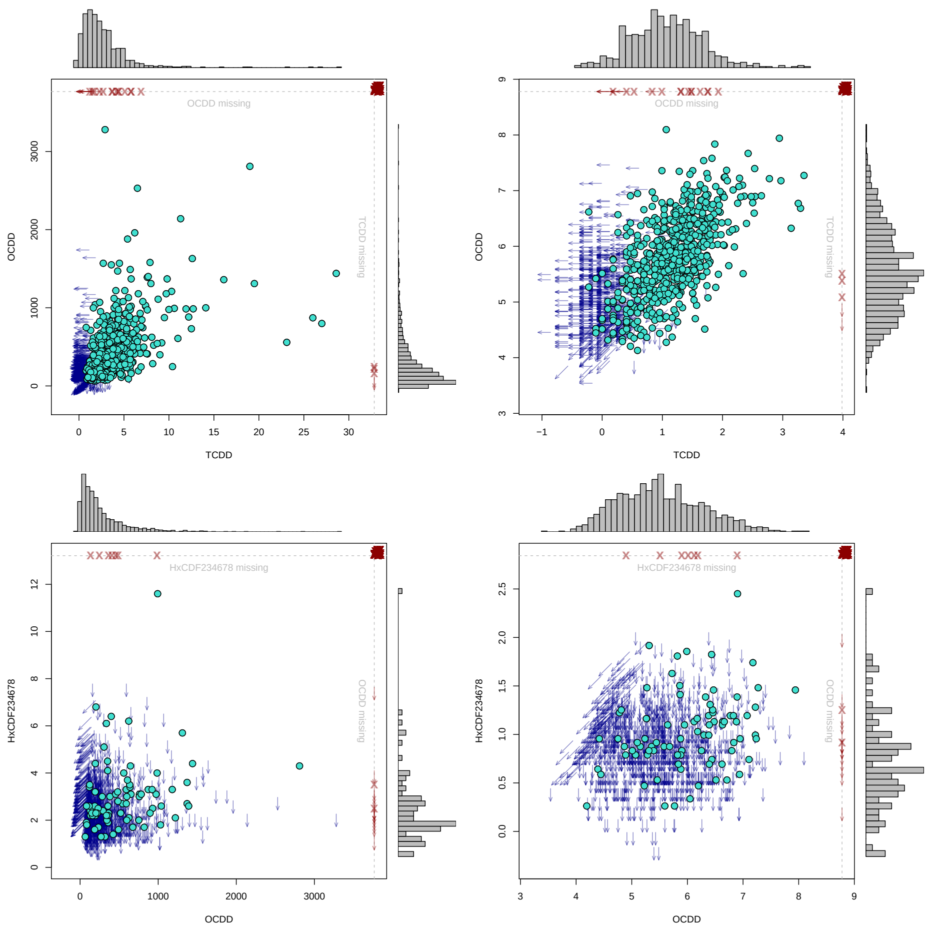


Figure 5: Example scatter plots between compound pairs with and without log transformation. Between *TCDD*(d1) and *ODD*(d7): (a) d1-d7 without log transformation, $\hat{\rho} = 0.22$, $p \approx 0$; (b) d1-d7 with log transformation, $\hat{\rho} = 0.60$, $p \approx 0$. Between *OCDD*(d7) and *HxCDF234678*(f6): (c) d7-f6 without log transformation, $\hat{\rho} = -0.05$, $p \approx 1$; (d) d7-f6 with log transformation, $\hat{\rho} = 0.38$, $p \approx 0$. The histograms on top and right of each plot are marginal sample distributions from observed data.

Sample size	0% Censoring rate			25% Censoring rate			75% Censoring rate		
	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$
n=50	0.938	0.962	0.946	0.938	0.946	0.968	0.958	0.954	0.956
	(0.956)	(0.968)	(0.954)	–	–	–	–	–	–
n=200	0.944	0.948	0.942	0.954	0.960	0.948	0.972	0.964	0.960
	(0.948)	(0.954)	(0.950)	–	–	–	–	–	–
n=500	0.932	0.934	0.952	0.946	0.948	0.964	0.952	0.944	0.964
	(0.938)	(0.948)	(0.954)	–	–	–	–	–	–

Table 4: 95% confidence interval coverage probabilities for bivariate normal data with fixed point censoring. Coverage probabilities are estimated from 500 replications. The Binomial random error is about ±0.01. Coverage probabilities in parentheses are calculated from Fisher transformation in the case of no censors. Data generated from the standard normal distribution with left censoring at 25% and 75% percentiles.

Sample size	Censoring distribution=N(-2,1)			Censoring distribution=N(0,1)			Censoring distribution=N(2,1)		
	Average censoring rate ≈ 0.25			Average censoring rate ≈ 0.50			Average censoring rate ≈ 0.75		
	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$
n=50	0.934	0.936	0.942	0.908	0.946	0.958	–	–	–
n=200	0.958	0.968	0.948	0.942	0.962	0.958	0.942	0.946	0.930
n=500	0.946	0.948	0.956	0.944	0.945	0.954	0.947	0.938	0.962

Table 5: 95% confidence interval coverage probabilities for bivariate normal data with random censoring. Coverage probabilities are estimated from 500 replications. The Binomial random error is about ±0.01. "–" means insufficient data for estimation.

Sample size	Censoring proportion configuration								
	(X 0%; XY 30%; Y 0%)			(X 15%; XY 15%; Y 15%)			(X 30%; XY 0%; Y 30%)		
	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$
n=50	36	50	59	22	25	32	1.2	1.4	1.8
n=200	166	206	233	89	103	110	1.5	2.1	2.2
n=500	267	515	728	176	275	374	1.8	2.4	3.5

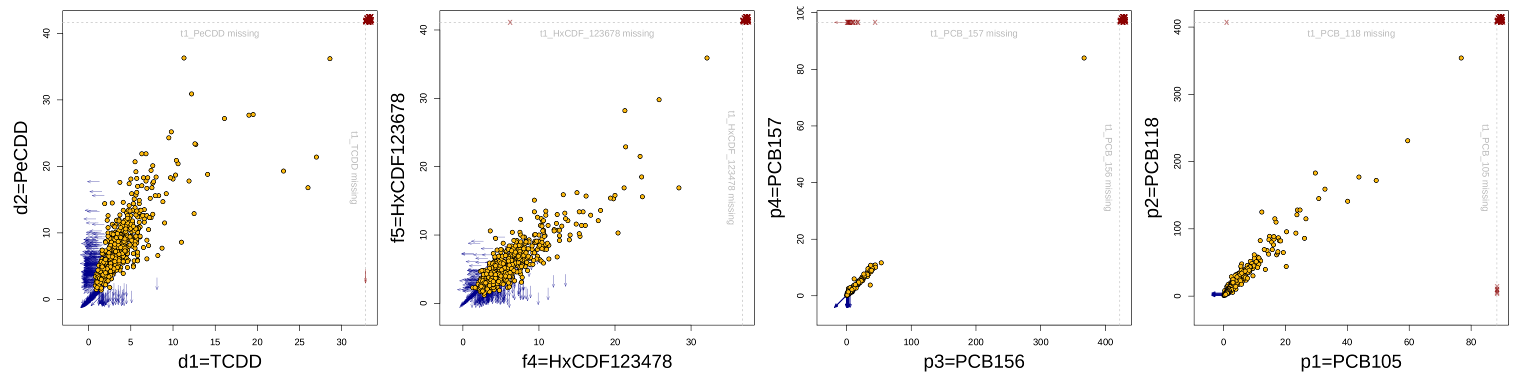
Table 6: Mean run time (sec.) for different settings of true correlations, sample sizes and censoring percentages. Mean time are estimated from 500 replications. Data are simulated from the standard bivariate normal distribution.

Sample size	Censoring distribution=N(-2,1) Average censoring rate $\approx$ 0.25			Censoring distribution=N(0,1) Average censoring rate $\approx$ 0.50			Censoring distribution=N(2,1) Average censoring rate $\approx$ 0.75		
	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$
n=50	0.006	-0.021	-0.018	0.031	0.007	0.008	–	–	–
	(0.019)	(0.030)	(0.053)	(0.035)	(0.064)	(0.076)	(–)	(–)	(–)
n=200	-0.005	-0.005	-0.026	0.020	0.003	-0.037	-0.017	0.053	0.021
	(0.008)	(0.012)	(0.014)	(0.012)	(0.010)	(0.014)	(0.024)	(0.034)	(0.029)
n=500	0.005	-0.008	-0.002	0.008	-0.001	-0.010	-0.006	-0.018	-0.010
	(0.002)	(0.003)	(0.005)	(0.003)	(0.006)	(0.007)	(0.010)	(0.009)	(0.013)

Table 7: Bias (mean squared error) for bivariate normal data. Bias and MSE are estimated from 50 replications. “–" means insufficient data for estimation.

Degree of freedom	Sample size = 50			Sample size = 200			Sample size = 500		
	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$	$\rho = 0.00$	$\rho = 0.50$	$\rho = 0.95$
df=3	0.95	0.81	0.69	0.96	0.76	0.55	0.96	0.70	0.54
		(0.93)	(0.93)		(0.97)	(0.97)		(0.95)	(0.96)
df=5	0.95	0.91	0.86	0.94	0.86	0.81	0.96	0.88	0.76
		(0.97)	(0.95)		(0.96)	(0.95)		(0.97)	(0.95)
df=10	0.94	0.93	0.89	0.93	0.96	0.89	0.95	0.94	0.91
			(0.93)			(0.94)			(0.94)
df=20	0.95	0.92	0.93	0.94	0.94	0.92	0.96	0.94	0.94

Table 8: 95% confidence interval coverage probabilities on bivariate t generated data. Coverage probabilities are estimated from 500 replications. The Binomial random error is about ± 0.01. For each df setting, the coverage probabilities on the first line are from inference based on the bivariate Normal distribution and the coverage probabilities in parenthesis on the second line are from inference based on the bivariate $t-$distribution. The latter are only given when the coverage probability from using the bivariate normal is ≤ 0.91.



(a) d1 vs. d2: $\hat{\rho}_{clikcorr} = 0.81$, $\hat{\rho}_{complete} = 0.77$, $n_{complete} = 585]$

(b) f4 vs. f5: $\hat{\rho}_{clikcorr} = 0.85$, $\hat{\rho}_{complete} = 0.89$, $n_{complete} = 717$

(c) p3 vs. p4: $\hat{\rho}_{clikcorr} = 0.99$, $\hat{\rho}_{complete} = 0.99$, $n_{complete} = 1048$

(d) p1 vs. p2: $\hat{\rho} = 0.97$, $\hat{\rho}_{complete} = 0.97$, $n_{complete} = 1340$

Figure 6: Scatter plots of the most highly correlated chemical pairs in each of the dioxin, furan and PCB groups in the NHANES data.

	Chemical index																					
	d1	d2	d3	d4	d5	d6	d7	f1	f2	f3	f4	f5	f6	f7	f8	f9	p1	p2	p3	p4	p5	p6
d1	–	**.81**	.68	.67	.61	.50	.22	.22	.30	.66	.65	.62	.36	.08	.05	.01	.44	.47	.30	.39	.57	.36
	–	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(1)	(0)	(0)	(0)	(0)	(.01)	(.46)	(.82)	(0)	(0)	(0)	(0)	(0)	(0)
d2	.79	–	.78	**.80**	.70	.52	.56	.46	.29	.73	.71	.71	.40	.10	.08	.03	.42	.47	.36	.49	.61	.38
	(0)	–	(0)	(0)	(0)	(0)	(0)	(0)	(.01)	(0)	(0)	(0)	(0)	(0)	(.17)	(.45)	(0)	(0)	(0)	(0)	(0)	(0)
d3	.69	.79	–	.71	.75	.58	.50	.19	.19	.60	.62	.61	.41	.07	.13	.08	.32	.35	.32	.36	.49	.39
	(0)	(0)	–	(0)	(0)	(0)	(1)	(0)	(.02)	(0)	(0)	(0)	(0)	(.01)	(.18)	(1)	(0)	(0)	(0)	(0)	(0)	(0)
d4	.65	.73	**.83**	–	.66	.58	.69	.15	.13	.75	.79	.77	.36	.13	.05	.00	.40	.46	.46	.51	.60	.39
	(0)	(0)	(0)	–	(0)	(0)	(0)	(.01)	(1)	(0)	(0)	(0)	(0)	(0)	(1)	(1)	(0)	(0)	(0)	(0)	(0)	(0)
d5	.60	.66	.75	.74	–	.55	.73	.30	.50	.56	.60	.63	.47	.13	.42	.18	.32	.34	.24	.28	.42	.27
	(0)	(0)	(0)	(0)	–	(0)	(0)	(0)	(1)	(0)	(0)	(0)	(0)	(0)	(1)	(1)	(0)	(0)	(0)	(0)	(0)	(0)
d6	.54	.49	.65	.56	.59	–	.76	.14	.15	.45	.57	.51	.31	.15	.12	.06	.39	.41	.22	.26	.41	.21
	(0)	(0)	(0)	(0)	(0)	–	(0)	(.08)	(1)	(0)	(0)	(0)	(0)	(0)	(1)	(1)	(0)	(0)	(0)	(0)	(0)	(0)
d7	.60	.58	.69	.66	.65	.77	–	.19	.33	.59	.70	.60	-.05	.14	-.53	.03	.38	.41	.28	.36	.56	.25
	(0)	(0)	(0)	(0)	(0)	(0)	–	(.01)	(1)	(0)	(0)	(0)	(1)	(0)	(1)	(.61)	(0)	(0)	(0)	(0)	(0)	(0)
f1	.29	.45	.20	.23	.35	.19	.24	–	**.83**	.37	.33	.36	.76	-.04	-.27	.24	.15	.15	.09	.10	.19	.61
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	–	(0)	(0)	(0)	(0)	(0)	(.39)	(1)	(1)	(0)	(0)	(.34)	(.05)	(0)	(1)
f2	.22	.20	.16	-.01	.21	.17	.20	.76	–	.46	.36	.43	**.82**	.06	.46	.21	.09	.08	.05	.07	.11	.57
	(0)	(0)	(.05)	(1)	(.02)	(.04)	(.02)	(0)	–	(0)	(0)	(0)	(0)	(.50)	(.04)	(1)	(.12)	(.16)	(.44)	(.19)	(.09)	(1)
f3	.67	.69	.70	.74	.62	.47	.59	.52	.55	–	.78	**.81**	.53	.12	.05	.02	.49	.54	.45	.51	.65	.38
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	–	(0)	(0)	(0)	(0)	(.48)	(1)	(0)	(0)	(0)	(0)	(0)	(0)
f4	.63	.62	.67	.74	.61	.57	.67	.34	.41	.73	–	**.85**	.50	.18	-.07	.03	.44	.48	.34	.41	.56	.31
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	–	(0)	(0)	(0)	(.30)	(1)	(0)	(0)	(0)	(0)	(0)	(0)
f5	.62	.67	.66	.75	.65	.55	.62	.42	.49	.77	.79	–	.58	.18	.001	.14	.38	.42	.34	.40	.51	.29
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	–	(0)	(0)	(.99)	(1)	(0)	(0)	(0)	(0)	(0)	(0)
f6	.39	.42	.43	.52	.53	.48	.38	.53	.78	.62	.59	.67	–	.12	.23	.36	.19	.33	.11	.13	.23	.26
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	–	(0)	(.02)	(1)	(0)	(1)	(0)	(0)	(0)	(0)
f7	.18	.20	.27	.30	.34	.28	.31	.21	.19	.26	.39	.42	.38	–	0	.13	.06	.06	.05	.05	.06	.04
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(.01)	(0)	(0)	(0)	(0)	–	(1)	(.13)	(.03)	(.02)	(.06)	(.05)	(.05)	(.25)
f8	.08	.12	.14	.02	.31	.06	.03	.31	.32	.08	-.02	.05	.29	.13	–	.28	.01	.02	.02	-.05	.06	-.15
	(.19)	(.06)	(.03)	(.75)	(0)	(.27)	(.54)	(0)	(0)	(.18)	(.77)	(.44)	(0)	(.02)	–	(0)	(.91)	(1)	(1)	(.40)	(1)	(.05)
f9	.02	.03	.04	.01	.08	.01	.05	.08	.11	.06	.07	.08	.09	.20	.42	–	.05	.01	.06	.09	.05	.02
	(.56)	(.47)	(.37)	(.84)	(.05)	(.85)	(.17)	(.30)	(.32)	(.11)	(.05)	(.03)	(.19)	(0)	(0)	–	(1)	(.55)	(1)	(1)	(.61)	(1)
p1	.58	.53	.53	.50	.48	.51	.54	.31	.36	.59	.54	.51	.34	.15	.02	.08	–	**.97**	.42	.44	.67	.27
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(1)	(0)	(0)	(0)	(0)	(0)	(.68)	(.03)	–	(0)	(0)	(0)	(0)	(0)
p2	.64	.59	.58	.60	.52	.56	.62	.32	.17	.67	.61	.57	.35	.15	.01	.08	**.96**	–	.52	.54	.79	.31
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(.02)	(0)	(0)	(0)	(0)	(0)	(.85)	(.02)	(0)	–	(0)	(0)	(0)	(0)
p3	.64	.68	.72	.69	.53	.39	.55	.32	.21	.73	.63	.64	.48	.13	-.05	.05	.60	.71	–	**.99**	**.83**	.69
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(.02)	(0)	(0)	(0)	(0)	(0)	(.39)	(.15)	(0)	(0)	–	(0)	(0)	(0)
p4	.65	.67	.67	.68	.47	.38	.55	.30	.26	.71	.62	.60	.35	.12	-.08	.02	.63	.72	**.96**	–	**.88**	.72
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(.16)	(.64)	(0)	(0)	(0)	–	(0)	(0)
p5	.63	.65	.67	.65	.48	.48	.59	.44	.22	.69	.62	.59	.31	.10	-.06	.03	.75	.83	**.89**	**.88**	–	.56
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(.05)	(0)	(0)	(0)	(0)	(0)	(.35)	(.40)	(0)	(0)	(0)	(0)	–	(0)
p6	.39	.39	.41	.43	.29	.23	.30	.23	.17	.44	.33	.33	.29	.11	-.17	-.07	.35	.41	.62	.67	.59	–
	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(.15)	(0)	(0)	(0)	(0)	(0)	(.02)	(.10)	(0)	(0)	(0)	(0)	(0)	–

Table 9: Correlation matrix for the NHANES dioxin data using **clikcorr**, upper-right=raw data, lower-left=log-transformed data.

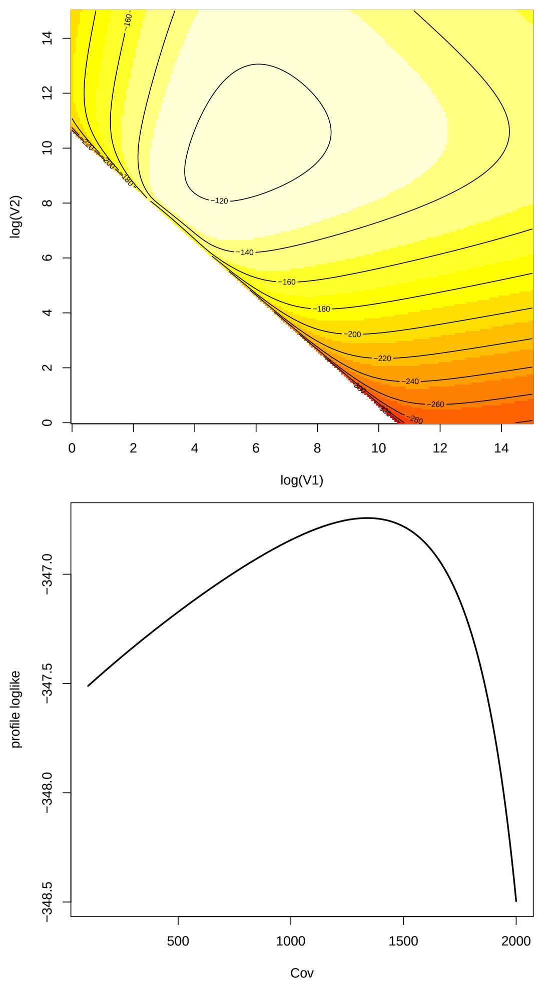


Figure 7: Illustration of effects of starting values on parameter estimates from profile log likelihood surfaces or curves. Upper panel: the profile joint log likelihood of parameters $(\log(\sigma_X^2), \log(\sigma_Y^2))$ with the mean and correlation parameters set at certain starting values. Values from highest to lowest are colored in order of white, yellow and red; Lower panel: the profile likelihood of covariance parameter with mean and variance parameters fixed at certain starting values.

Bibliography

P. D. Allison. *Survival Analysis Using SAS: A Practical Guide*. SAS Institute Inc., NC USA, 2010. [p161, 163]

M. C. Bryson. Heavy-tailed distributions: Properties and tests. *Technometrics*, 16:61–68, 1974. URL http://dx.doi.org/10.1080/00401706.1974.10489150. [p168]

R. A. Fisher. Frequency distribution of the values of the correlation coefficient in samples of an indefinitely large population. *Biometrika*, 10(4):507–521, 1915. URL http://dx.doi.org/10.2307/2331838. [p159, 160]

A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer-Verlag, Heideberg, 2009. ISBN 978-3-642-01688-2. [p165]

A. Genz, F. Bretz, T. Miwa, X. Mi, F. Leisch, F. Scheipl, and T. Hothorn. *mvtnorm: Multivariate Normal and t Distributions*, 2012. URL http://CRAN.R-project.org/package=mvtnorm. R package version 0.9-9994. [p165]

S. R. Giolo. Turnbull's nonparametric estimator for interval-censored data. *Technical Report*, 2004. URL http://www.est.ufpr.br/rt/suely04a.pdf. [p161]

D. R. Helsel. *Nondetects and Data Analysis: Statistics for Censored Environmental Data*. John Wiley & Sons, New Jersey, 2005. [p163]

D. R. Helsel. *Statistics for Censored Environmental Data Using MINITAB and R*. John Wiley & Sons, New Jersey, 2012. [p163]

T. Hothorn, F. Bretz, and A. Genz. *On Multivariate t and Gauss Probabilities in R*, 2013. URL http://cran.r-project.org/web/packages/mvtnorm/vignettes/MVT_Rnews.pdf. [p165]

V. L. Jaspers, D. Herzke, I. Eulaers, B. W. Gillespie, and M. Eens. Perfluoroalkyl substances in soft tissues and tail feathers of belgian barn owls using statistical methods for left censored data to handle non-detects. *Environment International*, 52:9–16, 2013. URL https://doi.org/10.1016/j.envint.2012.11.002. [p159, 160, 168]

L. Li, W. Wang, and I. Chan. Correlation coefficient inference on censored bioassay data. *Journal of Biopharmaceutical Statistics*, 15(3):501–512, 2005. URL https://doi.org/10.1081/BIP-200056552. [p159, 160, 163]

Y. Li, K. Shedden, B. W. Gillespie, and J. A. Gillespie. *clikcorr: Censoring Data and Likelihood-Based Correlation Estimation*, 2016. URL https://cran.r-project.org/web/packages/clikcorr/. [p159, 163]

R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, New York, 2002. [p161]

S. A. Murphy and A. W. van der Vaart. On profile likelihood. *J. Amer. Statist. Assoc*, 95:449–465, 2000. URL https://doi.org/10.1080/01621459.2000.10474219{"}. [p163]

R Core Team. *Non-Linear Minimization. The R Stats Package*, 2013a. URL http://ugrad.stat.ubc.ca/R/library/stats/html/nlm.html. [p171]

R Core Team. *General-Purpose Optimization. The R Stats Package*, 2013b. URL http://stat.ethz.ch/R-manual/R-devel/library/stats/html/optim.html. [p171]

C. R. Rao. *Linear Statistical Inference and Its Applications*. John Wiley & Sons, New York, 1973. [p161]

S. I. Resnick. *Heavy-Tail Phenomena Probabilistic and Statistical Modeling*. Springer-Verlag, New York, 2007. [p168]

M. Schemper, A. Kaider, S. Wakounig, and G. Heinze. Estimating the correlation of bivariate failure times under censoring. *Stat Med.*, 32(27):4781–4790, 2013. URL https://doi.org/10.1002/sim.5874. [p163]

H. Stryhn and J. Christensen. Confidence intervals by the profile likelihood methods, with applications in veterinary epidemiology. *Proceedings of the 10th International Symposium on Veterinary Epidemiology and Economics*, pages 208–211, 2003. [p163]

T. M. Therneau. *A Package for Survival Analysis in S*, 2015. URL https://CRAN.R-project.org/package=survival. version 2.38. [p163]

D. J. Venzon and S. H. Moolgavkar. A method for computing profile-likelihood-based confidence intervals. *Appl. Statist.*, 37(1):87, 1988. URL https://doi.org/10.2307/2347496. [p163]

H. Zhou, Z. Liao, S. Liu, W. Liang, X. Zhang, and C. Ou. Comparison of three methods in estimating confidence interval and hypothesis testing of logistic regression coefficients. *Journal of Mathematical Medicine*, 25(4):393–395, 2012. [p163]

Yanming Li
Department of Biostatistics
University of Michigan
Ann Arbor, MI 48109-2029
E-mail: liyanmin@umich.edu

Brenda Gillespie
Department of Biostatistics
University of Michigan
Ann Arbor, MI 48109-2029
E-mail: bgillesp@umich.edu

Kerby Shedden
Department of Statistics
University of Michigan
Ann Arbor, MI 48109-1107
E-mail: kshedden@umich.edu

John Gillespie
Department of Mathematics and Statistics
University of Michigan-Dearborn
Dearborn, MI 48128
E-mail: jgillesp@umich.edu

Spatial Uncertainty Propagation Analysis with the spup R Package

by Kasia Sawicka, Gerard B.M. Heuvelink and Dennis J.J. Walvoort

Abstract Many environmental and geographical models, such as those used in land degradation, agro-ecological and climate studies, make use of spatially distributed inputs that are known imperfectly. The R package **spup** provides functions for examining the uncertainty propagation from input data and model parameters onto model outputs via the environmental model. The functions include uncertainty model specification, stochastic simulation and propagation of uncertainty using Monte Carlo (MC) techniques. Uncertain variables are described by probability distributions. Both numerical and categorical data types are handled. The package also accommodates spatial auto-correlation within a variable and cross-correlation between variables. The MC realizations may be used as input to the environmental models written in or called from R. This article provides theoretical background and three worked examples that guide users through the application of **spup**.

Introduction and motivation

Environmental models that are used to understand and manage natural systems often rely on spatial data as input. Uncertainty in the input data propagates into model predictions. There is a need for systematic analysis of spatial uncertainty propagation in all major inputs and an accounting of spatial auto- and cross-correlation between input uncertainties. Spatial uncertainty propagation analysis has been widely recommended and practised across environmental disciplines, for example, hydrology and water quality (e.g. Hengl et al., 2010; Hamel and Guswa, 2015; Muthusamy et al., 2016; Nijhof et al., 2016; Yu et al., 2016), biogeochemistry (e.g. Boyer et al., 2006; Nol et al., 2010; Pennock et al., 2010; Hugelius, 2012; Vanguelova et al., 2016), or ecology and conservation planning (e.g. Jager et al., 2005; Fisher et al., 2010; Lechner et al., 2014). However, as far as we know, there is no widely applicable software that facilitates such analysis.

Currently, a growing number of software tools are available for uncertainty propagation and uncertainty assessment, some of which have functionality for dealing with spatial uncertainties. These include both free software, like OpenTURNS (Andrianov et al., 2007), Dacota (Adams et al., 2009) and DUE (Brown and Heuvelink, 2007); commercial options, like COSSAN (Schueller and Pradlwarter, 2006); or free packages for licenced software, like the SAFE (Pianosi et al., 2015) or UQLab (Marelli and Sudret, 2014) toolboxes for MATLAB. A broad review of existing software packages are available in Bastin et al. (2013). The use of powerful but complex languages like C++ in Dakota, Python in OpenTURNS or Java in DUE often discourage relevant portions of the scientific community without formal programming skills from the adoption of otherwise powerful tools. There are a number of R packages relating to uncertainty analysis through sensitivity analysis or use of Bayesian frameworks for model calibration. We have found only three packages: **propagate** (Spiess, 2014), **errors** (Ucar, 2017) and **metRology** (Ellison, 2017) that deal with uncertainty propagation explicitly. However, none of these packages provides functionality for spatial models and variables. Therefore, we undertook a project to develop an R package that facilitates uncertainty propagation analysis in spatial environmental and geographical modelling.

In this article, we present the **spup** package for R (Sawicka et al., 2017), which implements a methodology for spatial uncertainty propagation analysis using Monte Carlo (MC) methods. The **spup** package includes functions for uncertainty model specification, sampling, propagation of uncertainty using MC techniques and examples of results visualization. In this way, it provides support for the entire chain from uncertainty model definition, simulation and propagation to visualization. It is also a generic tool that can handle a great variety of cases. It is suitable for models that consider spatial variables saved in a spatial data frame or raster, as well as non-spatial variables, both continuous and categorical. It works with models written in R or other programming languages that can be called from R. The **spup** package is intended for researchers and practitioners who recognise the problems of uncertainty in data and models, and are looking for a simple, accessible implementation of a methodology for uncertainty propagation assessment and visualization. At the same time, it is designed to enable more experienced users to easily understand, customise, and possibly further develop the code.

Here, we provide a description of the implemented methodology, describe the package design and illustrate the usage with three simple examples. We demonstrate that the **spup** package is an effective and easy tool that can be used for multi-disciplinary research, model-based decision support and educational purposes.

Methodology implemented in spup

Uncertainty propagation aims to analyse how uncertainty (e.g. from measurement error, sampling, or interpolation) in spatial data, combined with modeling uncertainty (e.g. in model parameters and structure) propagate through the model (Heuvelink et al., 2007). Many environmental and geographical phenomena of interest are spatial in nature and often have strong spatial correlations imposed by the physics and dynamics of the natural systems, making uncertainty evaluation difficult.

A common approach to uncertainty propagation analysis makes use of MC stochastic simulation (Hammersley and Handscomb, 1979; Lewis and Orav, 1989). The MC approach is very flexible and can reach an arbitrary level of accuracy, and is therefore generally preferred over analytical methods such as the Taylor series method (Heuvelink et al., 1989). The MC method for uncertainty propagation analysis consists of the following main steps: (i) uncertainty about the variables is expressed by probability distribution functions (pdfs); (ii) many sets of possible uncertain inputs are generated from their marginal or joint probability distribution using a pseudo-random number generator; (iii) the model is run for each of the simulated input sets; (iv) the set of model outputs forms a random sample from the output pdf, so that the parameters of the output distribution, such as the mean, standard deviation and quantiles, can be estimated from the sample. In other words, the spread in the model outputs characterises how uncertainty about the model inputs has propagated to the model output. Note that the above ignores uncertainty in model parameters and model structure, but these can easily be included if available as pdfs. For example, these might have been obtained through Bayesian calibration (Van Oijen et al., 2005). The above four steps are described in detail further below.

Step 1 – Characterise uncertain model inputs with pdfs

The most frequently used uncertainty quantification approach represents uncertainty with probability distribution functions (pdfs). In order to represent input uncertainty with a pdf we have to define, for each possible input value, the corresponding probability or probability density. Probability distribution functions can be very complex and can have many parameters. In practice, when information and data from which to derive the pdf are limited, assumptions and simplifications have to be made to obtain reliable estimates of the pdf. In order to reduce the complexity of a pdf for a continuous variable, the distribution function may be described with a simple parametric shape, such as the normal, uniform or lognormal distribution (Bertsekas and Tsitsiklis, 2008). Categorical uncertain variables can be represented by non-parametric distribution, comprising a set of possible outcomes and their probabilities. The probabilities should be non-negative and the sum of the outcome probabilities should equal 1 (Heuvelink, 2007).

Individual variables and the uncertainties associated with them may also be statistically dependent on other variables and their uncertainties. For example, visibility and water vapour in the atmosphere are correlated, and so is the uncertainty between these two properties. Note that correlation between variables as derived from pair observations need not be the same as the correlation between measurement or interpolation errors associated with these variables. For example, elevation and temperature in mountainous area are negatively correlated, while the associated uncertainties may not be correlated at all because of independent measurements methods. Thus, when multiple inputs are uncertain, cross-correlations in their uncertainties must be addressed. In that case, the uncertainty model is described using a joint probability distribution function (jpdf). If the uncertainties are independent, the jpdf is the product of the univariate pdfs and can be estimated by estimating each pdf separately. If the uncertainties are statistically dependent, these dependencies must be estimated alongside the pdfs. In practice, few parametric shapes are available to describe the jpdf of variables that are statistically dependent. For continuous numerical variables, a common assumption is that the multivariate uncertainty model follows a joint-normal distribution with a vector of means μ and variance-covariance matrix Σ, where the latter must be square, symmetric and positive-semidefinite.

In different environmental or geographical studies, many variables are spatially distributed and it is likely that there is a relationship between nearby values (i.e. they are auto-correlated). Uncertainty in such cases may be described by the following geostatistical model:

$$Z(x) = \mu(x) + \sigma(x)\varepsilon(x) \tag{1}$$

where $\mu(x)$ is the mean of the variable $Z(x)$ at any location $x \in D$, D is the geographic domain of interest, $\sigma(x)$ is a spatially variable standard deviation associated with $\mu(x)$ (i.e. σ parameterises uncertainty such that it may be greater in some regions than in others), and ε is a zero-mean, unit-variance, second-order stationary residual (i.e. spatial auto-correlation depends only on a distance vector), modelled with a semivariogram or a correlogram (Diggle and Ribeiro, 2007; Webster and Oliver, 2007; Plant, 2012). For defining spatial correlation, **spup** supports a wide range of positive-definite functions implemented in the **gstat** package (Pebesma, 2004; Gräler et al., 2016). The mathematical background for these calculations is presented in Brown and Heuvelink (2007) and Heuvelink (2007). In case of spatial variables that are cross-correlated with other spatial variables, the linear model of coregionalization (Goovaerts, 1997) is assumed.

Table 1 summarises common assumptions and simplifications in uncertainty propagation analysis for three categories of data: continuous, discrete and categorical. These assumptions are included in the implementation of **spup**.

Step 2 – Repeatedly sample from spatial pdfs for uncertain inputs

When an uncertain variable is characterised by a pdf, simulation relies on a pseudo-random number generator. The most common sampling method is independent identically distributed (IID) sampling, in which case each realization is generated as independent draws from the same pdf. In case of a single parameter or univariate distribution, a realization can be easily drawn from various pdfs using algorithms implemented in the **stats** package in R (see `https://cran.r-project.org/web/views/Distributions.html` for details). For a spatially distributed variable (e.g. a DEM) that has a multivariate normal distribution, **spup** relies on the unconditional Gaussian simulation algorithm implemented in the **gstat** package (Pebesma, 2004). For simulation from multivariate normal pdfs of non-spatial variables, **spup** uses the **mvtnorm** package (Genz et al., 2017; Genz and Bretz, 2009).

In case the input is high-dimensional, straightforward random drawing from the jpdf may not yield a sample that is representative across the whole range of allowable values for each variable, unless the sample size is extremely large. Therefore stratified sampling methods, where values are selected for each variable from each of a pre-specified number of intervals with equal probability mass (Iman and Conover, 1982), are also implemented in **spup**. The latter can also be done in a spatial context (Pebesma and Heuvelink, 1999).

Step 3 – Run a model with sampled inputs and store model outputs

Step 3 consists of running a model (e.g. an environmental model) for all sample elements ("simulated realities") generated in Step 2. It is crucial that all models are automated and can be run in batch mode, as in many practical cases, the number of model runs for a Monte Carlo analysis should be at least 100, and often many more (Heuvelink, 1998). In general, the MC sample size must be large enough to guarantee that the outcome of the uncertainty propagation analysis is sufficiently accurate (Heuvelink, 2006). The limitation here is that the required sample size cannot be calculated beforehand because it is derived from the MC output.

It is important to plan what information one needs to retrieve from the set of model outputs. Strategies will vary if the user is interested in the combined or separate effect of uncertain input data or parameters. The simplest approach to estimating the uncertainty contribution of a single uncertain input or parameter is to perform the uncertainty propagation analysis with only the input or, one or more parameters made uncertain. The remaining uncertain parameters are then assumed certain and fixed on their central value. Comparison of the output uncertainties of different inputs (e.g. ratio of variances) provides a measure of the relative contribution. More details on stochastic sensitivity analysis can be found in Saltelli et al. (2000).

Running the model for multiple simulated realities resulting from step 2 may be computationally demanding, particularly when the model is complex and requires much computing time. The computation time can be reduced, as the MC method is very suited for parallel computing with multi-core machines and for grid computing technology (Leyva-Suarez et al., 2015). In this way, computation times can be dramatically reduced. Numerous high-performance and parallel computing packages have been developed in R suiting various approaches. A continuously maintained descriptive list of these is available online at `https://cran.r-project.org/web/views/HighPerformanceComputing.html` (Eddelbuettel, 2017).

Step 4 – Compute and visualise summary statistics of model outputs

The resulting set of model outputs forms a random sample from the model output probability distribution. This means that output pdf parameters, such as the mean, median, variance, standard deviation, interquartile range, percentiles, coefficient of variation and threshold exceedance probabilities can be estimated from the sample. For example, the percentiles of the sample can be computed by the `quantile()` function in R. The accuracy of the sample estimates can also be computed (de Gruijter et al., 2006), although this is currently not implemented in **spup**. The sample of output values should be stored in case the model output is used as uncertain input for a next model.

Table 1: Common assumptions for defining uncertainty models with a probability distribution function for continuous and discrete variables, and categorical variables.

	Quantifying the univariate pdf	**Uncertainty in spatially distributed variables**	**Cross-correlation between variables**
Continuous variable (e.g. rainfall, pollutant concentration, elevation)	A pdf is often assumed to follow the normal distribution with mean μ and standard deviation σ. Non-normal distributions (such as the exponential or lognormal distribution) may also be assumed. For distributions supported in **spup** see the **spup** manual. Alternatively, variables may be transformed to ensure that the transformed variable has a normal distribution. Common transformations are square root, logarithm and the Box-Cox transform. Hard minima and maxima can be imposed by using truncated distributions that leave the shape of distributions unchanged except that below the minimum and above the maximum the probability (density) is set to zero.	For continuous uncertain inputs that are spatially correlated, the normal distribution is often imposed because without it the uncertainty quantification would become too complex. Therefore, it is recommended to assume that the input or some transformation of it is multivariate normally distributed and define the uncertainty in terms of the normally distributed (transformed) variable. In addition, 'second-order stationarity' is also often assumed (Goovaerts, 1997; Diggle and Ribeiro, 2007), indicating that the spatial correlation between errors at two locations depends only on the distance vector between the locations. This enables us to represent spatial correlation with a simple function (correlogram).	The statistical dependencies between jointly normally-distributed uncertain variables are fully characterised with a correlation matrix. The correlation matrix is a square, symmetric and positive-semidefinite matrix with unit values on the diagonal and values between -1 and 1 on the off-diagonals. Correlations can be estimated using paired observations of the variables of interest. However, it is important to note that the correlation between variables as derived from paired observations need not be the same as the correlation between measurement or interpolation errors associated with these variables. E.g., DOC and NH4 concentrations of river water are typically positively correlated, while their respective measurement errors are not, if measured independently.
Discrete variable (e.g. fish count, number of wildfires in a given area and time period)	The pdf can be represented by a parametric shape, e.g. Poisson distribution, with mean or rate λ, where $\mu_Z = \sigma_Z = \lambda$. In case no appropriate shape is available, all possible outcomes and associated probabilities must be tabulated or listed in a non-parametric pdf.	When discrete or categorical variables are spatially distributed, the simplest approach is to assume spatial statistical independence or perfect spatial dependence. More advanced approaches that allow intermediate levels of statistical dependence include the Markov random field approach (Blake et al., 2011) and the Generalised Linear Geostatistical Model (Diggle and Ribeiro, 2007). It is also possible to divide or cluster the area into sub-regions, with a complete dependence within and independence between sub-regions.	Cross-dependencies between uncertain categorical variables are addressed by treating all combinations of categories as separate classes. Each combination is assigned a probability. This increases the number of classes dramatically, so that generalisation is often needed in practice and the total number of combined classes is reduced to ten or fewer.
Categorical variable (e.g. land use, building type)	For most categorical variables, a parametric shape may not be available. In that case, each possible outcome and associated probability must be listed in a non-parametric pdf.		

It is important that the uncertainty statistics are communicated in an efficient way that is simultaneously informative and understandable to users that have varied backgrounds in statistics. There should be ample explanation of the results and a visual representation of the uncertainty (MacEachren, 1992; Hengl et al., 2004). For non-spatial and non-temporal data, a number of simple statistical plots can be used to visualise uncertainty represented by probability distributions. If the full probability distribution function is known, this can be visualised in pdf or cumulative distribution function (cdf) plots. Error bars, interquartile range and box plots can be used to show the distribution of values. Uncertainties about categorical data can be shown using stacked bars for the probability for each of the categories, or entropy that provides a summary measure of the spread of probabilities over the categories.

It is important to distinguish between presentation techniques and presentation modes for uncertaintly visualisation of spatial data. General presentation techniques are (i) adjacent maps, (ii) sequential presentation and (iii) bi-variate maps (MacEachren et al., 2005). These techniques can be presented in three modes: static, dynamic and interactive (Kinkeldey et al., 2014). A common example of adjacent maps for continuous variables is to put maps of the mean and standard deviation next to each other. For visualising categorical attribute uncertainty, the simplest method is also using adjacent maps, for example by showing a map of the the most probable category next to a map of the probability of that category. Sequential presentation may be shown in dynamic mode.

The spup package

The **spup** package is developed in R to provide a simple solution for those who wish to include uncertainty analysis in their studies, with emphasises on studies including spatial variables. The **spup** package implements the methodology described in Section Methodology implemented in spup. We describe **spup** version 1.3-1 available on CRAN. The development version is available on gitHub at `https://github.com/ksawicka/spup`. The package is freely available under the GPL3.

Three example datasets are provided with the **spup** package:

1. Digital Elevation Model of Zlatibor region in Serbia (3x4.5km, 30m resolution) (Hengl et al., 2008)
2. Soil organic carbon and total nitrogen content of the south region of Lake Alaotra in Madagascar (33x33km, 250m resolution) (Hengl et al., 2017)
3. Rotterdam neighbourhood buildings distribution (Kadaster, The Netherlands).

These datasets are used in Section Application examples and in the vignettes included in the package. Additionally, the package includes simple R and C functions that represent environmental models used in examples.

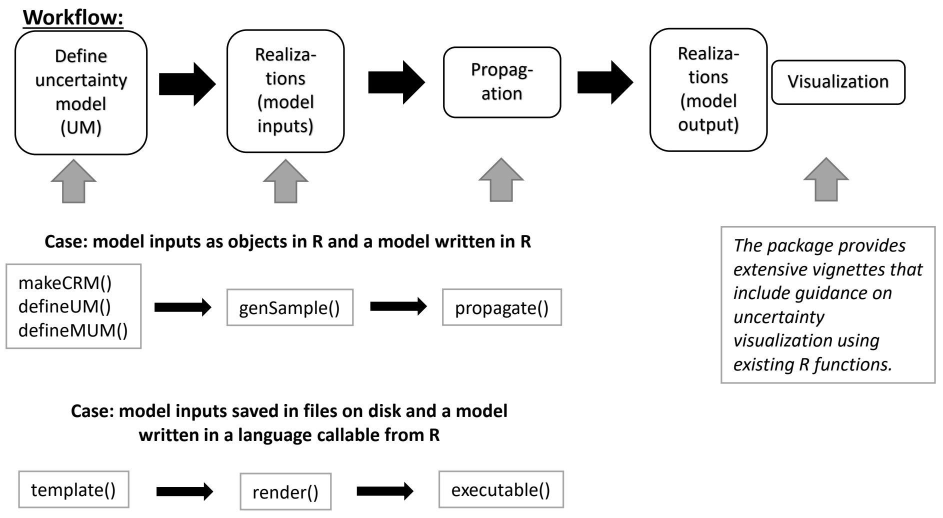

Workflow:
Define uncertainty model (UM)
Realiza-tions (model inputs)
Propag-ation
Realiza-tions (model output)
Visualization
Case: model inputs as objects in R and a model written in R
makeCRM()
defineUM()
defineMUM()
genSample()
propagate()
The package provides extensive vignettes that include guidance on uncertainty visualization using existing R functions.
Case: model inputs saved in files on disk and a model written in a language callable from R
template()
render()
executable()

Figure 1: Flowchart presenting workflow with functions implemented in **spup**.

Figure 1 presents a flowchart with the workflow and key functions implemented in **spup**. The package is designed in a way that the key functions correspond with the steps presented in Section Methodology implemented in spup. The output of functions for defining the uncertainty model becomes an input to functions for generating a MC sample. The generated MC sample in turn is used as input to functions facilitating propagation of uncertainty. Qualitative descriptions of the key functions are given in Table 2. More details about functions arguments and output can be found in the package manual (Sawicka et al., 2017) and in the examples below.

Table 2: Key functions in package **spup**.

Function	Description
`makeCRM()`	Define a spatial correlogram model.
`defineUM()`	Define marginal uncertainty distributions for model inputs/parameters for subsequent Monte Carlo analysis. Output class depends on the arguments provided.
`defineMUM()`	Allows the user to define joint uncertainty distributions for continuous model inputs/parameters for subsequent Monte Carlo analysis.
`genSample()`	Methods for generating a Monte Carlo sample of uncertain variables.
`template()`	Defines a `"container"` class to store all templates with model inputs.
`executable()`	Produces an R function wrapper around a model that can be called using `system2()` from R.
`render()`	Replaces the tags in moustaches in template file or character object by text.
`propagate()`	Runs the model repeatedly with Monte Carlo realizations of the uncertain input/parameters.

Application examples

Example 1 - Uncertainty propagation analysis with auto- and cross- correlated variables

In many geographical studies, variables are cross-correlated. As a result, uncertainty in one variable may be statistically dependent on uncertainty in the other. For example, soil properties such as organic carbon (OC) and total nitrogen (TN) content are cross-correlated. These variables are used to derive the C/N ratio, which is important information to evaluate soil management and increase crop productivity. Errors in OC and TN maps will propagate into the C/N ratio map. The cross-correlation between the uncertainties in OC and TN will affect the degree of uncertainty in the C/N ratio.

The example data for C/N calculations are a 250m resolution mean OC and TN of a 33km x 33km area adjacent to lake Alaotra in Madagascar (Figure 2) (Hengl et al., 2017). The datasets include four spatial objects: a mean OC and TN of the area, with their standard deviations assumed to be equal to 10% of the mean.

```
# load packages
library(spup)
library(raster)
library(purrr)

# set seed
set.seed(12345)

# load and view the data
data(OC, OC_sd, TN, TN_sd)
class(OC); class(TN)

par(mfrow = c(1, 2), mar = c(1, 1, 2, 2), mgp = c(1.7, 0.5, 0), oma = c(0, 0, 0, 1),
    las = 1, cex.main = 1, tcl = -0.2, cex.axis = 0.8, cex.lab = 0.8)
plot(OC, main = "Mean of Organic Carbon", xaxt = "n", yaxt = "n")
plot(TN, main = "Mean of Total Nitrogen", xaxt = "n", yaxt = "n")
```

The first step in uncertainty propagation analysis is to define an uncertainty model for the uncertain input variables, here OC and TN. First, the marginal uncertainty model is defined for each variable separately, and next the joint uncertainty model is defined for the variables together. In case of OC and TN, the ε (Eq. 1) are spatially correlated. For each of the variables, the `makeCRM()` function collates all necessary information into a list. We assume that the spatial autocorrelation of the OC and TN

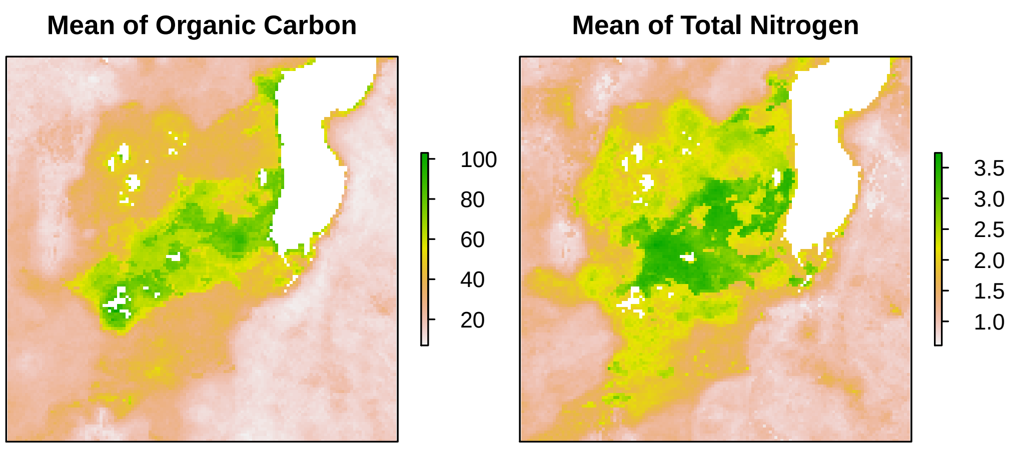


Figure 2: Maps of means of soil organic carbon and soil total nitrogen near lake Alaotra in Madagascar.

errors are both described by a spherical correlation function with a short-distance correlation of 0.6 for OC and 0.4 for TN, with a range parameter of 1000m (Figure 3). It is important at this step to ensure that the correlation function types as well as the ranges are the same for each variable. This is a requirement for further analysis, because **spup** employs the linear model of co-regionalization (Wackernagel, 2003).

```
# define spatial correlogram models
OC_crm <- makeCRM(acf0 = 0.6, range = 1000, model = "Sph")
TN_crm <- makeCRM(acf0 = 0.4, range = 1000, model = "Sph")

par(mfrow = c(1, 2), mar = c(3, 2.5, 2, 1), mgp = c(1.7, 0.5, 0), oma = c(0, 0, 0, 0),
    las = 1, cex.main = 1, tcl = -0.2, cex.axis = 0.8, cex.lab = 0.8)
plot(OC_crm, main = "OC correlogram")
plot(TN_crm, main = "TN correlogram")
```

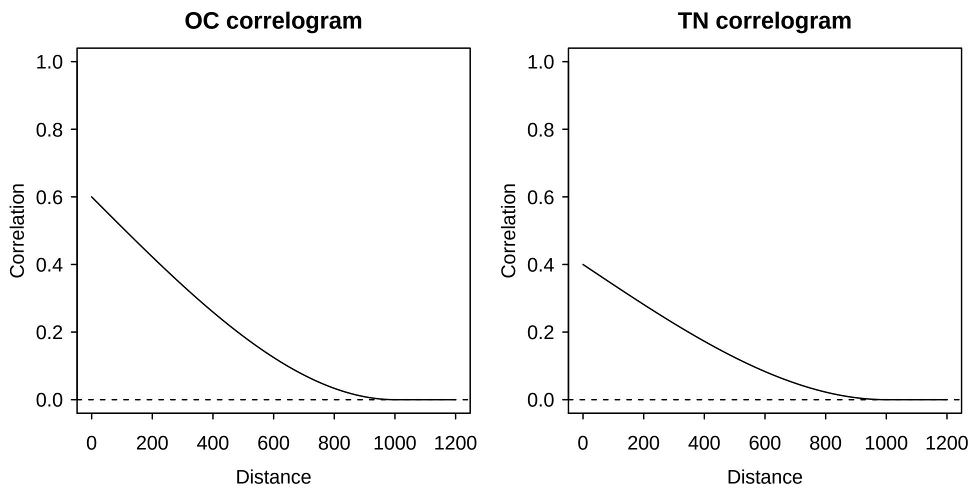


Figure 3: Spatial correlograms of soil OC and TN in the study area.

Spatial correlograms summarise patterns of spatial autocorrelation in data and model residuals. They show the degree of correlation between values at two locations as a function of the separation distance between the locations. In the case above, as is usually the case, the correlation declines with distance. Here, the correlation is zero for distances greater than 1000m. More information about correlograms is included in the **spup** DEM vignette. In order to complete the description of each individual uncertain variable, the `defineUM()` function collates all information about the OC and TN uncertainty.

```
# define uncertainty model for the OC and TN
```

```
OC_UM <- defineUM(distribution = "norm", distr_param = c(OC, OC_sd),
                  crm = OC_crm, id = "OC")

TN_UM <- defineUM(distribution = "norm", distr_param = c(TN, TN_sd),
                  crm = TN_crm, id = "TN")

class(OC_UM)
[1] "MarginalNumericSpatial"
class(TN_UM)
[1] "MarginalNumericSpatial"
```

Both variables are of the same class "MarginalNumericSpatial". This is one of the requirements for defining a multivariate uncertainty model. The defineMUM() function collates information about uncertainty in each variable, and information about their cross-correlation.

```
# define multivariate uncertainty model
mySpatialMUM <- defineMUM(UMlist = list(OC_UM, TN_UM),
                          cormatrix = matrix(c(1, 0.7, 0.7, 1),
                          nrow = 2, ncol = 2))

class(mySpatialMUM)
[1] "JointNumericSpatial"
```

In this example, a geostatistical model for OC and TN is defined that assumes multivariate normality and has spatial correlation functions as follows:

$$\begin{aligned}
&\rho_{OC}(h) = 1 \quad \text{for } h = 0\\
&\rho_{OC}(h) = 0.6 \cdot \{1 - \frac{3}{2}\frac{h}{1000} + \frac{1}{2}(\frac{h}{1000})^3\} \quad \text{for } 0 < h < 1000\\
&\rho_{OC}(h) = 0 \quad \text{for } h \geq 1000
\end{aligned} \tag{2}$$

$$\begin{aligned}
&\rho_{TN}(h) = 1 \quad \text{for } h = 0\\
&\rho_{TN}(h) = 0.4 \cdot \{1 - \frac{3}{2}\frac{h}{1000} + \frac{1}{2}(\frac{h}{1000})^3\} \quad \text{for } 0 < h < 1000\\
&\rho_{TN}(h) = 0 \quad \text{for } h \geq 1000
\end{aligned} \tag{3}$$

$$\begin{aligned}
&\rho_{OC,TN}(h) = 0.7 \quad \text{for } h = 0\\
&\rho_{OC,TN}(h) = 0.7 \cdot \sqrt{0.6 \cdot 0.4} \cdot \{1 - \frac{3}{2}\frac{h}{1000} + \frac{1}{2}(\frac{h}{1000})^3\} \quad \text{for } 0 < h < 1000\\
&\rho_{OC,TN}(h) = 0 \quad \text{for } h \geq 1000
\end{aligned} \tag{4}$$

where h is Euclidean distance and where the mathematical definition of the spherical correlation function (i.e., the semivariogram) has been used (Webster and Oliver, 2007).

Generating possible realities of the selected variables is completed by using the genSample() function:

```
# create possible realizations from the joint distribution of OC and TN
MC <- 100
OCTN_sample <- genSample(UMobject = mySpatialMUM, n = MC, samplemethod = "ugs",
                         nmax = 20, asList = FALSE)
```

Note the argument asList has been set to FALSE. This indicates that the sampling function will return an object of a class of the distribution parameters class. This is useful if there is a need to visualise the sample or compute summary statistics quickly.

A first assessment of uncertainty is done by plotting the means and standard deviations of the sampled OC and TN. Note that if the sample size is very large, then the sample mean and standard deviation would approximate the mean and standard deviation (Figure 4).

```
# compute and plot OC and TN sample statistics
# e.g. mean and standard deviation
OC_sample <- OCTN_sample[[1:MC]]
TN_sample <- OCTN_sample[[(MC+1):(2*MC)]]
OC_sample_mean <- mean(OC_sample)
```

```
TN_sample_mean <- mean(TN_sample)
OC_sample_sd <- calc(OC_sample, fun = sd)
TN_sample_sd <- calc(TN_sample, fun = sd)

par(mfrow = c(1, 2), mar = c(1, 1, 2, 2), mgp = c(1.7, 0.5, 0), oma = c(0, 0, 0, 1),
    las = 1, cex.main = 1, tcl = -0.2, cex.axis = 0.8, cex.lab = 0.8)
plot(OC_sample_mean, main = "Mean of OC realizations", xaxt = "n", yaxt = "n")
plot(TN_sample_mean, main = "Mean of TN realizations", xaxt = "n", yaxt = "n")
```

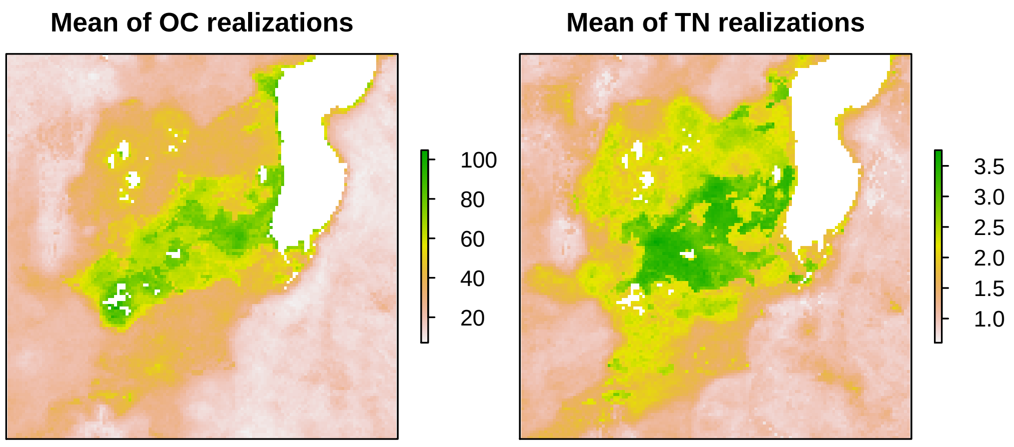


Figure 4: Maps of means of sampled realizations of OC and TN.

In order to perform uncertainty propagation analysis using **spup**, the model through which uncertainty is propagated needs to be defined as an R function:

```
# C/N model
C_N_model_raster <- function (OC, TN) OC/TN
```

The propagation of uncertainty occurs when the model is run with uncertain inputs. Running the model with a sample of realizations of uncertain input variable(s) yields an equally large sample of model outputs that can be further analysed. We use the `propagate()` function to run the C/N ratio model with the OC and TN realizations. To run `propagate()`, the samples of the uncertain input variables must be saved in lists and then collated into a list of lists. The existing `OCTN_sample` object can be coerced or generated automatically by setting `asList = TRUE` in `genSample()`.

```
# coerce  a raster stack to a list
l <- list()
l[[1]] <- map(1:100, function(x){OCTN_sample[[x]]})
l[[2]] <- map(101:200, function(x){OCTN_sample[[x]]})
OCTN_sample <- l

# run uncertainty propagation
CN_sample <- propagate(realizations = OCTN_sample, model = C_N_model_raster, n = MC)
```

Uncertainty in C/N predictions can be visualised by calculating and plotting the relative error (Figure 5). The relative error gives an indication of the accuracy relative to its size.

```
# coerce C/Ns list to a raster stack
CN_sample <- stack(CN_sample)
names(CN_sample) <- paste("CN.", c(1:nlayers(CN_sample)), sep = "")

# compute and plot the slope sample statistics
CN_mean <- mean(CN_sample)
CN_sd <- calc(CN_sample, fun = sd)

par(mfrow = c(1, 2), mar = c(1, 1, 2, 2), mgp = c(1.7, 0.5, 0), oma = c(0, 0, 0, 1),
    las = 1, cex.main = 1, tcl = -0.2, cex.axis = 0.8, cex.lab = 0.8)
plot(CN_mean, main = "Mean of C/N", xaxt = "n", yaxt = "n")
plot((CN_sd/CN_mean)*100, main = "Relative error of C/N [\%]", xaxt = "n", yaxt = "n")
```

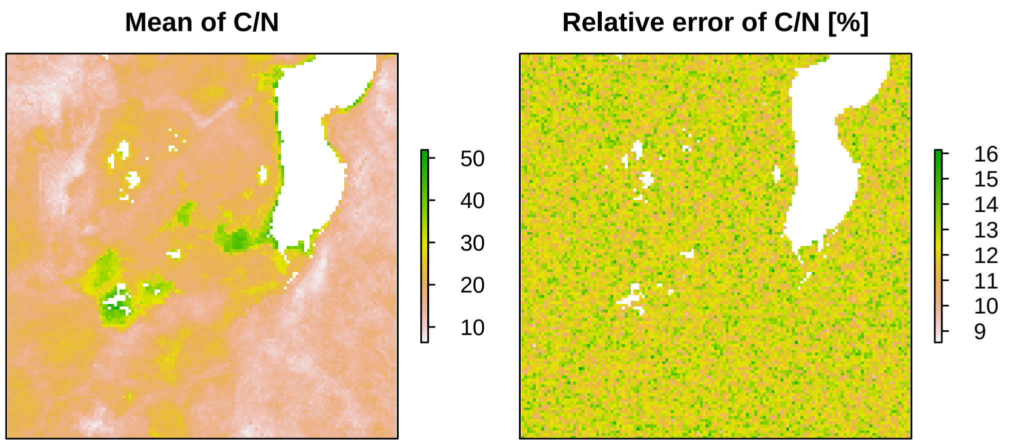


Figure 5: Maps of standard deviation and relative error of C/N ratio sample in the study area.

Further analysis may include identification of all locations where the C/N ratio is in a specific range with decreasing probability. For example identifying areas where the C/N ratio is higher than 20 might help farmers to identify which plots require action to improve soil quality (Figure 6).

```
# calculate quantiles
CN_sample_df <- as(CN_sample, "SpatialGridDataFrame")
CN_q <- quantile_MC_sgdf(CN_sample_df, probs = c(0.01, 0.1, 0.5, 0.9), na.rm = TRUE)

CN_q$good4crops99perc <- ifelse(CN_q$prob1perc > 20, 1, 0)
CN_q$good4crops90perc <- ifelse(CN_q$prob10perc > 20, 1, 0)
CN_q$good4crops50perc <- ifelse(CN_q$prob50perc > 20, 1, 0)
CN_q$good4crops10perc <- ifelse(CN_q$prob90perc > 20, 1, 0)

CN_q$good4crops <- CN_q$good4crops99perc + CN_q$good4crops90perc +
 CN_q$good4crops50perc + CN_q$good4crops10perc

CN_q$good4crops[CN_q$good4crops == 4] <- "No improvement needed"
CN_q$good4crops[CN_q$good4crops == 3] <- "Possibly improvement needed"
CN_q$good4crops[CN_q$good4crops == 2] <- "Likely improvement needed"
CN_q$good4crops[CN_q$good4crops == 1] <- "Definitely improvement needed"
CN_q$good4crops[CN_q$good4crops == 0] <- "Definitely improvement needed"

CN_q$good4crops <- factor(CN_q$good4crops,
                          levels = c("Definitely improvement needed",
                                     "Likely improvement needed",
                                     "Possibly improvement needed",
                                     "No improvement needed"))

spplot(CN_q, "good4crops", col.regions = c("red3", "sandybrown", "greenyellow",
           "forestgreen"), main = "Areas with sufficient C/N for cropping")
```

Example 2 - Uncertainty propagation analysis with categorical data

In many aspects of life, information take the form of categorical data. For example, in a city neighbourhood or a district, each building may be assigned a function such as housing, office, recreation, or other. The city council may impose different tax levels depending on the building function. Suppose that the function of a building is assigned depending on the percentage of the building used for living (i.e., the residential use) and number of addresses in the building. If the residential area is greater than 80% and has at least one address registered, then the building is classified as "residential". If the residential area is smaller than 80% and at least one address is present, then the building is classified as "office". If no address is present, the building function is classified as "other". The city council imposes annual tax of €1000 for residential buidings, €10000 for office buildlings, and €10 for other buildings. The 80% threshold is approximate, and some buildings classified as "other" could have an assigned

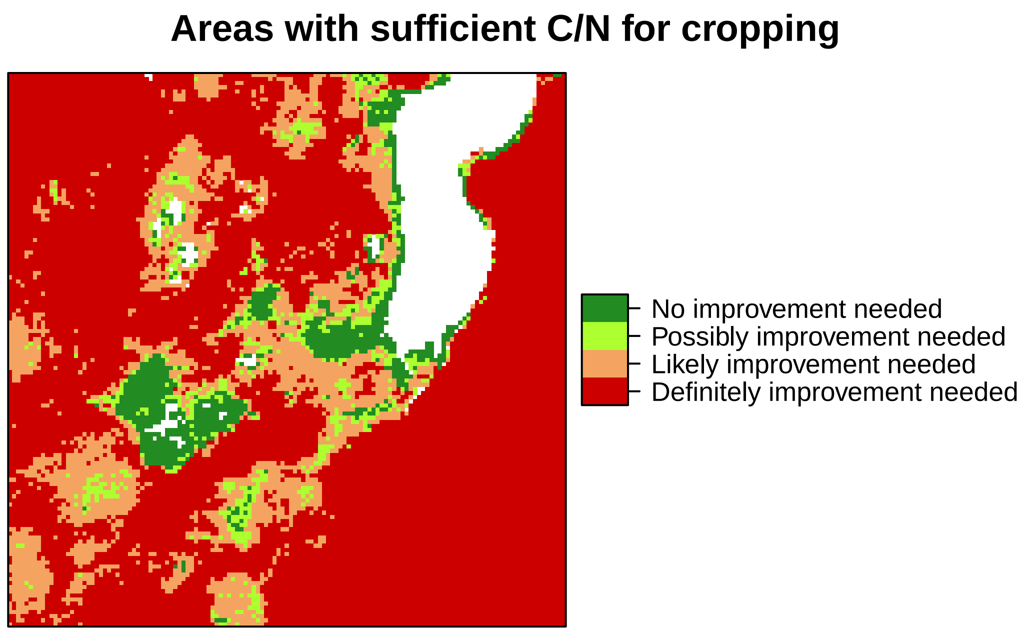

Areas with sufficient C/N for cropping
No improvement needed
Possibly improvement needed
Likely improvement needed
Definitely improvement needed

Figure 6: Map of soil quality classification (depending on C/N ratio) for crop production.

address that has not yet been entered into the tax system. Therefore, the council wishes to calculate the uncertainty in tax revenue introduced by erroneous building classification. For this example, we will use the spatial distribution of buildings in a neighborhood of Rotterdam, NL (Figure 7).

```
# load packages
library(sp)
library(spup)
library(purrr)
library(png)

set.seed(12345)

# load data
data(woon)

# Netherlands contour and Rotterdam location
NL <- readPNG("RotterdamNL.png")
# collate info about figure size and type
NL_map <- list("grid.raster", NL, x = 89987, y = 436047, width = 120, height = 152,
               default.units = "native")
# collate info about a scale bar location in the figure, type and colour
scale <- list("SpatialPolygonsRescale", layout.scale.bar(),
              offset = c(90000, 435600), scale = 100,
              fill = c("transparent", "black"))
# collate info about minimum value on a scale bar
text1 <- list("sp.text", c(89990, 435600), "0", cex = 0.8)
# collate info about maximum value on a scale bar
text2 <- list("sp.text", c(90130, 435600), "500 m", cex = 0.8)
# collate info about North arrow
arrow <- list("SpatialPolygonsRescale", layout.north.arrow(),
              offset = c(89990, 435630), scale = 50)
# plot "woon" object with a location miniature,
# North arrow and scale bar defined above
spplot(woon, "check", do.log = TRUE, col.regions = "transparent",
       colorkey = FALSE, key.space = list(x = 0.1, y = 0.93, corner = c(0,1)),
       sp.layout = list(scale, text1, text2, arrow, NL_map),
       main = "Neighbourhood in Rotterdam, NL")
```

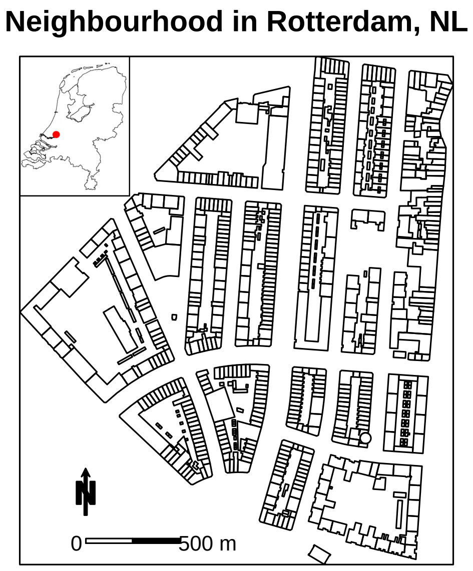


Figure 7: Map of a neighbourhood of Rotterdam used as study area. Contours represent buildings.

The woon object is a SpatialPolygonDataFrame where each building is represented by one polygon (Figure 7). The attributes include:

- vbos: The number of addresses present in the building
- woonareash: The percentage residential area
- Function: The assigned category depending on vbos and woonareash, where 1 represents residental; 2, office, and 3, other.
- residential: Probability that the building is residential
- office: Probability that the building is an office buildling
- other: Probability that the building has another function

Figure 8 illustrates the probabilities associated with each building for the three possible categories.

```
# plot probabilities for each polygon
spplot(woon[c(4,5,6)])
```

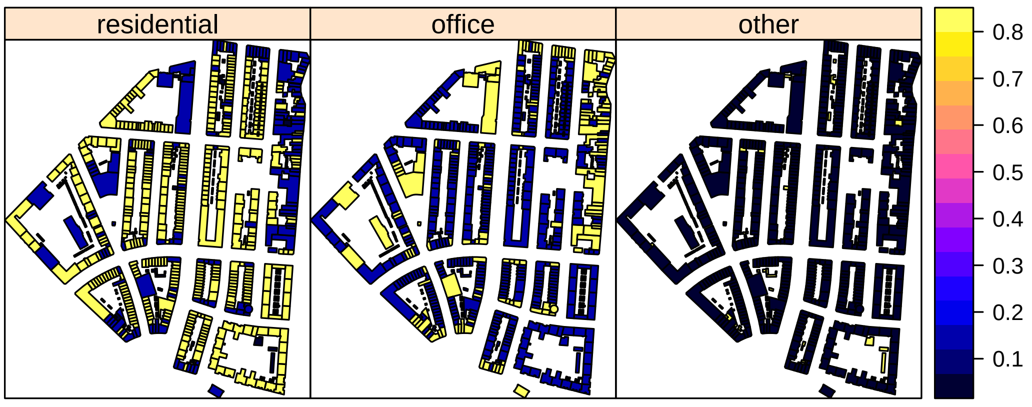


Figure 8: Probabilities of correct classification of each building.

In case of categorical variables, the uncertainty is described by a non-parametric pdf. In this case, it is a vector of probabilities that a building belongs to a certain category. In case of spatially distributed variables, this must be done for each polygon, hence the dataset has maps of these probabilities saved in the same object. We further assume that uncertainties are spatially independent, implying that the joint pdf of all buildings in the neighbourhood is the product of the marginal pdfs for all buildings. Thus, we can generate possible realities for all buildings independently. To unite all information of the uncertainty model for the building function, we use the `defineUM()` function that collates all information into one object.

```
# define uncertainty model for the building function
woonUM <- defineUM(TRUE, categories = c("residential", "office", "other"),
                   cat_prob = woon[, c(4:6)])

# create possible realizations of the building function
woon_sample <- genSample(woonUM, 100, asList = FALSE)

# view several realizations
woon_s <- woon_sample@data
woon_s <- lapply(woon_sample@data, as.factor)
woon_sample@data <- as.data.frame(woon_s)
rm(woon_s)

spplot(woon_sample[c(3,4,1,2)], col.regions = c("#5ab4ac", "#f5f5f5", "#d8b365"),
       main = list(label = "Examples of building function realizations", cex = 1))
```

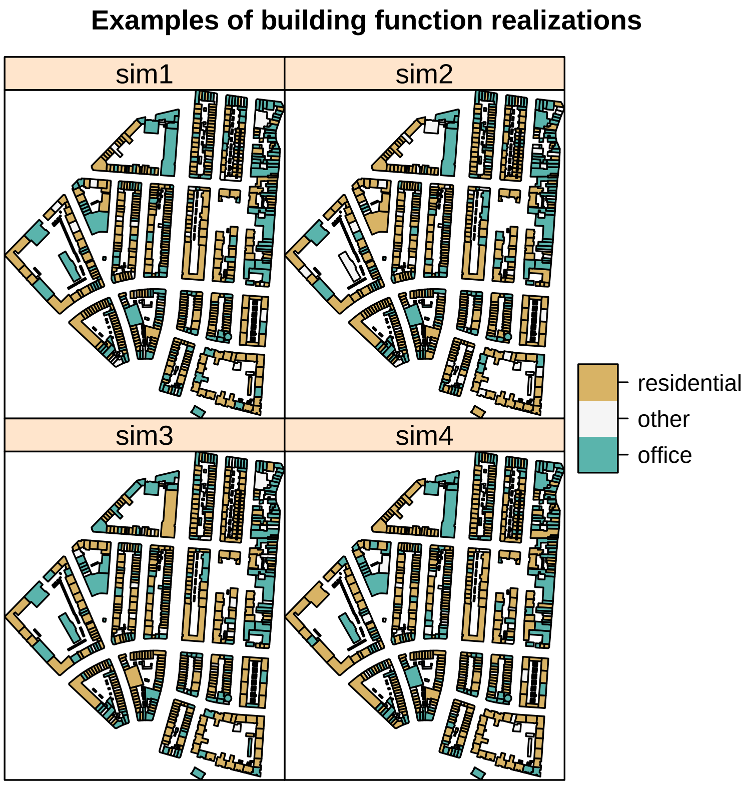


Figure 9: Examples of Monte Carlo realizations of possible classifications for each building.

Examples of building functions Monte Carlo realizations are shown in Figure 9.

To propagate uncertainty, we run the model repeatedly with the sample created above. The tax model is defined as:

```
# define tax model
tax <- function (building_Function)
{
  building_Function$tax2pay <- NA
  building_Function$tax2pay[building_Function$Function == "residential"] <- 1000
  building_Function$tax2pay[building_Function$Function == "office"] <- 10000
  building_Function$tax2pay[building_Function$Function == "other"] <- 10
  total_tax <- sum(building_Function$tax2pay)
  total_tax
}
```

As in the previous C/N ratio example, in order to run the propagation function, the sample of an uncertain input variable must be saved in a list. We must either coerce the existing `woon_sample` object or get it automatically by setting up `asList = TRUE` argument in `genSample()`.

```
# coerce SpatialPolygonDataFrame to a list of individual SpatialPolygonDataFrames
woon_sample <- map(1:ncol(woon_sample), function(x){woon_sample[x]})
for (i in 1:100) names(woon_sample[[i]]) <- "Function"

# run uncertainty propagation
tax_sample <- propagate(woon_sample, model = tax, n = 100)

tax_sample <- unlist(tax_sample)
ci95perc <- c(mean(tax_sample) - 1.96*(sd(tax_sample)/10),
mean(tax_sample) + 1.96*(sd(tax_sample)/10))
ci95perc
[1] 2281978 2312449
```

The result of the uncertainty propagation shows that on average, the city council should obtain a tax revenue of approximately €2,300,000 and that the associated 95% confidence interval is (€2,281,978, €2,312,449).

Example 3 - Uncertainty propagation analysis with a model written in C

Environmental models are often developed in languages other than R, such as C or FORTRAN, that can significantly speed up processing. In this example, we demonstrate how to perform uncertainty analysis with a basic model written in C.

We begin by showcasing functions to manipulate model inputs stored in ASCII files. For external models, this additional code is needed to (i) modify ASCII input files, and (ii) run the external model. For rendering ASCII input files, the mustache templating framework is available on GitHub at `https://mustache.github.io`, and in the R package **whisker**.

Suppose we have a simple linear regression model: $Y = b0 + b1 * X$ that requires an input file `input.txt`. The file contains values for the two parameters $b0$ and $b1$ as follows:

```
library(spup)
library(dplyr)
library(readr)
library(whisker)
library(purrr)

set.seed(12345)

read_lines("input.txt")
[1] "-0.789 0.016"
```

Function `template()` allow user to define a `"container"` class to store templates with model inputs. The aim of this class is to organise model input files and perform necessary checks. A `print()` method is available for the class `"template"`. A template is simply an ASCII file with:

1. The additional extension .template.
2. Input that needs to be modified is replaced by mustache-style tags.

The corresponding template file should be named `input.txt.template` and should replace the original numbers with `b0` and `b1` placed in moustaches: {{...}}.

```
# function template() reads the file into R as an object of class "template"
my_template <- template("input.txt.template")
my_template %>%
read_lines
[1] "{{b0}} {{b1}}"
```

Rendering is the process of replacing the tags in moustaches with text. Rendering creates a new file, called `input.txt`.

```
my_template %>%
render(b0 = 3, b1 = 4)
[1] "input.txt"
```

Below is an example external model written in the C language. This code can be saved in a file with the .c extension, for example, 'dummy_model.c'. The model below calculates a value of a simple linear regression:

```
#include <stdio.h>
int main() {
        FILE *fp;
        double x[9] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
        double y;
        double b0;
        double b1;
        int i;

        fp = fopen("input.txt", "r");
        if (fp == NULL) {
                printf("Can't read input.txt\n");
                return 1;
        }
        fscanf(fp, "%lf %lf\n", &b0, &b1);
        fclose(fp);

        fp = fopen("output.txt", "w");
        if (fp == NULL) {
                printf("Can't create output.txt\n");
                return 1;
        }
        else {
                for (i=0; i<9; i++) {
                        y = b0 + b1 * x[i];
                        fprintf(fp, "%10.2f\n", y);
                }
                fclose(fp);
        }
        return 0;
}
```

To compile this code to a MS-Windows executable one can use GCC at the DOS command prompt as follows: `gcc dummy_model.c -o dummy_model`. This creates a file `dummy_model.exe`, which can be wrapped in R as follows:

```
dummy_model <- executable("dummy_model.exe")
```

Running the rendering procedure passes values for $b0$ and $b1$ to the model, which gives:

```
# render the template
render(my_template, b0 = 3.1, b1 = 4.2)

# run external model
dummy_model()

# read output (output file of dummy_model is "output.txt")
scan(file = "output.txt", quiet = TRUE)
[1]  7.3 11.5 15.7 19.9 24.1 28.3 32.5 36.7 40.9
```

To perform the uncertainty propagation analysis, we derive multiple realizations of the model output in steps as follows: (i) render the template, (ii) run the model, (iii) read the results, and (iv) process the results. For example:

```
# number of Monte Carlo runs
n_realizations <- 100
n_realizations %>%
purrr::rerun({
        # render template
        render(my_template, b0 = rnorm(n = 1), b1 = runif(n = 1))

        # run model
        dummy_model()

        # read output
        scan("examples/output.txt", quiet = TRUE)
}) %>%
set_names(paste0("r", 1:n_realizations)) %>%
as_data_frame %>%
apply(MARGIN = 1, FUN = quantile)

[,1]    [,2]    [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
0%   -1.660 -0.9300 -0.7400 -0.730 -0.710 -0.700 -0.680 -0.660 -0.650
25%  -0.060  0.4775  0.8625  1.203  1.720  1.970  2.345  2.607  2.848
50%   0.745  1.2400  1.7200  2.310  2.965  3.375  3.910  4.405  4.850
75%   1.518  2.1425  2.7725  3.453  4.147  4.832  5.453  6.135  6.947
100%  2.990  3.3800  4.2700  5.170  6.080  6.990  7.890  8.800  9.710
```

Summary and future directions

In this paper, we presented the R package **spup** which implements methodologies for spatial uncertainty propagation analysis. The package architecture and its core components were described and three examples were presented. We explained the package functions for examining the uncertainty propagation, starting from input data and model parameters via the environmental model to model predictions. Sources of uncertainty include model specification, stochastic simulation and propagation of uncertainty using MC techniques. We include recommendations for visualizing results. We also explained how numerical and categorical data types are handled, and how we accomodate spatial auto-correlation within an attribute and cross-correlation between attributes. The MC realizations may be used as an input to environmental models written in R or other programming languages that can be called from R. The presented examples and the vignettes included in the **spup** package provide insight into the selection of appropriate static uncertainty visualizations that are understandable to audiences of differing levels of statistical literacy. As an effective and easy tool, **spup** has potential to be used for multi-disciplinary research, model-based decision support and educational purposes.

The package could benefit from further development including facilitating uncertainty analysis with time series, one of the most common types of data in environmental studies. For uncertainty propagation, the package has implemented the MC approach with efficient sampling algorithms such as stratified random sampling; implementing Latin Hypercube sampling would complement this approach. Further direction may include interactive visualization methods could be developed via the **shiny** package (Chang et al., 2017). As it is based on MC methodology, **spup** is suitable to be used with parallel computing.

Acknowledgements

We thank Sytze de Bruin, Damiano Luzzi and Stefan van Dam for valuable contributions to the development of **spup**. We thank Filip Biljecki and Branislav Bajat for providing the data used in the Rotterdam and Zlatibor examples. This project has received funding from the European Union's Seventh Framework Programme for Research, Technological Development and Demonstration under grant agreement no 607000.

Authors contributions

Kasia Sawicka is the main developer of **spup** and author of the text in this article. Gerard Heuvelink is a co-author of **spup**, he contributed to this article with his statistical and programming knowledge and edited the text. Dennis Walvoort is a co-author of **spup** and contributed to the development of the article.

Bibliography

B. M. Adams, L. E. Bauman, W. J. Bohnhoff, K. R. Dalbey, M. S. Ebeida, J. P. Eddy, M. S. Eldred, P. D. Hough, K. T. Hu, J. D. Jakeman, L. P. Swiler, and D. M. Vigil. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.4 user's manual. Report Sandia Technical Report SAND2010-2183, Sandia National Laboratories, 2009. [p180]

G. Andrianov, S. Burriel, S. Cambier, A. Dutfoy, I. Dutka-Malen, E. de Rocquigny, B. Sudret, P. Benjamin, R. Lebrun, F. Mangeant, and M. Pendola. Openturns, an open source initiative to treat uncertainties, risks'n statistics in 520 a structured industrial approach. In *ESREL'2007 Safety and Reliability Conference*, 2007. [p180]

L. Bastin, D. Cornford, R. Jones, G. B. M. Heuvelink, E. Pebesma, C. Stasch, S. Nativi, P. Mazzetti, and M. Williams. Managing uncertainty in integrated environmental modelling: The uncertweb framework. *Environmental Modelling & Software*, 39:116–134, 2013. ISSN 1364-8152. URL https://doi.org/10.1016/j.envsoft.2012.02.008. [p180]

D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, 2nd edition, 2008. ISBN 9781886529236. [p181]

A. Blake, P. Kohli, and C. Rother. *Markov Random Fields for Vision and Image Processing*. MIT Press, 2011. [p183]

E. W. Boyer, R. B. Alexander, W. J. Parton, C. Li, K. Butterbach-Bahl, S. D. Donner, R. W. Skaggs, and S. J. D. Grosso. Modeling denitrification in terrestrial and aquatic ecosystems at regional scales. *Ecological Applications*, 16(6):2123–2142, 2006. ISSN 1939-5582. URL https://doi.org/10.1890/1051-0761(2006)016[2123:mditaa]2.0.co;2. [p180]

J. D. Brown and G. B. M. Heuvelink. The data uncertainty engine (due): A software tool for assessing and simulating uncertain environmental variables. *Computers & Geosciences*, 33(2):172–190, 2007. ISSN 0098-3004. URL https://doi.org/10.1016/j.cageo.2006.06.015. [p180, 182]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *Shiny: Web Application Framework for R*, 2017. URL https://CRAN.R-project.org/package=shiny. R package version 1.0.5. [p195]

J. J. de Gruijter, D. J. Brus, M. F. P. Bierkens, and M. Knotters. *Sampling for Natural Resource Monitoring*. Springer, 2006. URL https://doi.org/10.1007/3-540-33161-1. [p182]

P. J. Diggle and P. J. Ribeiro. *Model-Based Geostatistics*. Springer-Verlag, 2007. ISBN 0387485368. URL https://doi.org/10.1007/978-0-387-48536-2. [p181, 183]

D. Eddelbuettel. Cran task view: High-performance and parallel computing with r, 2017. [p182]

S. L. R. Ellison. *Support for Metrological Applications*, 2017. URL https://CRAN.R-project.org/package=metRology. R package version 0.9-26-2. [p180]

R. Fisher, N. McDowell, D. Purves, P. Moorcroft, S. Sitch, P. Cox, C. Huntingford, P. Meir, and F. Ian Woodward. Assessing uncertainties in a second-generation dynamic vegetation model caused by ecological scale limitations. *New Phytologist*, 187(3):666–681, 2010. ISSN 1469-8137. URL https://doi.org/10.1111/j.1469-8137.2010.03340.x. [p180]

A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer-Verlag, Heidelberg, 2009. ISBN 9783642016882. [p182]

A. Genz, F. Bretz, T. Miwa, X. Mi, F. Leisch, F. Scheipl, and T. Hothorn. *mvtnorm: Multivariate Normal and t Distributions*, 2017. URL https://CRAN.R-project.org/package=mvtnorm. R package version 1.0-6. [p182]

P. Goovaerts. *Geostatistics for Natural Resources Evaluation*. Oxford University Press, New York, 1997. ISBN 9780195115383. [p182, 183]

B. Gräler, E. Pebesma, and G. Heuvelink. Spatio-temporal interpolation using gstat. *The R Journal*, 8: 204–218, 2016. ISSN 2073-4857. [p181]

P. Hamel and A. J. Guswa. Uncertainty analysis of a spatially explicit annual water-balance model: Case study of the cape fear basin, north carolina. *Hydrol. Earth Syst. Sci.*, 19(2):839–853, 2015. ISSN 1607-7938. URL https://doi.org/10.5194/hess-19-839-2015. [p180]

J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Chapman and Hall, London, 1979. [p181]

T. Hengl, D. J. J. Walvoort, A. Brown, and D. G. Rossiter. A double continuous approach to visualization and analysis of categorical maps. *International Journal of Geographical Information Science*, 18(2):183–202, 2004. ISSN 1365-8816. URL https://doi.org/10.1080/13658810310001620924. [p184]

T. Hengl, B. Bajat, D. Blagojevic, and H. I. Reuter. Geostatistical modeling of topography using auxiliary maps. *Computers & Geosciences*, 34(12):1886–1899, 2008. ISSN 0098-3004. URL https://doi.org/10.1016/j.cageo.2008.01.005. [p184]

T. Hengl, G. B. M. Heuvelink, and E. E. van Loon. On the uncertainty of stream networks derived from elevation data: The error propagation approach. *Hydrol. Earth Syst. Sci.*, 14(7):1153–1165, 2010. ISSN 1607-7938. URL https://doi.org/10.5194/hess-14-1153-2010. [p180]

T. Hengl, J. Mendes de Jesus, G. B. M. Heuvelink, M. Ruiperez Gonzalez, M. Kilibarda, A. Blagotić, W. Shangguan, M. N. Wright, X. Geng, B. Bauer-Marschallinger, M. A. Guevara, R. Vargas, R. A. MacMillan, N. H. Batjes, J. G. B. Leenaars, E. Ribeiro, I. Wheeler, S. Mantel, and B. Kempen. Soilgrids250m: Global gridded soil information based on machine learning. *PLOS ONE*, 12(2): e0169748, 2017. URL https://doi.org/10.1371/journal.pone.0169748. [p184, 185]

G. B. M. Heuvelink. *Error Propagation in Environmental Modelling with GIS*. Taylor and Francis, London, 1998. [p182]

G. B. M. Heuvelink. *Uncertainty in Remote Sensing and GIS*, book section Analysing Uncertainty Propagation in GIS: Why is it not that Simple?, pages 155–165. John Wiley & Sons, 2006. ISBN 9780470035269. URL https://doi.org/10.1002/0470035269.ch10. [p182]

G. B. M. Heuvelink. Error-aware gis at work: Real-world applications of the data uncertainty engine. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2007. [p181, 182]

G. B. M. Heuvelink, P. A. Burrough, and A. Stein. Propagation of errors in spatial modelling with gis. *International Journal of Geographical Information Systems*, 3(4):303–322, 1989. ISSN 0269-3798. URL https://doi.org/10.1080/02693798908941518. [p181]

G. B. M. Heuvelink, J. D. Brown, and E. E. van Loon. A probabilistic framework for representing and simulating uncertain environmental variables. *International Journal of Geographical Information Science*, 21(5):497–513, 2007. ISSN 1365-8816. URL https://doi.org/10.1080/13658810601063951. [p181]

G. C. G. B. Hugelius. Spatial upscaling using thematic maps: An analysis of uncertainties in permafrost soil carbon estimates. *Global Biogeochemical Cycles*, 26(2):n/a–n/a, 2012. ISSN 1944-9224. URL https://doi.org/10.1029/2011gb004154. [p180]

R. L. Iman and W. J. Conover. A distribution-free approach to inducing rank correlation among input variables. *Communications in Statistics - Simulation and Computation*, 11(3):311–334, 1982. ISSN 0361-0918. URL https://doi.org/10.1080/03610918208812265. [p182]

H. I. Jager, A. W. King, N. H. Schumaker, T. L. Ashwood, and B. L. Jackson. Spatial uncertainty analysis of population models. *Ecological Modelling*, 185(1):13–27, 2005. ISSN 0304-3800. URL https://doi.org/10.1016/j.ecolmodel.2004.10.016. [p180]

C. Kinkeldey, A. M. MacEachren, and J. Schiewe. How to assess visual communication of uncertainty? a systematic review of geospatial uncertainty visualisation user studies. *The Cartographic Journal*, 51(4):372–386, 2014. ISSN 0008-7041. URL https://doi.org/10.1179/1743277414y.0000000099. [p184]

A. M. Lechner, C. M. Raymond, V. M. Adams, M. Polyakov, A. Gordon, J. R. Rhodes, M. Mills, A. Stein, C. D. Ives, and E. C. Lefroy. Characterizing spatial uncertainty when integrating social data in conservation planning. *Conserv Biol*, 28(6):1497–511, 2014. ISSN 0888-8892. URL https://doi.org/10.1111/cobi.12409. [p180]

P. A. W. Lewis and E. J. Orav. *Simulation Methodology for Statisticians, Operations Analysts, and Engineers*, volume 1. Wadsworth & Brooks/Cole, 1989. ISBN 9780534094508. [p181]

E. Leyva-Suarez, G. S. Herrera, and L. M. de la Cruz. A parallel computing strategy for monte carlo simulation using groundwater models. *Geofísica Internacional*, 54(3):245–254, 2015. ISSN 0016-7169. URL https://doi.org/10.1016/j.gi.2015.04.020. [p182]

A. M. MacEachren. Visualizing uncertain information. *Cartographic Perspective*, 13:10–19, 1992. [p184]

A. M. MacEachren, A. Robinson, S. Hopper, S. Gardner, R. Murray, M. Gahegan, and E. Hetzler. Visualizing geospatial information uncertainty: What we know and what we need to know. *Cartography and Geographic Information Science*, 32(3):139–160, 2005. ISSN 1523-0406. URL `https://doi.org/10.1559/1523040054738936`. [p184]

S. Marelli and B. Sudret. *Vulnerability, Uncertainty, and Risk*, book section UQLab: A Framework for Uncertainty Quantification in Matlab, pages 2554–2563. American Society of Civil Engineers, 2014. ISBN 978-0-7844-1360-9. URL `https://doi.org/10.1061/9780784413609.257`. [p180]

M. Muthusamy, A. Schellart, S. Tait, and G. B. M. Heuvelink. Geostatistical upscaling of rain gauge data to support uncertainty analysis of lumped urban hydrological models. *Hydrol. Earth Syst. Sci. Discuss.*, 2016:1–27, 2016. ISSN 1812-2116. URL `https://doi.org/10.5194/hess-2016-279`. [p180]

C. O. Nijhof, M. A. Huijbregts, L. Golsteijn, and R. van Zelm. Spatial variability versus parameter uncertainty in freshwater fate and exposure factors of chemicals. *Chemosphere*, 149:101–7, 2016. ISSN 0045-6535. URL `https://doi.org/10.1016/j.chemosphere.2016.01.079`. [p180]

L. Nol, G. B. M. Heuvelink, A. Veldkamp, W. de Vries, and J. Kros. Uncertainty propagation analysis of an n2o emission model at the plot and landscape scale. *Geoderma*, 159(1–2):9–23, 2010. ISSN 0016-7061. URL `https://doi.org/10.1016/j.geoderma.2010.06.009`. [p180]

E. J. Pebesma. Multivariable geostatistics in s: The gstat package. *Computers & Geosciences*, 30(7): 683–691, 2004. ISSN 0098-3004. URL `https://doi.org/10.1016/j.cageo.2004.03.012`. [p181, 182]

E. J. Pebesma and G. B. M. Heuvelink. Latin hypercube sampling of gaussian random fields. *Technometrics*, 41(4):303–312, 1999. ISSN 00401706. URL `https://doi.org/10.2307/1271347`. [p182]

D. Pennock, T. Yates, A. Bedard-Haughn, K. Phipps, R. Farrell, and R. McDougal. Landscape controls on n2o and ch4 emissions from freshwater mineral soil wetlands of the canadian prairie pothole region. *Geoderma*, 155(3–4):308–319, 2010. ISSN 0016-7061. URL `https://doi.org/10.1016/j.geoderma.2009.12.015`. [p180]

F. Pianosi, F. Sarrazin, and T. Wagener. A matlab toolbox for global sensitivity analysis. *Environmental Modelling & Software*, 70:80–85, 2015. ISSN 1364-8152. URL `https://doi.org/10.1016/j.envsoft.2015.04.009`. [p180]

R. E. Plant. *Spatial Data Analysis in Ecology and Agriculture Using R*. CRC Press, 2012. ISBN 1439819130. [p181]

A. Saltelli, S. Tarantola, and F. Campolongo. Sensitivity anaysis as an ingredient of modeling. *Statistical Science*, pages 377–395, 2000. ISSN 0883-4237. URL `https://doi.org/10.1214/ss/1009213004`. [p182]

K. Sawicka, G. B. M. Heuvelink, and D. J. J. Walvoort. *Spup: Uncertainty Propagation Analysis*, 2017. R package version 0.1-1. [p180, 185]

G. I. Schueller and H. J. Pradlwarter. Computational stochastic structural analysis (cossan) — a software tool. *Structural Safety*, 28(1–2):68–82, 2006. ISSN 0167-4730. URL `https://doi.org/10.1016/j.strusafe.2005.03.005`. [p180]

A.-N. Spiess. *Propagate: Propagation of Uncertainty*, 2014. URL `https://CRAN.R-project.org/package=propagate`. R package version 1.0-4. [p180]

I. Ucar. *Errors: Error Propagation for R Vectors*, 2017. URL `https://CRAN.R-project.org/package=errors`. R package version 0.0.2. [p180]

M. Van Oijen, J. Rougier, and R. Smith. Bayesian calibration of process-based forest models: Bridging the gap between models and data. *Tree Physiology*, 25(7):915–927, 2005. ISSN 0829-318X. URL `https://doi.org/10.1093/treephys/25.7.915`. [p181]

E. I. Vanguelova, E. Bonifacio, B. De Vos, M. R. Hoosbeek, T. W. Berger, L. Vesterdal, K. Armolaitis, L. Celi, L. Dinca, O. J. Kjønaas, P. Pavlenda, J. Pumpanen, U. Püttsepp, B. Reidy, P. Simončič, B. Tobin, and M. Zhiyanski. Sources of errors and uncertainties in the assessment of forest soil carbon stocks at different scales—review and recommendations. *Environmental Monitoring and Assessment*, 188(11): 630, 2016. ISSN 1573-2959. URL `https://doi.org/10.1007/s10661-016-5608-5`. [p180]

H. Wackernagel. *Multivariate Geostatistics: An Introduction with Applications*. Springer, 2003. [p186]

R. Webster and M. A. Oliver. *Geostatistics for Environmental Scientists*. John Wiley & Sons, 2nd edition, 2007. [p181, 187]

X. Yu, A. Lamacova, C. Duffy, P. Kram, and J. Hruska. Hydrological model uncertainty due to spatial evapotranspiration estimation methods. *Computers & Geosciences*, 90, Part B:90–101, 2016. ISSN 0098-3004. URL `https://doi.org/10.1016/j.cageo.2015.05.006`. [p180]

Kasia Sawicka
Centre for Ecology & Hydrology
Deiniol Road
Bangor
LL57 2UW
United Kingdom
(Previously at: Soil Geography and Landscape Group
Wageningen University
PO Box 47
6700AA Wageningen
The Netherlands)
ORCiD: 0000-0002-7553-3149
`katwic55@ceh.ac.uk`

Gerard B.M. Heuvelink
Soil Geography and Landscape group
Wageningen University
PO Box 47
6700AA Wageningen
The Netherlands
ORCiD: 0000-0003-0959-9358
`gerard.heuvelink@wur.ml`

Dennis J.J. Walvoort
Wageningen Environmental Research
PO Box 47
6700AA Wageningen
The Netherlands
`dennis.walvoort@wur.nl`

clustMixType: User-Friendly Clustering of Mixed-Type Data in R

by Gero Szepannek

Abstract Clustering algorithms are designed to identify groups in data where the traditional emphasis has been on numeric data. In consequence, many existing algorithms are devoted to this kind of data even though a combination of numeric and categorical data is more common in most business applications. Recently, new algorithms for clustering mixed-type data have been proposed based on Huang's k-prototypes algorithm. This paper describes the R package **clustMixType** which provides an implementation of k-prototypes in R.

Introduction

Clustering algorithms are designed to identify groups in data where the traditional emphasis has been on numeric data. In consequence, many existing algorithms are devoted to this kind of data even though a combination of numeric and categorical data is more common in most business applications. For an example in the context of credit scoring, see, e.g. Szepannek (2017). The standard way to tackle mixed-type data clustering problems in R is to use either (1) Gower distance (Gower, 1971) via the **gower** package (van der Loo, 2017) or the `daisy(method = "gower")` in the **cluster** package (Maechler et al., 2018); or (2) Hierarchical clustering through `hclust()` or the `agnes()` function in **cluster**. Recent innovations include the package **CluMix** (Hummel et al., 2017), which combines both Gower distance and hierarchical clustering with some functions for visualization. As this approach requires computation of distances between any two observations, it is not feasible for large data sets. The package **flexclust** (Leisch, 2006) offers a flexible framework for k-centroids clustering through the function `kcca()` which allows for arbitrary distance functions. Among the currently pre-implemented `kccaFamilies`, there is no distance measure for mixed-type data. Alternative approaches based on expectation-maximization are given by the function `flexmixedruns()` in the **fpc** package (Hennig, 2018) and the package **clustMD** (McParland, 2017). Both require the variables in the data set to be ordered according to their data type, and that categorical variables to be preprocessed into integers. The clustMD algorithm (McParland and Gormley, 2016) also allows ordinal variables but is quite computationally intensive. The **kamila** package (Foss and Markatou, 2018) implements the KAMILA clustering algorithm which uses a kernel density estimation technique in the continuous domain, a multinomial model in the categorical domain, and the Modha-Spangler weighting of variables in which categorical variables have to be transformed into indicator variables in advance (Modha and Spangler, 2003).

Recently, more algorithms for clustering mixed-type data have been proposed in the literature (Amir and Dey, 2007; Dutta et al., 2012; Foss et al., 2016; He et al., 2005; HajKacem et al., 2016; Ji et al., 2012, 2013, 2015; Lim et al., 2012; Liu et al., 2017; Pham et al., 2011). Many of these are based on the idea of Huang's k-prototypes algorithm (Huang, 1998). The rest of this paper describes the R package **clustMixType** (Szepannek, 2018), which provides up to the author's knowledge the first implementation of this algorithm in R. The k-modes algorithm (Huang, 1997a) has been implemented in the package **klaR** (Weihs et al., 2005; Roever et al., 2018) for purely categorical data, but not for the mixed-data case. The rest of the paper is organized as follows: A brief description of the algorithm is followed by the functions in the **clustMixType** package. Some extensions to the original algorithm are discussed and as well as a worked example application.

k-prototypes clustering

The k-prototypes algorithm belongs to the family of partitional cluster algorithms. Its objective function is given by:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij} d\left(x_i, \mu_j\right), \tag{1}$$

where $x_i, i = 1, \ldots, n$ are the observations in the sample, $\mu_j, j = 1, \ldots, k$ are the cluster prototype observations, and u_{ij} are the elements of the binary partition matrix $U_{n \times k}$ satisfying $\sum_{j=1}^{k} u_{ij} = 1, \forall i$.

The distance function is given by:

$$d(x_i, \mu_j) = \sum_{m=1}^{q} \left(x_i^m - \mu_j^m\right)^2 + \lambda \sum_{m=q+1}^{p} \delta\left(x_i^m, \mu_j^m\right). \tag{2}$$

where m is an index over all variables in the data set where the first q variables are numeric and the remaining $p - q$ variables are categorical. Note that $\delta(a,b) = 0$ for $a = b$ and $\delta(a,b) = 1$ for $a \neq b$, and $d()$ corresponds to weighted sum of Euclidean distance between two points in the metric space and simple matching distance for categorical variables (i.e. the count of mismatches). The trade off between both terms can be controlled by the parameter λ which has to be specified in advance as well as the number of clusters k. For larger values of λ, the impact of the categorical variables increases. For $\lambda = 0$, the impact of the categorical variables vanishes and only numeric variables are taken into account, just as in traditional k-means clustering.

The algorithm iterates in a manner similar to the k-means algorithm (MacQueen, 1967) where for the numeric variables the mean and the categorical variables the mode minimizes the total within cluster distance. The steps of the algorithm are:

1. Initialization with random cluster prototypes.
2. For each observation do:
   (a) Assign observations to its closest prototype according to $d()$.
   (b) Update cluster prototypes by cluster-specific means/modes for all variables.
3. As long as any observations have swapped their cluster assignment in 2 or the maximum number of iterations has not been reached: repeat from 2.

k-prototypes in R

An implementation of the k-prototypes algorithm is given by the function

```
kproto(x, k, lambda = NULL, iter.max = 100, nstart = 1, na.rm = TRUE)
```

where

- `x` is a data frame with both numeric and factor variables. As opposed to other existing R packages, the factor variables do not need to be preprocessed in advance and the order of the variables does not matter.
- `k` is the number of clusters which has to be pre-specified. Alternatively, it can also be a vector of observation indices or a data frame of prototypes with the same columns as `x`. If ever at the initialization or during the iteration process identical prototypes do occur, the number of clusters will be reduced accordingly.
- `lambda` > 0 is a real valued parameter that controls the trade off between Euclidean distance for numeric variables and simple matching distance for factor variables for cluster assignment. If no λ is specified the parameter is set automatically based on the data and a heuristic using the function `lambdaest()`. Alternatively, a vector of length `ncol(x)` can be passed to `lambda` (cf. Section on Extensions to the original algorithm).
- `iter.max` sets the maximum number of iterations, just as in `kmeans()`. The algorithm may stop prior to `iter.max` if no observations swap clusters.
- `nstart` may be set to a value > 1 to run k-prototypes multiple times. Similar to k-means, the result of k-prototypes depends on its initialization. If `nstart` > 1, the best solution (i.e. the one that minimizes E) is returned.
- Generally, the algorithm can deal with missing data but as a default `NA`s are removed by `na.rm = TRUE`.

Two additional arguments, `verbose` and `keep.data`, can control whether information on missing values should be printed and whether the original data should be stored in the output object. The `keep.data=TRUE` option is required for the default call to the `summary()` function, but in case of large `x`, it can be set to `FALSE` to save memory.

The output is an object of class `"kproto"`. For convenience, the elements are designed to be compatible with those of class `"kmeans"`:

- `cluster` is an integer vector of cluster assignments
- `centers` stores the prototypes in a data frame

- `size` is a vector of corresponding cluster sizes
- `withinss` returns the sum over all within cluster distances to the prototype for each cluster
- `tot.withinss` is their sum over all clusters which corresponds to the objective function E
- `dists` returns a matrix of all observations' distances to all prototypes in order to investigate the crispness of the clustering
- `lambda` and `iter` store the specified arguments of the function call
- `trace` lists the objective function E as well as the number of swapped observations during the iteration process

Unlike `"kmeans"`, the `"kproto"` class is accompanied corresponding `predict()` and `summary()` methods. The `predict.kproto()` method can be used to assign clusters to new data. Like many of its cousins, it is called by

```
predict(object, newdata)
```

The output again consists of two elements: a vector `cluster` of cluster assignments and a matrix `dists` of all observations' distances to all prototypes.

The investigation resulting from a cluster analysis typically consists of identifying the differences between the clusters, or in this specific case, those of the k prototypes. For practical applications besides the cluster sizes, it is of further interest to take into account the homogeneity of the clusters. For numeric variables, this can be done by calling the R function `summary()`. For categorical variables the representativity of the prototypes is given their frequency distribution obtained by `prop.table()`. The `summary.kproto()` method applies these methods to the variables conditional to the resulting clusters and returns a comparative results table of the clusters for each variable.

The summary is not restricted to the training data but it can further be applied to new data by calling `summary(object,data)` where `data` are new data that will be internally passed to the `predict()` method on the `object` of class `"kproto"`. If no new data is specified (default: `data = NULL`), the function requires `object` to contain the original data (argument `keep.data = TRUE`). In addition, a function

```
clprofiles(object, x, vars = NULL)
```

supports the analysis of the clusters by visualization of cluster profiles based on an `object` of class `"kproto"` and data `x`. Note that the latter may also have different variables compared to the original data, such as for profiling variables that were not used for clustering. As opposed to `summary.kproto()`, no new prediction is done but the cluster assignments of the `"kproto"` object given by `object$cluster` are used. For this reason, the observations in `x` must be the same as in the original data. Via the `vars` argument, a subset of variables can be specified either by a vector of indices or variable names. Note that `clprofiles()` is not restricted to objects of class `"kproto"` but can also be applied to other cluster objects as long as they are of a `"kmeans"`-like structure with elements `cluster` and `size`.

Extensions to the original algorithm

For unsupervised clustering problems, the user typically has to specify the impact of the specific variables on the desired cluster solution which is controlled by the parameter λ. Generally, small values $\lambda \sim 0$ emphasize numeric variables and will lead to results similar to standard k-means whereas larger values of λ lead to an increased influence of categorical variables. In Huang (1997b), the average standard deviation σ of the numeric variables is the suggested choice for λ and in some practical applications in the paper values of $\frac{1}{3}\sigma \leq \lambda \leq \frac{2}{3}\sigma$ are used. The function

```
lambdaest(x, num.method = 1, fac.method = 1, outtype = "numeric")
```

provides different data based heuristics for the choice of λ: The average variance σ^2 (`num.method = 1`) or standard deviation σ (`num.method = 2`) over all numeric variables is related to the average concentration h_{cat} of all categorical variables. We compute h_{cat} by averaging either $h_m = 1 - \sum_c p_{mc}^2$ (`fac.method = 1`) or $h_m = 1 - \max_c p_{mc}$ (`fac.method = 2`) over all variables m where c are the categories of the factor variables. We set $\lambda = \frac{\sigma^t}{h_{cat}}, t \in \{1,2\}$ as a user-friendly default choice to prevent over-emphasizing either numeric or categorical variables. If `kproto()` is called without specifying `lambda`, the parameter is automatically set using `num.method = fac.method = 1`. Note that this should be considered a starting point for further analysis; the explicit choice of λ should be done carefully based on the application context.

Originally, λ is real-valued, but in order to up- or downweight the relevance of single variables in a specific application context, the function `kproto()` is extended to accept vectors as input where each

cluster	numeric	categorical
1	+	+
2	+	-
3	-	+
4	-	-

Table 1: Separation of clusters in the example.

element corresponds to a variable specific weight, λ_m. The formula for distance computation changes to:

$$d(x_i, p_j) = \sum_{m=1}^{q} \lambda_m \left(x_i^m - \mu_j^m\right)^2 + \sum_{m=q+1}^{p} \lambda_m \delta \left(x_i^m, \mu_j^m\right). \qquad (3)$$

Note that the choice of λ only affects the assignment step for the observations but not the computation of the prototype given a cluster of observations. By changing the `outtype` argument into a vector, the function `lambdaest()` returns a vector of λ_ms. In order to support a user-specific definition of λ based on the variables' variabilities, `outtype = "variation"` returns a vector of original values of variability for all variables in terms of the quantities described above.

An issue of great practical relevance is the ability of an algorithm to deal with missing values which can be solved by an intuitive extension of k-prototypes. During the iterations, cluster prototypes can be updated by ignoring `NA`s: Both means for numeric variables as well as modes for factors can be computed based on the available data. Similarly, distances for cluster assignment can be computed for each observation based on the available variables only. This not only allows cluster assignment for observations with missing values, but already takes these observations into account when the clusters are formed. By using `kproto()`, this can be obtained by setting the argument `na.rm = FALSE`. Note that in practice, this option should be handled with care unless the number of missing values is very small. The representativeness of a prototype might become questionable if its means and modes do not represent the major part of the cluster's observations.

Finally, a modification of the original algorithm presented in Section k-prototypes clustering allows for vector-wise computation of the iteration steps, reducing computation time. The update of the prototypes is not done after each new cluster assignment, but once each time the whole data set has been reassigned. The modified k-prototypes algorithm consists of the following steps:

1. Initialization with random cluster prototypes.
2. Assign all observations to its closest prototype according to $d()$.
3. Update cluster prototypes.
4. As long as any observations have swapped their cluster assignment in 2 or the maximum number of iterations has not been reached: repeat from 2.

Example

As an example, data `x` with two numeric and two categorical variables are simulated according to the documentation in `?kproto`: Using `set.seed(42)`, four clusters $j = 1, \ldots, 4$ are designed such that two pairs can be separated only by their numeric variables and the other two pairs only by their categorical variables. The numeric variables are generated as normally distributed random variables with cluster specific means $\mu_1 = \mu_2 = -\mu_3 = -\mu_4 = \Phi^{-1}(0.9)$ and the categorical variables have two levels (A and B) each with a cluster specific probability $p_1(A) = p_3(A) = 1 - p_2(A) = 1 - p_4(A) = 0.9$. Table 1 summarizes the clusters. It can be seen that both numeric and categorical variables are needed in order to identify all four clusters.

Given the knowledge that there are four clusters in the data, a straightforward call for k-prototypes clustering of the data will be:

```
kpres <- kproto(x = x, k = 4)
kpres                  # output 1
summary(kpres)         # output 2
library(wesanderson)
par(mfrow=c(2,2))
clprofiles(kpres, x, col = wes_palette("Royal1", 4, type = "continuous"))  # figure 1
```

The resulting output is of the form:

```
# Output 1:

Numeric predictors: 2
Categorical predictors: 2
Lambda: 5.52477

Number of Clusters: 4
Cluster sizes: 100 95 101 104
Within cluster error: 332.909 267.1121 279.2863 312.7261

Cluster prototypes:
    x1 x2         x3        x4
92   A  A  1.4283725  1.585553
54   A  A -1.3067973 -1.091794
205  B  B -1.4912422 -1.654389
272  B  B  0.9112826  1.133724

# Output 2 (only for variable x1 and x3):

x1

cluster     A     B
      1 0.890 0.110
      2 0.905 0.095
      3 0.069 0.931
      4 0.144 0.856

-----------------------------------------------------------------

x3
     Min. 1st Qu.  Median    Mean 3rd Qu.   Max.
1 -0.7263  0.9314  1.4080  1.4280  2.1280 4.5110
2 -3.4200 -1.9480 -1.3170 -1.3070 -0.6157 2.2450
3 -4.2990 -2.0820 -1.4600 -1.4910 -0.7178 0.2825
4 -1.5300  0.2788  0.9296  0.9113  1.5000 3.1480

-----------------------------------------------------------------
```

The first two as well as the last two cluster prototypes share the same mode in the factor variables but they can be distinguished by their location with respect to the numeric variables. For clusters 1 & 4 (as opposed to 2 & 3), it is vice versa. Note that the order of the identified clusters is not necessarily the same as in cluster generation. Calling `summary()` and `clprofiles()` provides further information on the homogeneity of the clusters. A color palette can be passed to represent the clusters across the different variables for the plot; here it is taken from the package **wesanderson** (Ram and Wickham, 2018).

By construction, taking into account either only numeric (k-means) or only factor variables (k-modes) will not be able to identify the underlying cluster structure without further preprocessing of the data in this example. A performance comparison using the Rand index (Rand, 1971) as computed by the package **clusteval** (Ramey, 2012) results in rand indices of 0.728 (k-means) and 0.733 (k-modes). As already seen above, the prototypes as identified by **clustMixType** do represent the true clusters quite well, as the corresponding Rand index improves to 0.870.

```
library(klaR)
library(clusteval)

kmres  <- kmeans(x[,3:4], 4) # kmeans using numerics only
kmores <- kmodes(x[,1:2], 4) # kmodes using factors only

cluster_similarity(kpres$cluster, clusid, similarity = "rand")
cluster_similarity(kmres$cluster, clusid, similarity = "rand")
cluster_similarity(kmores$cluster, clusid, similarity = "rand")
```

The runtime of `kproto()` is linear in the number of observations (Huang, 1997b) and thus it is also applicable to large data sets. Figure 2 (left) shows the behaviour of the runtime for 50 repeated simulations of the example data with an increased number of variables (half of them numeric and

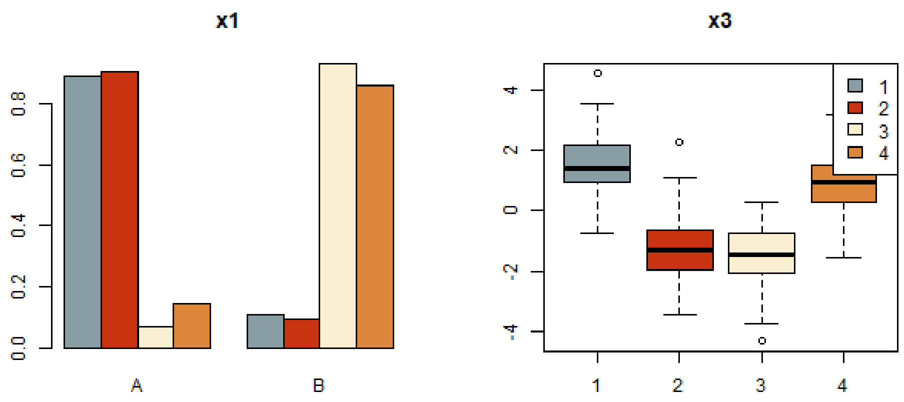


Figure 1: Cluster profiles for variables `x1` and `x3`.

half of them categorical). It is possible to run `kproto()` for more than hundred variables which is far beyond most practical applications where human interpretation of the resulting clusters is desired. Note that **clustMixType** is written in R and currently no C++ code is used to speed up computations which could be a subject of future work.

In order to determine the appropriate number of clusters for a data set, we can use the standard scree test. In this case, the objective function E is given by the output's `tot.withinss` element. The `kproto()` function is run multiple times for varying numbers of clusters (but fixed λ) and the number of clusters is chosen as the minimum k from whereon no strong improvements of E are possible. In Figure 2 (right), an elbow is visible at the correct number of clusters in the sample data; recall that we simulatd from four clusters. Note that from a practitioner's point of view, an appropriate solution requires clusters that are well represented by their prototypes. For this reason, the choice of the number of clusters should further take into account a homogeneity analysis of the clusters as returned by `summary()`, `clprofiles()` or the `withinss` element of the `"kproto"` output.

```
Es <- numeric(10)
for(i in 1:10){
        kpres <- kproto(x, k = i, nstart = 5)
        Es[i] <- kpres$tot.withinss
}
plot(1:10, Es, type = "b", ylab = "Objective Function", xlab = "# Clusters",
     main = "Scree Plot")  # figure 2
```

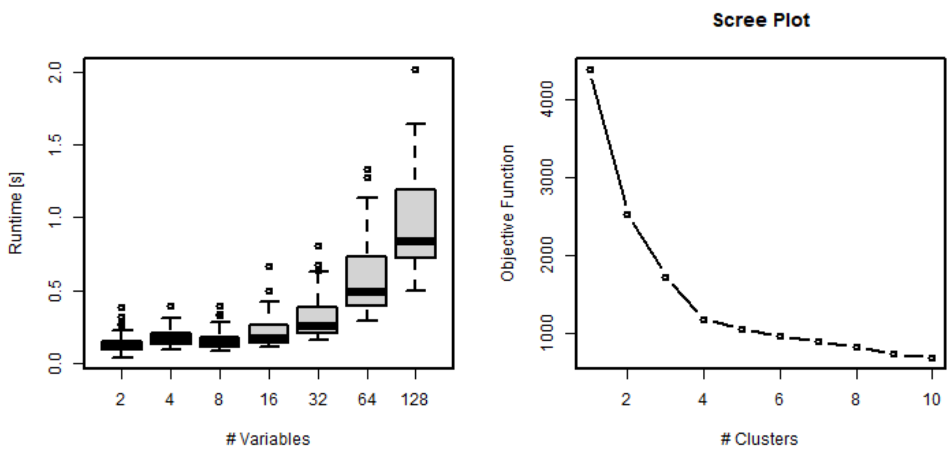


Figure 2: Runtime for increasing number of variables (left) and screeplot (right).

Summary

The **clustMixType** package provides a user-friendly way for clustering mixed-type data in R given by the k-prototypes algorithm. As opposed to other packages, no preprocessing of the data is necessary, and in contrast to standard hierarchical approaches, it is not restricted to moderate data sizes. Its application requires the specification of two hyperparameters: the number of clusters k as well as a second parameter λ that controls the interplay of the different data types for distance computation. As an extension to the original algorithm, the presented implementation allows for a variable-specific choice of λ and can deal with missing data. Furthermore, with regard to business purposes, functions for profiling a cluster solution are presented.

This paper is based on **clustMixType** version 0.1-36. Future work may focus on the development of further guidance regarding the choice of the parameter λ, such as using stability considerations (cf. Hennig, 2007), or the investigation of possible improvements in computation time by integrating **Rcpp** (Eddelbuettel et al., 2018; Eddelbuettel and François, 2011).

Bibliography

A. Amir and L. Dey. A k-mean clustering algorithm for mixed numeric and categorical data. *Data and Knowledge Engineering*, 63:88–96, 2007. doi: 10.1016/j.datak.2007.03.016. [p200]

D. Dutta, P. Dutta, and J. Sil. Data clustering with mixed features by multi objective genetic algorithm. In *12th International Conference on Hybrid Intelligent Systems (HIS)*, pages 336 – 341, 2012. doi: 10.1109/HIS.2012.6421357. [p200]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08. URL `http://www.jstatsoft.org/v40/i08/`. [p206]

D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2018. URL `https://CRAN.R-project.org/package=Rcpp`. R package version 0.12.18. [p206]

A. Foss and M. Markatou. *kamila: Methods for Clustering Mixed-Type Data*, 2018. URL `https://CRAN.R-project.org/package=kamila`. R package version 0.1.1.2. [p200]

A. Foss, M. Markatou, B. Ray, and A. Heching. A semiparametric method for clustering mixed data. *Machine Learning*, 105(3):419–458, 2016. doi: 10.1007/s10994-016-5575-7. [p200]

J. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27:857–874, 1971. doi: 10.2307/2528823. [p200]

M. A. B. HajKacem, C.-E. B. N'cir, and N. Essoussi. An accelerated mapreduce-based k-prototypes for big data. In P. Milazzo, D. Varro, and M. Wimmer, editors, *Software Technologies: Applications and Foundations 2016, Springer LNCS 9946*, pages 13–25, 2016. doi: 10.1007/978-3-319-50230-4_2. [p200]

Z. He, X. Xu, and S. Deng. A cluster ensemble method for clustering categorical data. *Information Fusion*, 6, 2005. doi: 10.1016/j.inffus.2004.03.001. [p200]

C. Hennig. Cluster-wise assessment of cluster stability. *Computational Statistics and Data Analysis*, 52: 258–271, 2007. doi: 10.1016/j.csda.2006.11.025. [p206]

C. Hennig. *fpc: Flexible Procedures for Clustering*, 2018. URL `https://CRAN.R-project.org/package=fpc`. R package version 2.1-11.1. [p200]

Z. Huang. A fast clustering algorithm to cluster very large categorical data sets in data mining. In *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 1 – 8, 1997a. [p200]

Z. Huang. Clustering large data sets with mixed numeric and categorical values. In *Proceedings of the First Pacific Asia Knowledge Discovery and Data Mining Conference, Singapore*, pages 21 – 34, 1997b. [p202, 204]

Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical variables. *Data Mining and Knowledge Discovery*, 2:283–304, 1998. doi: 10.1023/A:1009769707641. [p200]

M. Hummel, D. Edelmann, and A. Kopp-Schneider. *CluMix: Clustering and Visualization of Mixed-Type Data*, 2017. URL `https://CRAN.R-project.org/package=CluMix`. R package version 2.1. [p200]

J. Ji, W. Pang, C. Zhou, X. Han, and Z. Wang. A fuzzy k-prototype clustering algorithm for mixed numeric and categorical data. *Knowledge-Based Systems*, 30:129–135, 2012. doi: 10.1016/j.knosys.2012.01.006. [p200]

J. Ji, T. Bai, C. Zhou, C. Maa, and Z. Wang. An improved k-prototypes clustering algorithm for mixed numeric and categorical data. *Neurocomputing*, 120:590–596, 2013. doi: 10.1016/j.neucom.2013.04.011. [p200]

J. Ji, W. Pang, Y. Zheng, Z. Wang, Z. Ma, and L. Zhang. A novel cluster center initialization method for the k-prototypes algorithms using centrality and distance. *Applied Mathematics and Information Sciences*, 9:2933–2942, 2015. doi: 10.12785/amis/090621. [p200]

F. Leisch. A toolbox for k-centroids cluster analysis. *Computational Statistics and Data Analysis*, 51(2): 526–544, 2006. doi: 10.1016/j.csda.2005.10.006. [p200]

J. Lim, J. Jun, S. H. Kim, and D. McLeod. A framework for clustering mixed attribute type datasets. In *Proceeding of the fourth International Conference on Emerging Databases (EDB)*, 2012. [p200]

S.-H. Liu, B. Zhou, D. Huang, and L. Shen. Clustering mixed data by fast search and find of density peaks. *Mathematical Problems in Engineering*, 2017:1–7, 2017. doi: 10.1155/2017/5060842. [p200]

J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. Le Cam and J. Neyman, editors, *Proc. 5th Berkeley Symp. Math Stat and Prob*, pages 281 – 297, 1967. [p201]

M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2018. URL https://CRAN.R-project.org/package=cluster. R package version 2.0.7-1. [p200]

D. McParland. *clustMD: Model Based Clustering for Mixed Data*, 2017. URL https://CRAN.R-project.org/package=clustMD. R package version 1.2.1. [p200]

D. McParland and I. Gormley. Model based clustering for mixed data: clustmd. *Advances in Data Analysis and Classification*, 10(2):155–170, 2016. doi: 10.1007/s11634-016-0238-x. [p200]

D. Modha and S. Spangler. Feature weighting in k-means clustering. *Machine Learning*, 52(3):217–237, 2003. doi: 10.1023/A:1024016609528. [p200]

D. Pham, M. Suarez-Alvarez, and Y. I. Prostov. Random search with k-prototypes algorithm for clustering mixed datasets. *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, 467:2387–2403, 2011. doi: 10.1098/rspa.2010.0594. [p200]

K. Ram and H. Wickham. *wesanderson: A Wes Anderson Palette Generator*, 2018. URL https://CRAN.R-project.org/package=wesanderson. R package version 0.3.6. [p204]

J. A. Ramey. *clusteval: Evaluation of Clustering Algorithms*, 2012. URL https://CRAN.R-project.org/package=clusteval. R package version 0.1. [p204]

W. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971. doi: 10.2307/2284239. [p204]

C. Roever, N. Raabe, K. Luebke, U. Ligges, G. Szepannek, and M. Zentgraf. *klaR: Classification and visualization*, 2018. URL https://CRAN.R-project.org/package=klaR. R package version 0.6-14. [p200]

G. Szepannek. On the practical relevance of modern machine learning algorithms for credit scoring applications. *WIAS Report Series*, 29:88–96, 2017. doi: 10.20347/wias.report.29. [p200]

G. Szepannek. *clustMixType: k-Prototypes Clustering for Mixed Variable-Type Data*, 2018. URL https://CRAN.R-project.org/package=clustMixType. R package version 0.1-36. [p200]

M. van der Loo. *gower: Gower's Distance*, 2017. URL https://CRAN.R-project.org/package=gower. R package version 0.1.2. [p200]

C. Weihs, U. Ligges, K. Luebke, and N. Raabe. klar analyzing german business cycles. In D. Baier, R. Decker, and L. Schmidt-Thieme, editors, *Data Analysis and Decision Support*, pages 335–343, Berlin, 2005. Springer-Verlag. doi: 10.1007/3-540-28397-8_36. [p200]

Gero Szepannek
Stralsund University of Applied Sciences
Zur Schwedenschanze 15
18435 Stralsund
Germany
gero.szepannek@hochschule-stralsund.de

Stilt: Easy Emulation of Time Series AR(1) Computer Model Output in Multidimensional Parameter Space

by Roman Olson, Kelsey L. Ruckert, Won Chang, Klaus Keller, Murali Haran, and Soon-Il An

Abstract Statistically approximating or "emulating" time series model output in parameter space is a common problem in climate science and other fields. There are many packages for spatio-temporal modeling. However, they often lack focus on time series, and exhibit statistical complexity. Here, we present the R package **stilt** designed for simplified AR(1) time series Gaussian process emulation, and provide examples relevant to climate modelling. Notably absent is Markov chain Monte Carlo estimation – a challenging concept to many scientists. We keep the number of user choices to a minimum. Hence, the package can be useful pedagogically, while still applicable to real life emulation problems. We provide functions for emulator cross-validation, empirical coverage, prediction, as well as response surface plotting. While the examples focus on climate model emulation, the emulator is general and can be also used for kriging spatio-temporal data.

Introduction

Emulation of computer model behavior in parameter space is a challenging problem (Drignei et al., 2008; Higdon et al., 2008; Holden et al., 2010; Bhat et al., 2012; Olson et al., 2012; Sexton et al., 2012; Olson et al., 2013; Chang et al., 2014a,b). Often a limited number of runs are available and we desire to estimate model output at a variety of new parameter settings (Drignei et al., 2008; Holden et al., 2010; Bhat et al., 2012; Olson et al., 2012; Sexton et al., 2012; Olson et al., 2013; Chang et al., 2014b). Emulation is often used in the context of another common problem, input parameter estimation. Here, we desire to find probability distribution functions for model input parameters given observational data (e.g., Olson et al., 2012; Chang et al., 2014b). While model output is often multi-dimensional, we restrict the discussion here to time series although our approach can apply to vector output more generally.

Gaussian processes (Kennedy and O'Hagan, 2001; Rasmussen and Williams, 2006; Higdon et al., 2008; Rougier, 2008) are a very useful methodology for such emulation. Gaussian processes provide the best linear unbiased prediction under highly general conditions (Stein, 1999). This methodology assumes that model response is a smooth function of parameter settings. Central to the method is formulation of the covariance function, which quantifies the covariance between model output as a function of distance in each model input parameter. This covariance is typically parametrized by parameters controlling, for example, magnitude or the correlation ranges in each coordinate dimension. Associated with emulator parameters is a likelihood function given the model output. There are two main approaches: (i) find the "best" parameter setting that maximizes this likelihood, or (ii) find the full probability distribution of the parameters. The next step is prediction at new parameter settings. The former approach ignores the uncertainty in emulator parameters during the prediction, while the latter approach accounts for it. The prediction is probabilistic as it includes the predictive mean and the uncertainty around it.

There are many existing R packages to perform Gaussian process emulation for vector and time series output. In particular, they include **gstat** (Gräler et al., 2016), **mlegp** (Dancik and Dorman, 2008; Dancik, 2013), **spBayes** (Finley et al., 2015), **ramps** (Smith et al., 2008), **spTimer** (Bakar and Sahu, 2015a,b), and **RandomFields** (Schlather et al., 2015). These spatio-temporal packages are usually general and do not focus on time series specifically. Moreover, they are usually presented in context of spatial interpolation ("kriging") rather than emulation. They may provide flexible functionality in terms of covariance functions (**RandomFields**), handling replicates (**mlegp**), dealing with missing observations (**spTimer**), or considering both areal as well as point observations (**ramps**). The procedure for fitting emulator parameters differs between packages. Some packages use maximum likelihood or another form of optimization to find the best parameters, while others use Bayesian analysis.

Due to their statistical complexity, the broad scientific community may find it challenging to use existing software. Particularly, the concept of Markov chain Monte Carlo (MCMC) employed in some work may be unfamiliar to some scientists. In addition, a large variety of modeling or prior choices may leave a user struggling with which model or prior they should employ and why. Many packages appear to be tailored to a statistical audience, and elaborate statistical terminology may be beyond the grasp for some scientists. Second, the packages do not usually focus on the spatio-temporal output ubiquitous in computer modelling. Thus, scientists may invest substantial time in finding the right function among the plethora available. Finally, there is an issue of terminology: many scientists may not realize that "spatio-temporal modeling" may disguise the same technique as "emulation" and hence, may be oblivious to the applicability of these packages.

We introduce the R package **stilt** version 1.3.0 for simplified Gaussian Process AR(1) time series emulation with a focus on climate modeling (Olson et al., 2017). The package is available on Comprehensive R Archive Network (CRAN). The main differences from prior approaches are: (i) a simplified framework with fewer modeling choices and no MCMC that is still applicable to real-life problems, and (ii) a focus on emulation of time-series in parameter space. Specifically, there is only one function to fit the emulator to model output and only one function for prediction. Statistical modeling accounts for random noise in the model output through the nugget term, and also allows for user-chosen linear terms in model parameters and/or time. Mathematically, this is the first public implementation of a separable covariance model of Rougier (2008). In this model, space-time covariance between two locations is a product of a space covariance and a time covariance term. This allows the use of matrix algebra to achieve considerable computational gains. We include useful utility functions for extensive cross-validation, 2D response surface plotting, and empirical 95% prediction interval coverage. In three examples, we apply **stilt** to emulation in spaces ranging from one- to five-dimensional. Note that whilst the software is for emulation of time series output in parameter space, the package is general and can be used for interpolation of observations (or model output) in geographical coordinates (e.g., Cressie and Johannesson, 2008; Jones et al., 2009; Hansen et al., 2010; Bhat et al., 2012; Hirahara et al., 2014).

In the rest of the paper we first outline the statistical approach, then give three examples in 1D, 2D, and 5D parameter space, then present some concluding remarks. We provide technical details in the Appendix.

Statistical approach

Overview of Approach

Here, we briefly describe our statistical approach. Our statistical model includes optional linear terms in parameters and/or time, a smooth Gaussian process, and a purely random nugget term. The Gaussian process represents smooth non-linear effects of parameters or time on the model output. The nugget term represents purely random effects. For climate model output, this corresponds to internal climate variability.

The first step is fitting the Gaussian process statistical model to the computer model output. Specifically, the statistical model has several parameters. They control the linear slopes in parameters and time, the correlation ranges of the field in each of the input parameters, the temporal autocorrelation, the overall magnitude of the covariance, and the nugget strength. Associated with the emulator parameters is the likelihood which quantifies how likely the emulator parameter values are given the model output. During the emulator fitting, we vary the parameters so that this likelihood is maximized to obtain the optimized emulator. We use this optimized emulator for prediction. The prediction follows the standard Gaussian process theory. The rest of the section describes the details of the statistical methodology.

Statistical model

We base our implementation on standard Gaussian process theory (Cressie, 1993; Stein, 1999; Rasmussen and Williams, 2006). Consider the case of interpolating spatio-temporal model output of a perturbed parameter model ensemble in parameter space. Let $y_{i,j} \in R$ be physical model output at a parameter setting $\boldsymbol{\theta_i}$ and a time t_j. Time values form an n-dimensional regularly spaced vector $\boldsymbol{t} = (t_1, \ldots, t_n)^T$.

Each parameter setting is an m-dimensional vector: $\boldsymbol{\theta_i} = (\theta_{1,i}, \ldots, \theta_{m,i})$. For all p model runs, the parameter settings $\boldsymbol{\theta_i}$ form a $p \times m$ parameter matrix Θ. Let $\boldsymbol{y_j} = (y_{1,j}, \ldots, y_{p,j})^T$ be a p-dimensional vector of model outputs for all p parameters for time t_j. Consecutively, the stacked $pn \times 1$ column matrix of all model output for times from 1 through n is $\mathbf{Y} = (\boldsymbol{y_1}^T, \ldots, \boldsymbol{y_n}^T)^T$. Associated with $\mathbf{Y}$ is the $pn \times (m+1)$ design matrix $\mathbf{D}$. Its columns represent parameters and time, whereas rows correspond to elements of $\mathbf{Y}$. We calculate the design matrix $\mathbf{D}$ as:

$$\mathbf{D} = \left(\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{n\times 1} \otimes \Theta \qquad \mathbf{t} \otimes \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{p\times 1} \right), \tag{1}$$

where $\otimes$ is the Kronecker product. We model the output as a Gaussian process such that

$$\mathbf{Y} \sim N(\boldsymbol{\mu}_{\boldsymbol{\beta}}, \Sigma(\boldsymbol{\xi}_{\boldsymbol{y}})), \tag{2}$$

where $\boldsymbol{\mu}_{\boldsymbol{\beta}}$ is a mean function that is either constant or linear in any combination of parameters and/or time, and $\boldsymbol{\xi}_{\boldsymbol{y}}$ is a vector of covariance matrix parameters. We assume that $\boldsymbol{\mu}_{\boldsymbol{\beta}} = \mathbf{X}\boldsymbol{\beta}$, where $\boldsymbol{\beta}$ is a column vector of regression coefficients, and $\mathbf{X}$ is a matrix of covariates. It always includes a column of ones as its first column to represent the intercept. Depending on the number of regressors, it can additionally have corresponding columns of the design matrix $\mathbf{D}$. As an example, for a mean function that is linear in time, $\boldsymbol{\beta}$ has dimensions of 2×1, and $\mathbf{X}$ is $pn \times 2$:

$$\mathbf{X} = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_n \end{bmatrix}_{n\times 2} \otimes \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{p\times 1.} \tag{3}$$

Under the assumption of separability (see Rougier, 2008), we can represent the covariance matrix Σ as a Kronecker product of a separate covariance matrix in time Σ_t and in parameters Σ_θ:

$$\Sigma = \Sigma_t \otimes \Sigma_\theta. \tag{4}$$

This means that the covariance between any two locations in time and parameter space is a product of the time covariance term, and the parameter covariance term. This is a different approach to speed up the computation, compared to performing singular value decomposition (SVD) of model output (Dancik and Dorman, 2008; Dancik, 2013; Higdon et al., 2008; Chang et al., 2014b). We assume that the time covariance matrix Σ_t $(n \times n)$ has an AR(1) structure. AR(1) dependence in time is a feature of many environmental processes (e.g., Hasselmann, 1976; Keller and McInerney, 2008; Olson et al., 2013). To avoid identifiability issues, we do not use any multipliers for this matrix. Specifically, its (j,k) element is:

$$\varsigma_{t,jk} = \frac{\rho^{|t_j - t_k|}}{1-\rho^2}, \tag{5}$$

where ρ is the lag-1 autocorrelation parameter. We assume that the parameter covariance Σ_θ $(p \times p)$ is squared exponential, as in the package **mlegp** (Dancik and Dorman, 2008; Dancik, 2013). The squared exponential covariance function is frequently used in computer model emulation, as computer model outputs can be often represented using a highly smooth Gaussian process. The (i,j) element of Σ_θ is:

$$\varsigma_{\theta,ij} = \kappa \exp\Big(-\sum_{k=1}^{m} \frac{|\theta_{k,i} - \theta_{k,j}|^2}{\phi_k^2}\Big) + \zeta 1(i=j) \tag{6}$$

where κ is the partial sill, ζ is the nugget, and ϕ_k is the range parameter for the k^{th} model input parameter. The range parameters form a vector $\boldsymbol{\phi} = \phi_1, \ldots, \phi_m$. We construct the total covariance matrix $(np \times np)$ as:

$$\Sigma = \begin{bmatrix} \varsigma_{t,11}\Sigma_\theta & \cdots & \varsigma_{t,1n}\Sigma_\theta \\ \vdots & \ddots & \vdots \\ \varsigma_{t,n1}\Sigma_\theta & \cdots & \varsigma_{t,nn}\Sigma_\theta \end{bmatrix}. \tag{7}$$

Hence, the covariance parameters are $\boldsymbol{\xi}_{\boldsymbol{y}} = (\rho, \kappa, \boldsymbol{\phi}, \zeta)^T$. The emulator parameters are $\boldsymbol{\psi} = (\boldsymbol{\beta}^T, \boldsymbol{\xi}_{\boldsymbol{y}}{}^T)^T$. The actual number of emulator parameters will be different depending on the number of model parameters that the ensemble varies and on the number of covariates.

Estimating emulator parameters

We can write the log-likelihood for the model output $\mathbf{Y}$ given the emulator parameters $\boldsymbol{\psi}$ as (see, e.g., Rasmussen and Williams, 2006):

$$\ln L(\mathbf{Y}|\boldsymbol{\psi}) = -\frac{1}{2}(\mathbf{Y}-\boldsymbol{\mu}_\beta)^T\Sigma^{-1}(\mathbf{Y}-\boldsymbol{\mu}_\beta) - \frac{1}{2}\ln|\Sigma| - \frac{np}{2}\ln 2\pi. \tag{8}$$

Under the uniform priors for the emulator parameters,

$$L(\boldsymbol{\psi}|\mathbf{Y}) \propto L(\mathbf{Y}|\boldsymbol{\psi}). \tag{9}$$

Consequently, maximizing the log-likelihood for the model output also maximizes the likelihood for the parameters. Note that this likelihood evaluation involves a computationally expensive inverse of an $np \times np$ matrix. However, we can reduce the dimension of the inverted matrices to $p \times p$ and $n \times n$, and speed up the computation (see Appendix). Thus, the computational cost of inversion becomes $O(p^3)$ or $O(n^3)$, as opposed to $O([pn]^3)$. Further computational savings accrue when calculating $|\Sigma|$, considering that Σ is a Kronecker product of two positive definite matrices (see Appendix).

We optimize the emulator parameters $\boldsymbol{\psi}$ by maximizing the likelihood function over a reasonable parameter range using a local optimization routine. Specifically, **stilt** uses the `nlminb` function which calls FORTRAN code (Gay, 1990). In the package, there is an option to either fix $\boldsymbol{\beta}$ parameters at their multiple linear regression estimates or to optimize them along with other emulator parameters.

Prediction

We are interested in predicting model output for all times for a new parameter setting $\boldsymbol{\theta}^*$. We denote the n-dimensional vector output as $\boldsymbol{y}^* = (y_{\theta^*,1}, \ldots, y_{\theta^*,n})^T$. Associated with the prediction parameter setting is an $n \times 1$ prediction design matrix $\mathbf{D}^*$ and a matrix of covariates $\mathbf{X}^*$ evaluated at predictions points. It is constructed similarly to $\mathbf{X}$. For a mean function that is linear in time, $\mathbf{X}^*$ is:

$$\mathbf{X}^* = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_n \end{bmatrix}_{n\times 2}. \tag{10}$$

The prediction is given by the following multivariate normal distribution (Rasmussen and Williams, 2006):

$$\boldsymbol{y}^* \sim N(\boldsymbol{\mu}^*_\beta, \Sigma^*). \tag{11}$$

Here,

$$\boldsymbol{\mu}^*_\beta = \mathbf{X}^*\boldsymbol{\beta} + (\Sigma_t \otimes \Sigma_{\theta^*\theta})\Sigma^{-1}(\mathbf{Y}-\boldsymbol{\mu}_\beta), \tag{12}$$

where $\Sigma_{\theta^*\theta}$ is a $1 \times p$ cross-covariance matrix between the prediction parameter setting and all the ensemble parameters settings. For this matrix, we use the same covariance function as for Σ_θ. To evaluate the mean function, we need the inverse of the Σ matrix ($np \times np$). However, using matrix algebra (Golub and Van Loan, 1996; Rougier, 2008), we can reduce the dimension to $p \times p$ by writing $\boldsymbol{\mu}^*_\beta$ in the following way:

$$\boldsymbol{\mu}^*_\beta = \mathbf{X}^*\boldsymbol{\beta} + (\mathbf{I}_{n\times n} \otimes \Sigma_{\theta^*\theta}\Sigma_\theta^{-1})(\mathbf{Y}-\boldsymbol{\mu}_\beta). \tag{13}$$

The predictive covariance also requires an inversion of a $p \times p$ matrix only:

$$\Sigma^* = (\kappa+\zeta)\Sigma_t - \Sigma_t \otimes \Sigma_{\theta^*\theta}\Sigma_\theta^{-1}\Sigma^T_{\theta^*\theta}. \tag{14}$$

Prediction for many points uses the same inverse parameter covariance. Hence, we can calculate the inverse once and recycle it for prediction at many points. This further enhances computational savings.

Examples using stilt

Preparing model output

Here, we describe how to prepare model output for use with the **stilt** emulator. Two R list objects are required. These objects can be easily prepared by the user before the start of the analysis. The first, `mpars`, contains information on parameter settings in the ensemble. It has the following components:

- `par` is the actual matrix of parameter settings, with rows corresponding to parameter index, and columns to the model run index.
- `parnames` is a vector of parameter names corresponding to the rows of `par`,
- `parunits` is a vector of units.

The second object, `moutput` has information on model output. It also has several components, the most important being `out` which is the output model matrix with rows referencing time and columns referencing model run. Other elements contain metadata: `t` is the corresponding time vector, `tunuts` are time units, `outname` is the name of the modeled variable, and `outunits` are the corresponding units.

Simple 1D example

This and the following examples are based on running **stilt** version 1.3.0 on R version 3.3.3 using a 3 Ghz Intel core i5 16GB 2400 MHz DDR4 Macintosh 10.13.6 computer. The following versions were used for other required packages: **fields** (9.0), **maps** (3.3.0), **spam** (2.1-2), and **dotCall64** (0.9-5). The results were observed to differ slightly according to the programming environment and the operating system.

We first consider interpolating a toy model dataset consisting of a simple time-series output for one parameter with a total of 21 parameter settings and 11 time points. The output of this simple model as a function of time t and parameter θ is: $y = \sin(\theta)(1 + 2t + t^2)$. The model is evaluated for $\theta = (0, 1, \ldots, 20)$, and each run produces a time series for times $t = (0, 1, \ldots, 10)$. First, we can plot model output for all parameters:

```
R> data("Data.1D.model")
R> data("Data.1D.par")
R> plot(NA, xlim=c(0, 11), ylim=c(-150, 150), xlab="Time", ylab="Model output")
R> for (c in 1:21) {lines(Data.1D.model$t, Data.1D.model$out[,c])}
```

The output of this code is shown in Figure 1.

Now, we fit a Gaussian process emulator to these data. While we do not use any parameter covariances (`par.reg=FALSE`), we do use a linear time covariate (`time.reg=TRUE`). The optimization follows the default behavior of fixing the linear regressors at the multiple linear regression estimates. We select 100 as the starting values for both κ and ζ because this leads to reasonable optimization results.

```
R> emul1D = emulator(Data.1D.par, Data.1D.model, par.reg=FALSE, time.reg=TRUE,
kappa0=100, zeta0=100)
Initializing the emulator...

Initial regression parameters:
-0.665481 0.570413

Initial emulator likelihood is: -960.2755

Optimizing the emulator...
Obtaining emulator parameter ranges for optimization...

Relative tolerance to be used in optimization: 1e-10

Option 'fix.betas': during optimization beta parameters are going to be
fixed at the following values:
-0.665481 0.570413

-----------------------------------
Starting parameter optimization...
------------------------------------
  0:     960.27550: 0.900000  100.000  100.000  10.0000
  1:     947.88498: 0.903188  72.2395  233.781  10.1709
  2:     898.20175: 0.957801 0.00240862  151.661  13.5627
  3:     889.24528: 0.977342 0.00240862  131.201  13.4426

 48:     464.48240: 0.982420  1076.06 0.00240862  3.93464
Optimization SUCCESSFUL! Optimization message below:
```

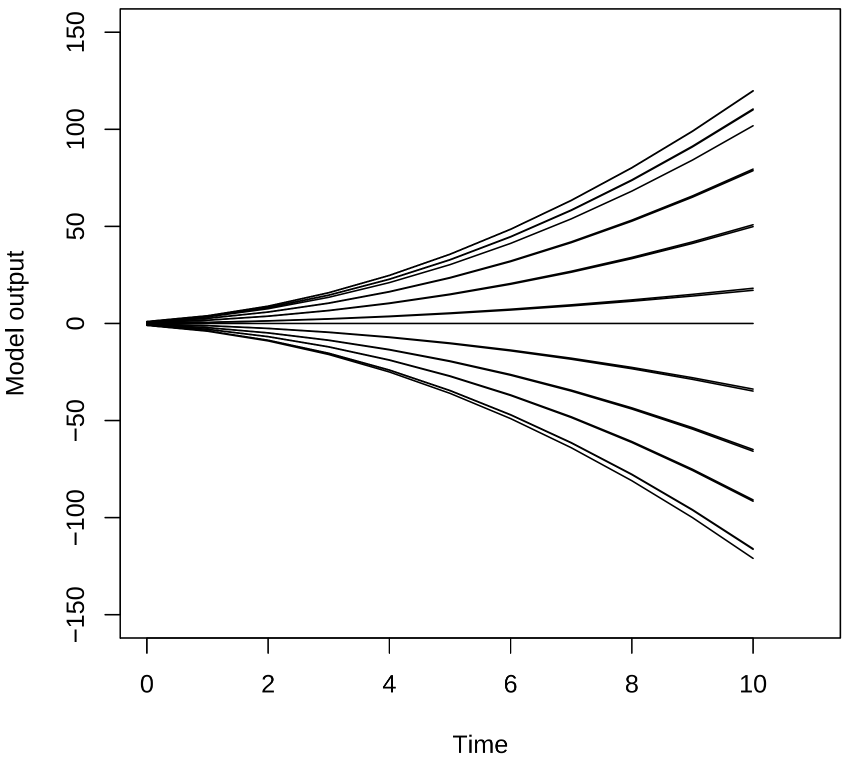


Figure 1: Sample time series output of the toy model for different parameter settings

```
relative convergence (4)

Final parameterss
rho kappa zeta phi
0.98242004 1076.05714589 0.00240862 3.93464218

48 iterations were performed
Final likelihood = -464.4824

CAUTION! The optimization might only find a local minimum.
```

The package informs us of key emulator parameter settings, and of optimization results. (For brevity, only the start and the end of optimization process are shown above.) The list of parameters to be optimized is ρ, κ, ζ, and ϕ_1 in this case. The final emulator is a highly autocorrelated process with a large partial sill, but a very low nugget. The emulator object `emul1D` is a list with many components, with a secondary custom `"emul"` class. The components include information on the data used to fit the emulator, optimized emulator parameters, some settings used during optimization, time and parameter covariance matrices, and the inverse of the parameter covariance matrix. Now, we validate the emulator using one-at-a-time cross-validation for time index 9 (close to the end of the time series). Specifically, we remove each parameter setting from the ensemble one at a time, and use the emulator to predict the output at the excluded parameter setting, given the output at other parameter settings.

```
R> test.all(emul1D, 9)
```

The emulator mean prediction shows a remarkable accuracy at predicting actual model response: the two fall almost perfectly on the 1:1 line of a reliability diagram (Figure 2). The emulator prediction error is much less than 1% for almost all runs (Figure 2). Note that the emulator does not extrapolate beyond the ensemble parameter range.

The emulator predicts at new parameter settings using the function `predict.emul`. This is im-

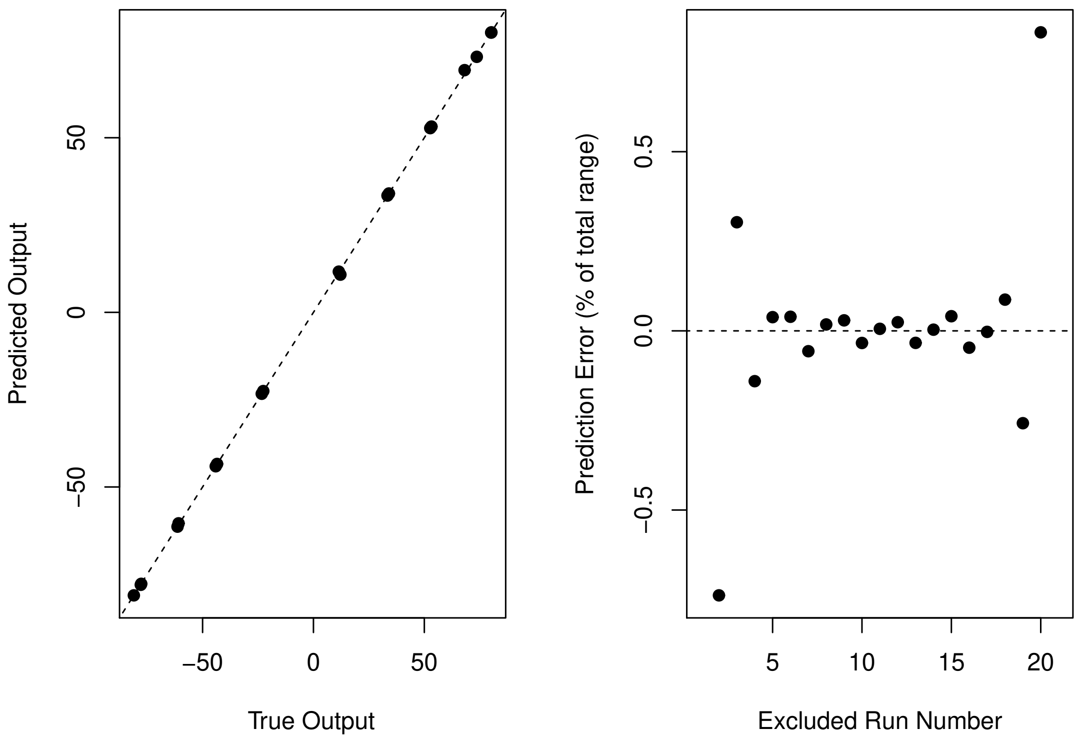


Figure 2: Toy example emulator one-at-a-time cross-validation results: (left) predicted vs. actual model output (black dots) and a 1:1 line; (right) relative prediction error as a function of the excluded model run number.

plemented using the generic S3 `predict` function that dispatches to the `predict.emul()` method for objects of class `"emul"`. This function returns an object with components `mean` and `covariance` representing the mean and the covariance of the prediction.

A more challenging test case: 2D Korean summer mean maximum temperature variability and change

Next, we consider Korean summer mean maximum temperature output from 29 Coupled Model Intercomparison Project phase 5 (CMIP5) climate models (Taylor et al., 2012) for years 2081-2100 for the RCP8.5 forcing scenario (Moss et al., 2010). All of the models and calculations used here follow Shin et al. (2018), however here we exclude the model IPSL_CM5A_LR from the analysis. Of interest is the relationship between the present-day (years 1973-2005) sample red noise properties for annual temperature time-series, and the future mean maximum summer temperature changes in these climate models. Hence, each model i is associated with $\boldsymbol{\theta_i} = (\sigma_i, \rho_i)$, where σ_i is sample innovation standard deviation, and ρ_i is sample first-order autocorrelation. In this case, $m = 2$, $p = 29$, and $n = 20$.

First, we load the relevant datasets and plot the temperature time series for all models.

```
R> data("Data.AR1Korea.model")
R> data("Data.AR1Korea.par")
R> mycolors = rainbow(29)
R> plot.default(NA, xlim=c(2081, 2100), ylim=c(0,10), xlab="Year",
            ylab="JJA Mean Max Temp Anomaly wrt. 1973-2005 [K]")
R> for (c in 1:29) {lines(Data.AR1Korea.model$t, Data.AR1Korea.model$out[,c],
                        col=mycolors[c])}
```

We note that the models show a considerable internal variability superimposed on a trend of slow warming (Figure 3). Next, we fit an emulator while also allowing for optimization of regression slopes

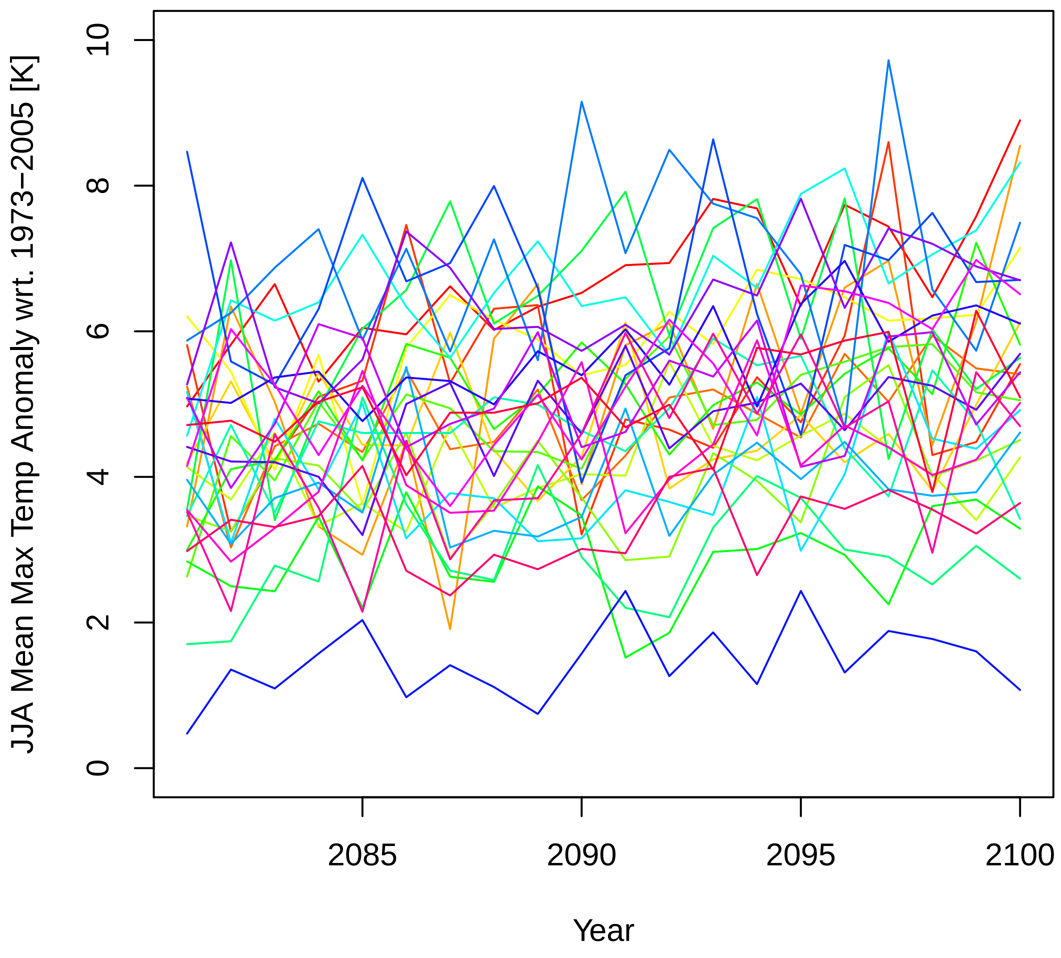


Figure 3: Korean summer mean maximum temperature change in years 2081-2100 with respect to period 1973-2005 for 29 CMIP5 climate models.

in innovation standard deviation and time. We also use a custom relative tolerance to illustrate how this parameter may be changed to fit user needs.

```
R> emul = emulator(Data.AR1Korea.par, Data.AR1Korea.model, par.reg=c(TRUE,
FALSE), time.reg=TRUE, kappa0=1, zeta0=1, myrel.tol=1E-9, fix.betas=FALSE)

Initializing the emulator...

Initial regression parameters:
-116.0626827 3.8544881 0.0563853

Initial emulator likelihood is: -937.479

Optimizing the emulator...
Obtaining emulator parameter ranges for optimization...

Relative tolerance to be used in optimization: 1e-09
```

```
------------------------------------
Starting parameter optimization...
-------------------------------------
 0: 937.479: 0.900000  1.00000  1.00000 -116.063  3.85449 0.0563853 0.322532 0.257
 1: 923.956: 0.899929 0.468880  1.36686 -116.087  3.85444 0.0563734 0.322768 0.257
 2: 901.460: 0.899874 9.24879e-05  1.05990 -116.096  3.85442 0.0563689 0.322900 0.257
 3: 900.363: 0.899860 0.0377996  1.17131 -116.068  3.85445 0.0563831 0.322900 0.257

54: 858.446: 0.616698 9.24879e-05  1.10326 -123.432 3.86546 0.0599039 6.45065 4.396
Optimization SUCCESSFUL! Optimization message below:

relative convergence (4)

Final parameterss
rho kappa zeta beta beta beta phi phi
0.616698 9.24879e-05  1.10326 -123.432 3.86546 0.0599039 6.45065 4.396

54 iterations were performed
Final likelihood = -858.4463

CAUTION! The optimization might only find a local minimum.
```

Note that we have formatted the output slightly to fit the page width.

We see that the initial multiple regression provides reasonable mean and parameter slopes. They are close to the final optimized results (fourth through sixth elements of the optimized parameter vector called "`Final parameters`" above). There is a considerable linear dependence on both time and innovation standard deviation. The non-linear part of the emulator has a strong random component, and a very weak Gaussian process component. This suggests no systematic dependence of the model output on the present-day autocorrelation. We can predict the temperature response at an arbitrary setting of the parameters using the `predict` function. We do this for $\sigma = 1$ and $\rho = 0$:

```
R> pred = predict(emul, c(1, 0))
R> plot.default(NA, xlim=c(2081, 2100), ylim=c(3.5,7.5), xlab="Year",
             ylab="JJA Mean Max Temp Anomaly wrt. 1973-2005 [K]")
R> lines(emul$t.vec, pred$mean)
R> lines(emul$t.vec, pred$mean + sqrt(diag(pred$covariance)), col="brown")
R> lines(emul$t.vec, pred$mean - sqrt(diag(pred$covariance)), col="brown")
```

The mean vector of the prediction is the `mean` component of `pred`, and the variance vector is composed of the diagonal entries of `covariance`. Figure 4 shows the predicted response, with the associated 1-std uncertainty. We note the linearity of the warming in time. This illustrates the ability of the emulator to identify fluctuations of temperature in the models around the linear trend as random, and not to include them into the emulated response.

An even more challenging test case: Five-dimensional ice sheet model output

Next, we consider a 5D emulator for the SICOPOLIS ice sheet model output (Greve, 1997; Greve et al., 2011) for Greenland ice mass loss relative to the year 2003. This is a perturbed parameter 100-member ensemble which varies five model parameters: flow enhancement factor, basal sliding factor, geothermal heat flux, snow positive degree-day (PDD) factor, and ice PDD factor. The future forcing scenario is that of a gradual temperature increase stabilizing at approximately 5 K warmer than present (Applegate et al., 2012). Output is available annually between years 1840 and 2500.

We load relevant data and fit an emulator to the ensemble using all five parameters and time as covariates. For computational expediency, we fix slope parameters at their multiple regression estimates.

```
R> data(Data.Sicopolis.par)
R> data(Data.Sicopolis.model)
R> emul.Sicopolis = emulator(Data.Sicopolis.par, Data.Sicopolis.model,
        par.reg=c(TRUE, TRUE, TRUE, TRUE, TRUE),
        time.reg=TRUE, kappa0=1000000, zeta0=50000)
Initializing the emulator...

Initial regression parameters:
```

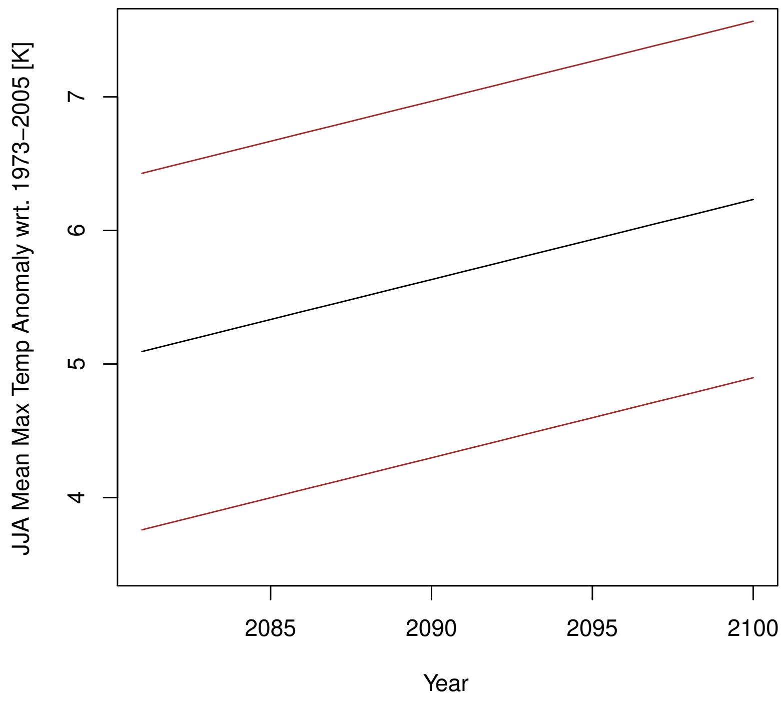


Figure 4: Emulated Korean summer mean maximum temperature change projections with respect to 1973-2005 at $\sigma = 1$ K and $\rho = 0$. Solid and dashed lines: mean response $\pm$ standard errors.

```
5154878.375 171.401 8590.267 -182.659 49920.313 1300.488 -2725.378

Initial emulator likelihood is: -10782007

Optimizing the emulator...
Obtaining emulator parameter ranges for optimization...

Relative tolerance to be used in optimization: 1e-10

Option 'fix.betas': during optimization beta parameters are going to be
fixed at the following values:
5154878.375 171.401 8590.267 -182.659 49920.313 1300.488 -2725.378

-----------------------------------
Starting parameter optimization...
------------------------------------
 0:     10782007.: 0.900 1.00e+06  50000.0  1.97305  9.84502  19.9205  1.98380  7.45
 1:     672836.29:  1.00 3.29e+07 7.05408e+07  2.83691  8.68834  27.1292  2.24728 0.00149
 2:     672613.27: 0.999 3.29e+07 7.05341e+07  2.83691  8.68834  27.1292  2.24728 0.0537
 3:     667210.63: 0.997 3.20e+07 7.08502e+07  2.86488  8.69561  27.3808  2.26410  14.54

79:     485611.17: 0.999989 5.83e+06  41529.0  13.4764  20.1003  199.578  5.72313  10.9
Optimization SUCCESSFUL! Optimization message below:

relative convergence (4)

Final parameters
rho kappa zeta phi phi phi phi phi
0.999989 5829746.770135 41528.993339 13.476403 20.100261 199.578404 5.723128 10.901509
```

```
79 iterations were performed
Final likelihood = -485611.2

CAUTION! The optimization might only find a local minimum.
```

The emulator takes roughly 3 minutes to fit on a 3 Ghz Intel core i5 16GB 2400 MHz DDR4 Macintosh computer. The final emulator is very smooth as evidenced by the relatively high range parameters compared to the ensemble parameter range, and has an extremely low nugget compared to the partial sill parameter.

Next, we perform cross-validation of the emulator for the entire time series. We withhold three ensemble members, and predict at withheld parameters using the model output at the remaining 97 parameter settings.

```
R> test.csv(emul.Sicopolis, num.test=3, plot.std=TRUE, theseed=13241240)
Predicting for run number:  3
Predicting for run number:  26
Predicting for run number:  93
```

Here, we have specified a random seed, and the output tells us which model runs were excluded. Alternatively, we can select the runs to withhold via the `test.runind` argument. We present the results in Figure 5. The emulator has remarkable skill at recovering the output of the withheld models.

Now, we withhold more runs and perform a more systematic analysis of emulator behavior:

```
R> mytest = test.csv(emul.Sicopolis, num.test=10, plot.std=FALSE, theseed=13241240,
                  make.plot=FALSE)
Predicting for run number:  3
Predicting for run number:  7
Predicting for run number:  26
Predicting for run number:  34
Predicting for run number:  37
Predicting for run number:  43
  ...Prediction error. Likely because prediction parameters are out of bounds
Predicting for run number:  91
Predicting for run number:  93
Predicting for run number:  99
Predicting for run number:  100
NOTE: 1  prediction points were omitted

R> cat("95% CI coverage:", mytest$coverage, "\n")
95% CI coverage: 0.9768028
```

Note that **stilt** does not extrapolate beyond the parameter range of the ensemble. Since one of the parameters is at its maximum among the ensemble for the 43^{rd} run, this run is skipped during the cross-validation. We disable the plotting since our main interest here is empirical coverage of the 95% prediction interval. The coverage (0.977) is relatively close to the ideal theoretical value of 0.95, which suggests that the emulator is relatively well calibrated.

We finish by plotting the response surface of the emulator for parameters 4 (snow PDD factor), and 5 (ice PDD factor). We fix flow enhancement factor, basal sliding factor, and geothermal heat flux at values of 3.0, 10.0 $\mathrm{m\,y^{-1}\,Pa^{-1}}$, and 45.0 $\mathrm{m\,W\,m^{-2}}$, respectively.

```
R> rsurface.plot(emul.Sicopolis, parind=c(4,5), parvals=c(3, 10, 45, NA, NA),
              tind=600, n1=10, n2=10)
```

We look at the 600^{th} time index (year 2439), where `n1` and `n2` are the number of grid points to use in the x and y directions, respectively. Figure 6 shows the response surface. A monotonic positive relationship exists between the ice mass anomaly as a function of the snow PDD factor across most of the ice PDD factor range. However, the relationship between the ice mass loss and the ice PDD factor appears to be non-monotonic.

Concluding remarks

Here, we present **stilt** - a package for simplified Gaussian process emulation. The package is designed for emulation of time series model output in parameter space, although it can be applied for kriging more generally. The focus is on simplicity, so the package could be easily applied to many challenging

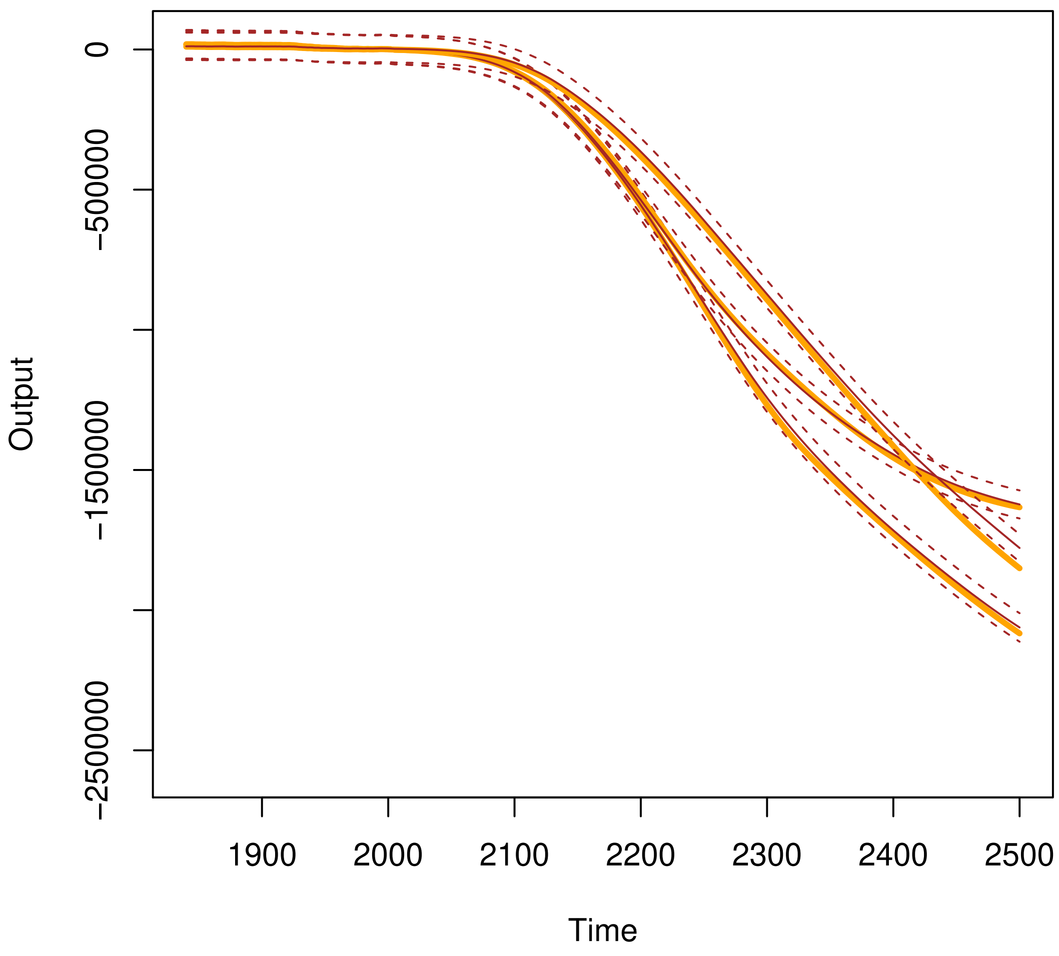


Figure 5: Cross-validation of Greenland Ice Sheet anomaly with respect to year 2003 [Gt] from the SICOPOLIS ensemble. Thick beige lines: actual model output. Brown lines: emulator predictions with standard error confidence intervals.

problems by users outside of the statistics research community. The streamlined emulator fitting and prediction means the package is also useful pedagogically in demonstrating the Gaussian process emulation. We assume separability in space and time for the Gaussian process. This allows us to reduce computational burden using matrix algebra, and makes it possible to apply the package to moderately large datasets, especially in the time dimension. We showcase package capabilities on three examples which differ in the amount of parameters in the model ensemble. Specifically, we use a 1D toy dataset, a 2D dataset of Korean summer mean maximum temperatures, and the 5D SICOPOLIS ice sheet model output. Using cross-validation functions, we show the capability of the emulator to predict model response at excluded parameter values in up to five dimensions. We demonstrate the capacity to visualize estimated 2D model response surfaces. The package can be useful when computational resources are limited, and a relatively fast statistical approximator is required for a complex model across a range of parameter space. Some limitations of the package are homoskedasticity, separability, and the fixed covariance structure (e.g., AR(1) and the squared exponential function). We choose the AR(1) model because it is applicable to data in diverse scientific fields (Hasselmann, 1976; Keller and McInerney, 2008; Li, 2011; Olson et al., 2013). The disadvantage is that such an emulator may not handle seasonal or periodic effects that may be present in computer model output. Extending the package to account for heteroskedasticity (e.g., **hetGP** package, Binois and Gramacy, 2017) should be considered in future work.

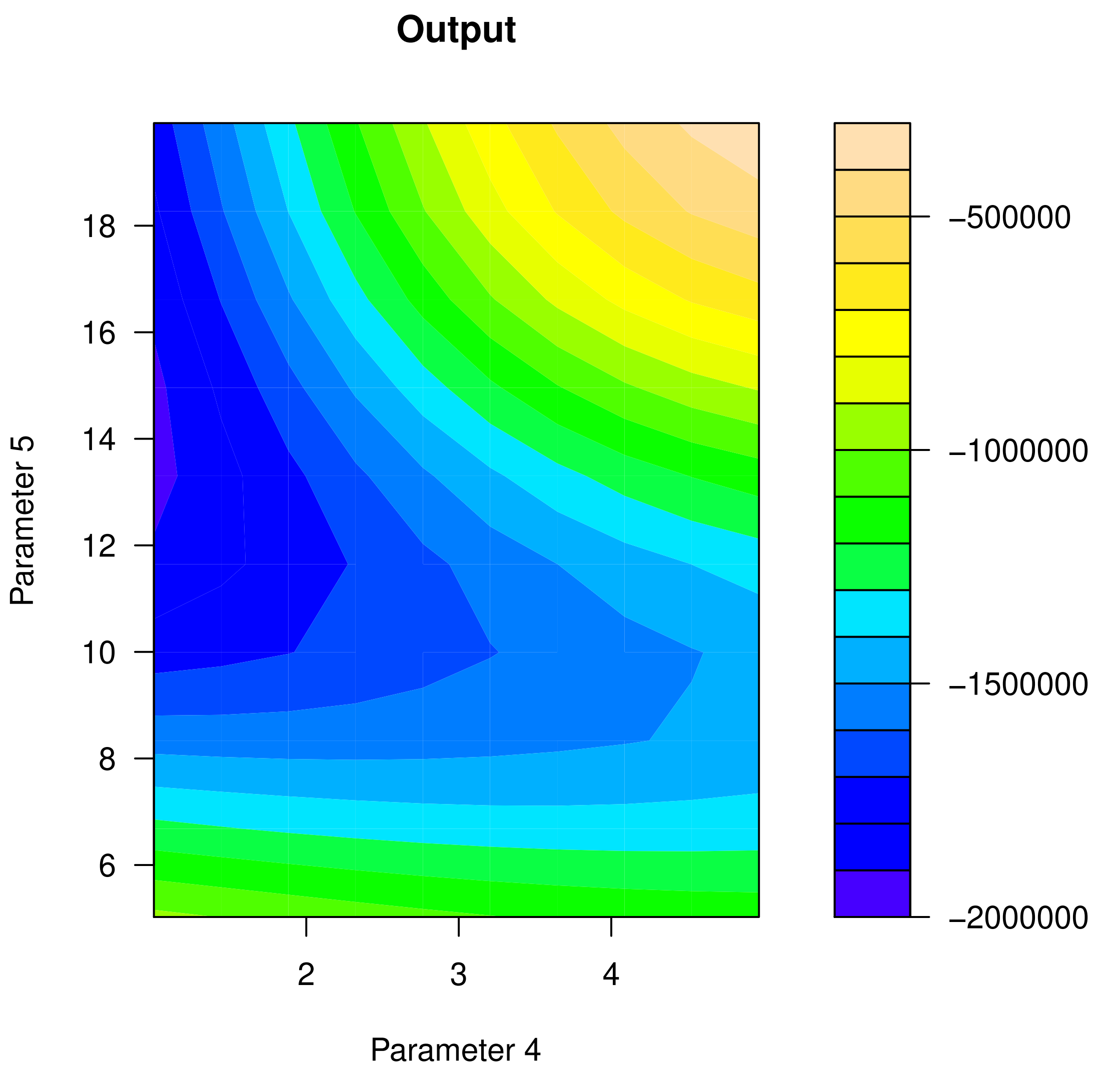


Figure 6: Response surface of SICOPOLIS Greenland Ice Sheet anomaly with respect to year 2003 [Gt] for year 2439 as a function of parameter 4 (snow PDD factor, mm day^{-1} K^{-1}) and parameter 5 (ice PDD factor, mm day^{-1} K^{-1})

Acknowledgments

For their roles in producing, coordinating, and making available the CMIP5 model output, we acknowledge the climate modeling groups, the World Climate Research Programme's (WCRP) Working Group on Coupled modeling (WGCM), and the Global Organization for Earth System Science Portals (GO-ESSP). We thank Jong-Soo Shin for help with extracting Korean temperature output, and Patrick Applegate for sharing the SICOPOLIS ice sheet model output. We acknowledge financial support from National Research Foundation of Korea (NRF-2009-0093069, NRF-2018R1A5A1024958), and from the Institute for Basic Science (project code IBS-R028-D1). This work was also co-supported by the National Science Foundation through the Network for Sustainable Climate Risk Management (SCRiM) under NSF cooperative agreement GEO-1240507 and the Penn State Center for Climate Risk Management. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, or any other foundation or entity.

Appendix

This Appendix describes the technique to reduce computational cost when evaluating the likelihood (Equation 8). In its original form, it involves a computationally expensive inverse of an $np \times np$ matrix. Consider a $p \times n$ matrix $\mathbf{C}$, where $\mathbf{Y} - \boldsymbol{\mu}_\beta = \text{vec}(\mathbf{C})$, and the vec operation stacks columns of a matrix into a column vector, from left to right (Rougier, 2008). Using properties of Kronecker products (Golub and Van Loan, 1996), of the vec operator (Magnus and Neudecker, 2007), and other matrix algebra:

$$\begin{aligned}(\mathbf{Y} - \boldsymbol{\mu}_\beta)^T \Sigma^{-1} (\mathbf{Y} - \boldsymbol{\mu}_\beta) &= (\text{vec}(\mathbf{C}))^T [\Sigma_t^{-1} \otimes \Sigma_\theta^{-1}] \text{vec}(\mathbf{C}) \\ &= (\text{vec}(\mathbf{C}))^T [(\Sigma_t^{-1})^T \otimes \Sigma_\theta^{-1}] \text{vec}(\mathbf{C}) \\ &= (\text{vec}(\mathbf{C}))^T \text{vec}(\Sigma_\theta^{-1} \mathbf{C} \Sigma_t^{-1}) \\ &= \text{sum}\left[\mathbf{C} * (\Sigma_\theta^{-1} \mathbf{C} \Sigma_t^{-1})\right],\end{aligned} \tag{15}$$

where $*$ is the Hadamard product. This reduces the dimension of matrices that are inverted to $p \times p$ and $n \times n$, thus substantially reducing computational burden. We further note that both matrices Σ_θ and Σ_t are positive definite (Wicklin, 2013, and others). Thus, the inverses can be found through the Cholesky decomposition of $\Sigma_\theta = \mathbf{R}_\theta^T \mathbf{R}_\theta$, where $\mathbf{R}_\theta$ (called Cholesky factor) is an upper triangular matrix; and similarly for Σ_t.

Additionally, determinant computations can be simplified considerably (Gentle, 2007):

$$|\Sigma| = |\Sigma_t \otimes \Sigma_\theta| = |\Sigma_t|^p |\Sigma_\theta|^n. \tag{16}$$

Thus, we only need to evaluate the determinants of the individual covariance matrices. Moreover, since Σ_θ and Σ_t are positive definite, $|\Sigma_\theta| = \prod_{i=1}^p r_{\theta,ii}^2$, where $r_{\theta,ii}$ are diagonal elements of the Cholesky factor $\mathbf{R}_\theta$. Similar calculations can be performed for $|\Sigma_t|$.

Bibliography

P. J. Applegate, N. Kirchner, E. J. Stone, K. Keller, and R. Greve. An assessment of key model parametric uncertainties in projections of Greenland ice sheet behavior. *Cryosphere*, 6(3):589–606, 2012. URL `https://doi.org/10.5194/tc-6-589-2012`. [p217]

K. S. Bakar and S. K. Sahu. spTimer: Spatio-temporal Bayesian modeling using R. *Journal of Statistical Software*, 63(15):1–32, 2015a. URL `https://doi.org/10.18637/jss.v063.i15`. [p209]

K. S. Bakar and S. K. Sahu. *spTimer: Spatio-Temporal Bayesian Modeling Using R*, 2015b. R package version 2.0-1. [p209]

K. S. Bhat, M. Haran, R. Olson, and K. Keller. Inferring likelihoods and climate system characteristics from climate models and multiple tracers. *Environmetrics*, 23(4):345–362, 2012. URL `https://doi.org/10.1002/env.2149`. [p209, 210]

M. Binois and R. B. Gramacy. *hetGP: Heteroskedastic Gaussian Process Modeling and Design under Replication*, 2017. URL `https://CRAN.R-project.org/package=hetGP`. R package version 1.0.1. [p220]

W. Chang, P. J. Applegate, M. Haran, and K. Keller. Probabilistic calibration of a Greenland ice sheet model using spatially resolved synthetic observations: Toward projections of ice mass loss with uncertainties. *Geoscientific Model Development*, 7(5):1933–1943, 2014a. ISSN 1991-959X. URL `https://doi.org/10.5194/gmd-7-1933-2014`. [p209]

W. Chang, M. Haran, R. Olson, and K. Keller. Fast dimension-reduced climate model calibration and the effect of data aggregation. *Annals of Applied Statistics*, 8(2):649–673, 2014b. ISSN 1932-6157. URL `https://doi.org/10.1214/14-AOAS733`. [p209, 211]

N. Cressie. *Statistics for Spatial Data*. Wiley Series in Probability and Mathematical Statistics, New York, U.S.A, 1993. URL `https://doi.org/10.1002/9781119115151`. [p210]

N. Cressie and G. Johannesson. Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 70(1):209–226, 2008. URL `https://doi.org/10.1111/j.1467-9868.2007.00633.x`. [p210]

G. M. Dancik. *mlegp: Maximum Likelihood Estimates of Gaussian Processes*, 2013. URL `https://CRAN.R-project.org/package=mlegp`. R package version 3.1.4. [p209, 211]

G. M. Dancik and K. S. Dorman. mlegp: Statistical analysis for computer models of biological systems using R. *Bioinformatics*, 24(17):1966–1967, 2008. URL https://doi.org/10.1093/bioinformatics/btn329. [p209, 211]

D. Drignei, C. E. Forest, and D. Nychka. Parameter estimation for computationally intensive nonlinear regression with an application to climate modeling. *Annals of Applied Statistics*, 2(4):1217–1230, 2008. URL https://doi.org/10.1214/08-AOAS210. [p209]

A. O. Finley, S. Banerjee, and A. E.Gelfand. spBayes for large univariate and multivariate point-referenced spatio-temporal data models. *Journal of Statistical Software*, 63(13):1–28, 2015. URL https://doi.org/10.18637/jss.v063.i13. [p209]

D. M. Gay. Usage summary for select optimization routines. Computing Science Technical Report 153. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey, U. S. A., 1990. [p212]

J. E. Gentle. *Matrix Algebra. Theory, Computations, and Applications in Statitics*. Springer, New York, U.S.A, 2007. URL https://doi.org/10.1007/978-0-387-70873-7. [p222]

G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 3 edition, 1996. [p212, 222]

R. Greve. Application of a polythermal three-dimensional ice sheet model to the Greenland ice sheet: Response to steady-state and transient climate scenarios. *Journal of Climate*, 10(5):901–918, MAY 1997. [p217]

R. Greve, F. Saito, and A. Abe-Ouchi. Initial results of the SeaRISE numerical experiments with the models SICOPOLIS and IcIES for the Greenland ice sheet. *Annals of Glaciology*, 52(58):23–30, 2011. URL https://doi.org/10.3189/172756411797252068. [p217]

B. Gräler, E. Pebesma, and G. Heuvelink. Spatio-Temporal Interpolation using gstat. *The R Journal*, 8 (1):204–218, 2016. [p209]

J. Hansen, R. Ruedy, M. Sato, and K. Lo. Global surface temperature change. *Reviews of Geophysics*, 48 (4), 2010. URL https://doi.org/10.1029/2010RG000345. [p210]

K. Hasselmann. Stochastic climate models part I. Theory. *Tellus*, 28(6):473–485, 1976. ISSN 2153-3490. URL https://doi.org/10.3402/tellusa.v28i6.11316. [p211, 220]

D. Higdon, J. Gattiker, B. Williams, and M. Rightley. Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, 103(482):570–583, 2008. URL https://doi.org/10.1198/016214507000000888. [p209, 211]

S. Hirahara, M. Ishii, and Y. Fukuda. Centennial-scale sea surface temperature analysis and its uncertainty. *Journal of Climate*, 27(1):57–75, 2014. URL https://doi.org/10.1175/JCLI-D-12-00837.1. [p210]

P. B. Holden, N. R. Edwards, K. I. C. Oliver, T. M. Lenton, and R. D. Wilkinson. A probabilistic calibration of climate sensitivity and terrestrial carbon change in GENIE-1. *Climate Dynamics*, 35(5): 785–806, 2010. URL https://doi.org/10.1007/s00382-009-0630-8. [p209]

D. A. Jones, W. Wang, and R. Fawcett. High-quality spatial climate data-sets for Australia. *Australian Meteorological and Oceanographic Journal*, 58(4):233–248, 2009. URL https://doi.org/10.22499/2.5804.003. [p210]

K. Keller and D. McInerney. The dynamics of learning about a climate threshold. *Climate Dynamics*, 30 (2-3):321–332, 2008. URL https://doi.org/10.1007/s00382-007-0290-5. [p211, 220]

M. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society Series B - Statistical Methodology*, 63(3):425–450, 2001. ISSN 1369-7412. URL https://doi.org/10.1111/1467-9868.00294. [p209]

Z. Li. Evaluating GDP forecasting models for Korea. Technical report, International Monetary Fund, 2011. [p220]

J. Magnus and H. Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. John Wiley & Sons, 3rd edition, 2007. [p222]

R. H. Moss, J. A. Edmonds, K. A. Hibbard, M. R. Manning, S. K. Rose, D. P. Van Vuuren, T. R. Carter, S. Emori, M. Kainuma, T. Kram, G. A. Meehl, J. F. B. Mitchell, N. Nakicenovic, K. Riahi, S. J. Smith, R. J. Stouffer, A. M. Thomson, J. P. Weyant, and T. J. Wilbanks. The next generation of scenarios for climate change research and assessment. *Nature*, 463(7282):747–756, 2010. URL https://doi.org/10.1038/nature08823. [p215]

R. Olson, R. Sriver, M. Goes, N. M. Urban, H. D. Matthews, M. Haran, and K. Keller. A climate sensitivity estimate using Bayesian fusion of instrumental observations and an Earth system model. *Journal of Geophysical Research Atmospheres*, 117(4), 2012. URL https://doi.org/10.1029/2011JD016620. [p209]

R. Olson, R. Sriver, W. Chang, M. Haran, N. M. Urban, and K. Keller. What is the effect of unresolved internal climate variability on climate sensitivity estimates? *Journal of Geophysical Research Atmospheres*, 118(10):4348–4358, 2013. URL https://doi.org/10.1002/jgrd.50390. [p209, 211, 220]

R. Olson, W. Chang, K. Keller, and M. Haran. *stilt: Separable Gaussian Process Interpolation (Emulation)*, 2017. URL https://CRAN.R-project.org/package=stilt. R package version 1.2.0. [p210]

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006. URL http://www.gaussianprocess.org/gpml/chapters/RW.pdf. [p209, 210, 212]

J. Rougier. Efficient emulators for multivariate deterministic functions. *Journal of Computational and Graphical Statistics*, 17(4):827–843, 2008. URL https://doi.org/10.1198/106186008X384032. [p209, 210, 211, 212, 222]

M. Schlather, A. Malinowski, P. J. Menck, M. Oestin, and K. Strokorb. Analysis, simulation and prediction of multivariate random fields with package randomfields. *Journal of Statistical Software*, 63(8):1–25, 2015. URL https://doi.org/10.18637/jss.v063.i08. [p209]

D. M. H. Sexton, J. M. Murphy, M. Collins, and M. J. Webb. Multivariate probabilistic projections using imperfect climate models part I: Outline of methodology. *Climate Dynamics*, 38(11-12):2513–2542, 2012. URL https://doi.org/10.1007/s00382-011-1208-9. [p209]

J. Shin, R. Olson, and S.-I. An. Projected heat wave characteristics over the Korean Peninsula during the twenty-first century. *Asia-Pacific Journal of Atmospheric Sciences*, 54(1):53–61, 2018. URL https://doi.org/10.1007/s13143-017-0059-7. [p215]

B. J. Smith, J. Yan, and M. K. Cowles. Unified geostatistical modeling for data fusion and spatial heteroskedasticity with R package ramps. *Journal of Statistical Software*, 25(10):1–21, 2008. URL https://doi.org/10.18637/jss.v025.i10. [p209]

M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Science+Business Media, New York, U.S.A, 1999. URL https://doi.org/10.1007/978-1-4612-1494-6. [p209, 210]

K. E. Taylor, R. J. Stouffer, and G. A. Meehl. An overview of CMIP5 and the experiment design. *Bulletin of the American Meteorological Society*, 93(4):485–498, 2012. URL https://doi.org/10.1175/BAMS-D-11-00094.1. [p215]

R. Wicklin. *Simulating Data with SAS*. The SAS Institute, 2013. [p222]

Roman Olson
Department of Atmospheric Sciences
Yonsei University
529A Sciences Building, 50 Yonsei-ro
Sinchon-dong, Seodaemun-gu, Seoul, 03722, South Korea
And:
Center for Climate Physics
Institute for Basic Science
Room 1111, Tonghapgigyegwan Building
Busandaehak-ro 63 beon-gil 2
Geumjeong-gu, Busan, 46241
South Korea
And:
Pusan National University, Busan, South Korea, 46241
romanolson@pusan.ac.kr

Kelsey L. Ruckert
Penn State University
Earth and Environmental Systems Institute
320c EES Building
University Park, PA, 16802, United States of America
klr324@psu.edu

Won Chang
Division of Statistics and Data Science
Department of Mathematical Sciences
University of Cincinatti
5516 French Hall, 2815 Commons Way
Cincinatti, OH, 45221-0025, United States of America
won.chang@uc.edu

Klaus Keller
Department of Geosciences
Penn State University
436 Deike Building
University Park, PA, 16802, United States of America
kzk10@psu.edu

Murali Haran
Department of Statistics
Penn State University
326 Thomas Building
University Park, PA, 16802, United States of America
mharan@stat.psu.edu

Soon-Il An
Department of Atmospheric Sciences
Yonsei University
538 Sciences Building, 50 Yonsei-ro
Sinchon-dong, Seodaemun-gu, Seoul, 03722, South Korea
sian@yonsei.ac.kr

SMM: An R Package for Estimation and Simulation of Discrete-time semi-Markov Models

by Vlad Stefan Barbu, Caroline Bérard, Dominique Cellier, Mathilde Sautreuil and Nicolas Vergne

Abstract Semi-Markov models, independently introduced by Lévy (1954), Smith (1955) and Takacs (1954), are a generalization of the well-known Markov models. For semi-Markov models, sojourn times can be arbitrarily distributed, while sojourn times of Markov models are constrained to be exponentially distributed (in continuous time) or geometrically distributed (in discrete time). The aim of this paper is to present the R package **SMM**, devoted to the simulation and estimation of discrete-time multi-state semi-Markov and Markov models. For the semi-Markov case we have considered: parametric and non-parametric estimation; with and without censoring at the beginning and/or at the end of sample paths; one or several independent sample paths. Several discrete-time distributions are considered for the parametric estimation of sojourn time distributions of semi-Markov chains: Uniform, Geometric, Poisson, Discrete Weibull and Binomial Negative.

Introduction

Semi-Markov models, independently introduced by Lévy (1954), Smith (1955) and Takacs (1954), are a generalization of the well-known Markov models. For semi-Markov models, sojourn times can be arbitrarily distributed, while sojourn times of Markov models are constrained to be exponentially distributed (in continuous time) or geometrically distributed (in discrete time). For this reason, semi-Markov processes are more general and more adapted for applications than the Markov processes.

Semi-Markov processes have become important tools in probability and statistical modeling with applications in various domains like survival analysis, biology, reliability, DNA analysis, insurance and finance, earthquake modeling, meteorology studies, etc.; see, e.g., Heutte and Huber-Carol (2002), Ouhbi and Limnios (2003), Chryssaphinou et al. (2008), Janssen and Manca (2006), Votsi et al. (2012), D'Amico et al. (2013), Votsi et al. (2014), D'Amico et al. (2016b), Barbu et al. (2016), D'Amico et al. (2016a) for semi-Markov processes in continuous or discrete time with various applications and Sansom and Thomson (2001), Bulla and Bulla (2006), Barbu and Limnios (2008), for hidden semi-Markov models and applications in climatology, finance and DNA analysis.

Note that the semi-Markov theory is developed mainly in a continuous-time setting, while utterly less works address the discrete-time case. We refer the reader to Limnios and Oprisan (2001) for continuous-time framework and to Barbu and Limnios (2008) and references therein for discrete-time framework. The R package **SMM** that we present in this paper is developed in discrete time. Note that undertaking works also in discrete time (modeling stochastic tools, associated estimation procedures, corresponding software, etc.) is an important issue for several reasons. In our opinion, the most relevant of these reasons is that the time scale is intrinsically discrete in several applications. For instance, in DNA studies, when modeling a nucleotide or protein sequence by means of a stochastic process, the "time" of that process is in fact the position along the sequence, so it is discrete. Other examples can be found in some reliability/survival analysis applications where the time represents the number of cycles of a system or the counting of days/hours/etc. We can argue further for the importance of developing works also in discrete time, in parallel to their analogous ones developed in continuous time. For instance, we can mention the simplicity of computations in discrete time, the fact that a discrete-time stochastic process does not explode, the potential use of discrete processes after the discretization of continuous ones, etc.

Few R packages have been developed to handle semi-Markov models or hidden semi-Markov models. For semi-Markov models we have the recent **semiMarkov** R package (Król and Saint-Pierre, 2015) that performs maximum likelihood estimation for parametric continuous-time semi-Markov processes, where the distribution can be chosen between Exponential, Weibull or exponentiated Weibull. That package computes associated hazard rates; covariates can also be taken into account through the Cox proportional hazard model. Two R packages are also dedicated to hidden semi-Markov models, implementing estimation and prediction methods: the **hsmm** R package (Bulla et al., 2010) and the **mhsmm** R package (O'Connell and Højsgaard, 2011).

Note that there is no R package developed for discrete-time multi-state semi-Markov models. Thus the purpose of this paper is to present an R package that we have developed, called **SMM**, which performs parametric and non-parametric estimation and simulation for multi-state discrete-time semi-Markov processes. For the parametric estimation, several discrete distributions are considered for the sojourn times: Uniform, Geometric, Poisson, Discrete Weibull and Negative Binomial. The non-parametric estimation concerns the sojourn time distributions, where no assumptions are made on the shape of distributions. Moreover, the estimation can be done on the basis of one or several trajectories, with or without censoring. The aim of this paper is to describe the different possibilities of this package. To summarize, the package **SMM** that we present deals with different problems:

- Parametric estimation for sojourn time distributions (Uniform, Geometric, Poisson, Discrete Weibull and Negative Binomial) or non-parametric estimation;
- One or several sample paths;
- Four different types of sojourn times: a general one depending on the current state and on the next state to be visited, one depending only on the next state, one depending only on the current state, and one depending neither on the current state nor on the next state;
- Four different types of censoring: censoring at the beginning of sample paths, censoring at the end of sample paths, censoring at the beginning and at the end of sample paths or no censoring at all.

Let us make some remarks about these points.

First, concerning the censoring, the simplest situation is the one when all the sojourn times are completely observed (non censored). A more complicated and realistic framework is when the last sojourn time is not completely observed, thus being right censored; in most practical situations this case occurs. An analogous situation is when the first sojourn time is not completely observed, thus being also right censored. In practice, this last case occurs when one does not know the beginning of a phenomenon modeled by a semi-Markov chain. Note that this censoring at the beginning of the sample path is a right censoring (not a left censoring); indeed, when the first sojourn time is censored as in our paper, the available information is that the real sojourn time (that is not observed) is greater than this censored observed time. Thus we are clearly in a right censoring framework, although this happens at the beginning (i.e., left) of the trajectory, which could seem confusing. The most complete framework is when both the first and the last sojourn times are right censored.

Second, when considering estimation starting from several independent sample paths of a semi-Markov chain, it is assumed that all the trajectories are censored in the same way; note that this is not a real constraint, but we imposed this condition only in order to avoid useless technical notations that would make the comprehension more difficult.

Third, note that it is important for the four types of models (of sojourn times) to be considered separately because: (i) in practical situations, one model could be more adapted than some other; (ii) different models will yield specific parameter estimators, so it is important to study them separately.

The paper is organized as follows. The next section **Semi-Markov models** describes the semi-Markov models used in this package. Section **The SMM package** illustrates the different functions of the **SMM** package. We end the paper by presenting some concluding remarks on this R package.

Semi-Markov models

Let us consider a random system with finite state space $E = \{1, \dots, s\}$, $s < \infty$. Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space and assume that the time evolution of the system is governed by a stochastic process $Y = (Y_k)_{k \in \mathbb{N}^*}$, defined on $(\Omega, \mathcal{A}, \mathbb{P})$ with values in E; that is to say that Y_k gives the state of the system at time k. Let $T = (T_m)_{m \in \mathbb{N}^*}$, defined on $(\Omega, \mathcal{A}, \mathbb{P})$ with values in $\mathbb{N}$, be the successive time points when state changes in $(Y_k)_{k \in \mathbb{N}^*}$ occur (the jump times) and let also $J = (J_m)_{m \in \mathbb{N}^*}$, defined on $(\Omega, \mathcal{A}, \mathbb{P})$ with values in E, be the successively visited states at these time points. The relation between the process Y and the process J of the successively visited states is given by $Y_k = J_{N(k)}$, or, equivalently, $J_m = Y_{T_m}, m, k \in \mathbb{N}$, where $N(k) := \max\{m \in \mathbb{N} \mid T_m \leq k\}$ is the discrete-time counting process of the number of jumps in $[1, k] \subset \mathbb{N}$.

In this paper we consider four different semi-Markov models corresponding to the following four types of sojourn times.

- Sojourn times depending on the current state and on the next state:

$$f_{ij}(k) = \mathbb{P}(T_{m+1} - T_m = k | J_m = i, J_{m+1} = j);$$

- Sojourn times depending only on the current state:

$$f_{i\bullet}(k) = \mathbb{P}(T_{m+1} - T_m = k | J_m = i);$$

- Sojourn times depending only on the next state to be visited:

$$f_{\bullet j}(k) = \mathbb{P}(T_{m+1} - T_m = k | J_{m+1} = j);$$

- Sojourn times depending neither on the current state nor on the next state:

$$f(k) = \mathbb{P}(T_{m+1} - T_m = k).$$

Note that the sojourn times of the type $f_{i\bullet}(\cdot)$, $f_{\bullet j}(\cdot)$ or $f(\cdot)$ are particular cases of the general type $f_{ij}(\cdot)$. Nonetheless, in some specific applications, particular cases can be important because they are adapted to the phenomenon under study; that is the reason why we investigate these cases separately.

General case: sojourn times of the type $f_{ij}(.)$

Definition 1 (semi-Markov chain SMC and Markov renewal chain MRC). *If we have*

$$\mathbb{P}(J_{m+1} = j, T_{m+1} - T_m = k | J_m = i, J_{m-1}, \ldots, J_1, T_m, \ldots, T_1) = \mathbb{P}(J_{m+1} = j, T_{m+1} - T_m = k | J_m = i), \tag{1}$$

then $Y = (Y_k)_k$ is called a semi-Markov chain *(SMC) and $(J, T) = (J_m, T_m)_m$ is called a* Markov renewal chain *(MRC).*

All along this paper we assume that the MRC or SMC are homogeneous with respect to the time in the sense that Equation (1) is independent of m.

Note that if (J, T) is a MRC, then it can be proved that $J = (J_m)_{m \in \mathbb{N}^*}$ is a Markov chain with state space E, called the *embedded Markov chain* of the MRC (J, T) (or of the SMC Y).

Definition 2. *For a semi-Markov chain we define:*

- *the* semi-Markov kernel $(q_{ij}(k))_{i,j \in E, k \in \mathbb{N}}$,

$$q_{ij}(k) = \mathbb{P}(J_{m+1} = j, T_{m+1} - T_m = k | J_m = i);$$

- *the* initial distribution $(\mu_i)_{i \in E}$,

$$\mu_i = \mathbb{P}(J_1 = i) = \mathbb{P}(Y_1 = i);$$

- *the* transition matrix $(p_{ij})_{i,j \in E}$ *of the embedded Markov chain* $J = (J_m)_m$,

$$p_{ij} = \mathbb{P}(J_{m+1} = j | J_m = i);$$

- *the* conditional sojourn time distributions $(f_{ij}(k))_{i,j \in E, k \in \mathbb{N}}$,

$$f_{ij}(k) = \mathbb{P}(T_{m+1} - T_m = k | J_m = i, J_{m+1} = j).$$

Note that

$$q_{ij}(k) = p_{ij} f_{ij}(k). \tag{2}$$

Clearly, a semi-Markov chain is uniquely determined a.s. by an initial distribution $(\mu_i)_{i \in E}$ and a semi-Markov kernel $(q_{ij}(k))_{i,j \in E, k \in \mathbb{N}}$ or, equivalently, by an initial distribution $(\mu_i)_{i \in E}$, a Markov transition matrix $(p_{ij})_{i,j \in E}$ and conditional sojourn time distributions $(f_{ij}(k))_{i,j \in E, k \in \mathbb{N}}$.

Other assumptions we make are: (i) We do not allow transitions to the same state, i.e., $p_{ii} = 0$ for all $i \in E$, or equivalently $q_{ii}(k) = 0$, for all $i \in E$, $k \in \mathbb{N}$; (ii) We assume that there are not instantaneous transitions, that is $q_{ij}(0) \equiv 0$ or equivalently $f_{ij}(0) \equiv 0$ for all $i, j \in E$; note that this implies that T is a strictly increasing sequence.

For the conditional sojourn time distributions, one can consider the associated cumulative distribution function defined by

$$F_{ij}(k) := \mathbb{P}(T_{m+1} - T_m \leq k | J_m = i, J_{m+1} = j) = \sum_{t=1}^{k} f_{ij}(t).$$

For any distribution function $F(\cdot)$ we can consider the associated survival/reliability function defined by

$$\overline{F}(k) := 1 - F(t).$$

Consequently we have

$$\overline{F}_{ij}(k) := \mathbb{P}(T_{m+1} - T_m > k | J_m = i, J_{m+1} = j) = 1 - \sum_{t=1}^{k} f_{ij}(t) = \sum_{t=k+1}^{\infty} f_{ij}(t).$$

Particular cases: sojourn times of the type $f_{i.}(.)$, $f_{.j}(.)$ and $f(.)$

We have considered up to here general semi-Markov models with the semi-Markov kernel of the type given in (2). Particular types of this semi-Markov model can be taken into account, by considering particular cases of holding time distributions $f_{ij}(k)$, where these distributions depend only on the current state i, or only on the next state j, or neither on i nor on j. For each case, let us define the semi-Markov kernel and the distribution function associated to the sojourn time distribution.

- Sojourn times depending only on the current state:

$$\begin{aligned} q_{ij}(k) &:= p_{ij} f_{i\bullet}(k), \text{ where} \\ f_{i\bullet}(k) &= \mathbb{P}(T_{m+1} - T_m = k | J_m = i) = \sum_{v \in E} p_{iv} f_{iv}(k), \\ F_{i\bullet}(k) &:= \mathbb{P}(T_{m+1} - T_m \leq k | J_m = i) = \sum_{t=1}^{k} f_{i\bullet}(t) = \sum_{t=1}^{k} \sum_{v \in E} p_{iv} f_{iv}(t). \end{aligned} \tag{3}$$

- Sojourn times depending only on the next state:

$$\begin{aligned} q_{ij}(k) &:= p_{ij} f_{\bullet j}(k), \text{ where} \\ f_{\bullet j}(k) &= \mathbb{P}(T_{m+1} - T_m = k | J_{m+1} = j), \\ F_{\bullet j}(k) &:= \mathbb{P}(T_{m+1} - T_m \leq k | J_{m+1} = j) = \sum_{t=1}^{k} f_{\bullet j}(t). \end{aligned} \tag{4}$$

- Sojourn times depending neither on the current state nor on the next state:

$$\begin{aligned} q_{ij}(k) &:= p_{ij} f(k), \text{ where} \\ f(k) &= \mathbb{P}(T_{m+1} - T_m = k). \\ F(k) &:= \mathbb{P}(T_{m+1} - T_m \leq k) = \sum_{t=1}^{k} f(t). \end{aligned} \tag{5}$$

We also denote the associated survival/reliability functions respectively by $\overline{F}_{i\bullet}(k)$, $\overline{F}_{\bullet j}(k)$, $\overline{F}(k)$.

The SMM package

The **SMM** R package is mainly devoted to the simulation and estimation of discrete-time semi-Markov models in different cases by the two following functions:

- `simulSM()` for the simulation of sequences from a semi-Markov model:
  - One or several trajectories
  - According to classical distributions for the sojourn times (Uniform, Geometric, Poisson, Discrete Weibull and Negative Binomial) or according to distributions given by the user
  - Four different types of censoring mechanisms: censoring at the beginning of sample paths, censoring at the end, censoring at the beginning and at the end, no censoring
  - Four different types of sojourn times: depending on the current state and on the next state, depending only on the current state, depending only on the next state, depending neither on the current state nor on the next state
- `estimSM()` for the estimation of model parameters:
  - One or several trajectories
  - Parametric (Uniform, Geometric, Poisson, Discrete Weibull and Negative Binomial) or non-parametric distributions for the sojourn times
  - Four different types of censoring mechanisms: censoring at the beginning of sample paths, censoring at the end, censoring at the beginning and at the end, no censoring

  - Four different types of sojourn times: depending on the current state and on the next state, depending only on the current state, depending only on the next state, depending neither on the current state nor on the next state

As the Negative Binomial distribution and the Discrete Weibull distribution can have several different parameterizations, note that we have considered the following ones in our package:

  - Discrete Weibull of type I with the density $f(x) = q^{(x-1)^\beta} - q^{x^\beta}$ for $x = 1, 2, \ldots, n$ with $n > 0$, q is the first parameter, $0 < q < 1$, and β is the second parameter, $\beta > 0$;
  - Negative Binomial with the density $f(x) = \frac{\Gamma(x+\alpha)}{\Gamma(\alpha)x!}\left(\frac{\alpha}{\alpha+\mu}\right)^\alpha\left(\frac{\mu}{\alpha+\mu}\right)^x$ for $x = 0, 1, \ldots, n$ with $n > 0$, Γ is the Gamma function, α is the parameter of overdispersion, $\alpha > 0$, and μ is the mean, $\mu \in \mathbb{R}$.

In order to avoid any confusion, note also that the expressions of the different densities considered in this package are also provided in the corresponding manual.

The **SMM** R package is also devoted to the simulation and estimation of discrete-time Markov models by the two following functions:

- `simulMk()` for the simulation of sequences from a kth order Markov model;
- `estimMk()` for the estimation of the parameters of the model.

All the different possibilities of the package are illustrated in Figure 1.

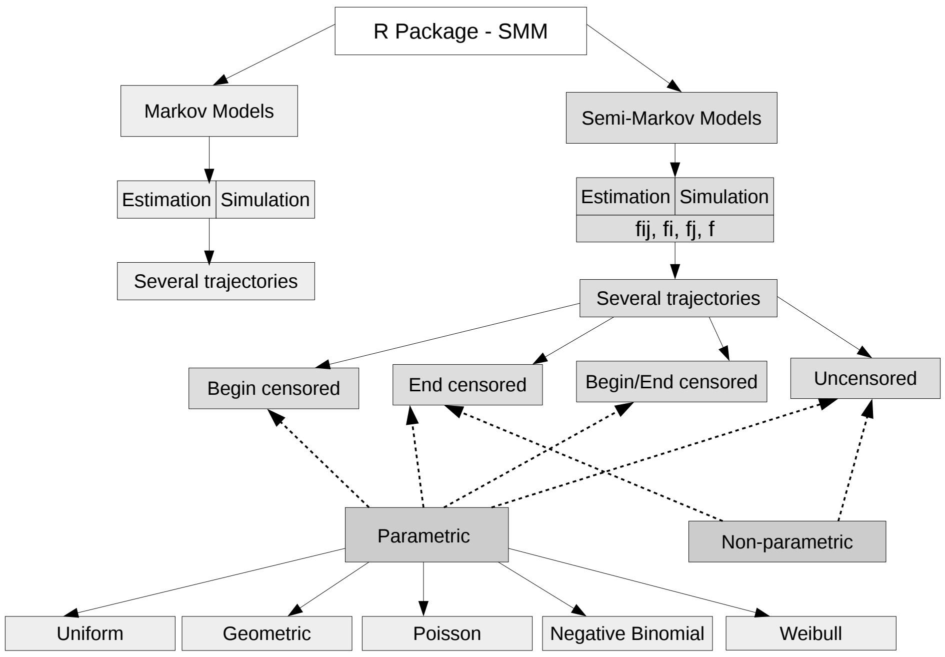


Figure 1: Schema of the **SMM** package.

Simulation of semi-Markov models

Parametric simulation: according to classical distributions

In this part, we will consider the simulation according to classical distributions.

Parameters: This simulation is carried out by the function `simulSM()`. The different parameters of the function are:

- `E`: Vector of state space of length S
- `NbSeq`: Number of simulated sequences

- `lengthSeq`: Vector containing the lengths of each simulated sequence
- `TypeSojournTime`: Type of sojourn time; it can be `"fij"`, `"fi"`, `"fj"` or `"f"` according to the four cases previously discussed
- `init`: Vector of initial distribution of length S
- `Ptrans`: Matrix of transition probabilities of the embedded Markov chain $J = (J_m)_m$ of size $S \times S$
- `distr`: Sojourn time distributions:
  - is a matrix of distributions of size $S \times S$ if `TypeSojournTime` is equal to `"fij"`,
  - is a vector of distributions of size S if `TypeSojournTime` is equal to `"fi"` or `"fj"`,
  - is a distribution if `TypeSojournTime` is equal to `"f"`,

  where the distributions to be used can be one of `"uniform"`, `"geom"`, `"pois"`, `"weibull"` or `"nbinom"`.
- `param`: Parameters of sojourn time distributions:
  - is an array of parameters of size $S \times S \times 2$ if `TypeSojournTime` is equal to `"fij"`
  - is a matrix of parameters of size $S \times 2$ if `TypeSojournTime` is equal to `"fi"` or `"fj"`
  - is a vector of parameters if `TypeSojournTime` is equal to `"f"`
- `cens.beg`: Type of censoring at the beginning of sample paths; 1 (if the first sojourn time is censored) or 0 (if not). All the sequences must be censored in the same way.
- `cens.end`: Type of censoring at the end of sample paths; 1 (if the last sojourn time is censored) or 0 (if not). All the sequences must be censored in the same way.
- `File.out`: Name of fasta file for saving the sequences; if `File.out` = NULL, no file is created. A fasta file is a simple text file containing sequences only described by one description line beginning by a ">": an example is given in Figure 2.

```
> sequence 1
aaacgtacgagtcgatcgatcgactcgatcgtacgtacggt
> sequence 2
acgtattacgatgctagctaggttggggactgcatgcatgaatgagcgatc
```

Figure 2: Example of a fasta file.

The R commands below generate three sequences of size 1000, 10000 and 2000 respectively with the finite state space $E = \{a, c, g, t\}$, where the sojourn times depend on the current state and on the next state.

```
## state space
E = c("a","c","g","t")
S = length(E)
## sequence sizes
lengthSeq3 = c(1000, 10000, 2000)
## creation of the initial distribution
vect.init = c(1/4,1/4,1/4,1/4)
## creation of transition matrix
Pij = matrix(c(0,0.2,0.3,0.4,0.2,0,0.5,0.2,0.5,0.3,0,0.4,0.3,0.5,0.2,0),
             ncol=4)
## creation of the distribution matrix
distr.matrix = matrix(c("dweibull", "pois", "geom", "nbinom",
                         "geom", "nbinom", "pois", "dweibull",
                         "pois", "pois", "dweibull", "geom",
                         "pois","geom", "geom", "nbinom"),
                       nrow = S, ncol = S, byrow = TRUE)
## creation of an array containing the parameters
param1.matrix = matrix(c(0.6,2,0.4,4,0.7,2,5,0.6,
                                         2,3,0.6,0.6,4,0.3,0.4,4),
                                       nrow = S, ncol = S, byrow = TRUE)
param2.matrix = matrix(c(0.8,0,0,2,0,5,0,0.8,
                          0,0,0.8,0,4,0,0,4),
                                           nrow = S, ncol = S, byrow = TRUE)
param.array = array(c(param1.matrix, param2.matrix), c(S,S,2))
## for the reproducibility of the results
set.seed(1)
## simulation of 3 sequences
seq3 = simulSM(E = E, NbSeq = 3, lengthSeq = lengthSeq3,
                           TypeSojournTime = "fij", init = vect.init,
                  Ptrans = Pij, distr = distr.matrix, param = param.array,
                  File.out = "seq3.txt")
```

First, note that in this simulation, the parameters cens.beg and cens.end are equal to 0, that is to say the simulated sequences are not censored.

Second, note also that the parameters of the distributions are given in the following way: for example, $f_{13}(\cdot)$ is Geometric distribution with parameter 0.4, while $f_{14}(\cdot)$ is Negative Binomial with parameters 4 and 2. In other words, the parameters of $f_{13}(\cdot)$ are given in the vector param.array[1,3,] that is equal to (0.4, 0) and the parameters of $f_{14}(\cdot)$ are given in the vector param.array[1,4,] that is equal to (4, 2); that means that if a distribution has only 1 parameter, the corresponding vector of parameters will have 0 on the second position.

Values: The function simulSM() returns a list of simulated sequences. These sequences can be saved in a fasta file by using the parameter File.out.

```
seq3[[1]][1:15]
[1] "g" "g" "g" "g" "c" "c" "c" "a" "a" "a" "c" "c" "c" "g" "g"
```

Non-parametric simulation: according to distributions given by the user

Now we will consider the simulation according to distributions given by the user.

Parameters: This simulation is carried out by the function simulSM(). The different parameters of the function are:

- E: Vector of state space of length S
- NbSeq: Number of simulated sequences
- lengthSeq: Vector containing the lengths of each simulated sequence
- TypeSojournTime: Type of sojourn time; it can be "fij", "fi", "fj" or "f" according to the four cases previously discussed
- init: Vector of initial distribution of length S
- Ptrans: Matrix of transition probabilities of the embedded Markov chain $J = (J_m)_m$ of size $S \times S$
- laws: Sojourn time distributions introduced by the user:
  - is an array of size $S \times S \times Kmax$ if TypeSojournTime is equal to "fij",
  - is a matrix of size $S \times Kmax$ if TypeSojournTime is equal to "fi" or "fj",
  - is a vector of length $Kmax$ if TypeSojournTime is equal to "f",

  where $Kmax$ is the maximum length for the sojourn times.
- cens.beg: Type of censoring at the beginning of sample paths; 1 (if the first sojourn time is censored) or 0 (if not). All the sequences must be censored in the same way.
- cens.end: Type of censoring at the end of sample paths; 1 (if the last sojourn time is censored) or 0 (if not). All the sequences must be censored in the same way.
- File.out: Name of fasta file for saving the sequences; if File.out = NULL, no file is created.

The R commands below generate three sequences of size 1000, 10000 and 2000 respectively with the finite state space $E = \{a, c, g, t\}$, where the sojourn times depend only on the next state.

```
## state space
E = c("a","c","g","t")
S = length(E)
## sequence sizes
lengthSeq3 = c(1000, 10000, 2000)
```

```
## creation of the initial distribution
vect.init = c(1/4,1/4,1/4,1/4)
## creation of transition matrix
Pij = matrix(c(0,0.2,0.3,0.4,0.2,0,0.5,0.2,0.5,0.3,0,0.4,0.3,0.5,0.2,0),
             ncol=4)
## creation of a matrix corresponding to the conditional
## sojourn time distributions
Kmax = 6
nparam.matrix = matrix(c(0.2,0.1,0.3,0.2,0.2,0,0.4,0.2,0.1,
                                            0,0.2,0.1,0.5,0.3,0.15,0.05,0,0,
                                            0.3,0.2,0.1,0.2,0.2,0),
                       nrow = S, ncol = Kmax, byrow = TRUE)
## for the reproducibility of the results
set.seed(2)
## simulation of 3 sequences with censoring at the beginning
seqNP3_begin = simulSM(E = E, NbSeq = 3, lengthSeq = lengthSeq3,
TypeSojournTime = "fj", init = vect.init, Ptrans = Pij,
                                 laws = nparam.matrix, File.out = "seqNP3_begin.txt",
                                 cens.beg = 1, cens.end = 0)
## simulation of 3 sequences with censoring at the end
seqNP3_end = simulSM(E = E, NbSeq = 3, lengthSeq = lengthSeq3,
                                 TypeSojournTime = "fj", init = vect.init, Ptrans = Pij,
                                 laws = nparam.matrix, File.out = "seqNP3_end.txt",
                                 cens.beg = 0, cens.end = 1)
## simulation of 3 sequences censored at the beginning and at the end
seqNP3_begin_end = simulSM(E = E, NbSeq = 3, lengthSeq = lengthSeq3,
                                 TypeSojournTime = "fj", init = vect.init, Ptrans = Pij,
                                 laws = nparam.matrix, File.out = "seqNP3_begin_end.txt",
                                 cens.beg = 1, cens.end = 1)
## simulation of 3 sequences without censoring
seqNP3_no = simulSM(E = E, NbSeq = 3, lengthSeq = lengthSeq3,
                                 TypeSojournTime = "fj", init = vect.init, Ptrans = Pij,
                                 laws = nparam.matrix, File.out = "seqNP3_no.txt")
## for the reproducibility of the results
seqNP3_begin = read.fasta("seqNP3_begin.txt")
seqNP3_end = read.fasta("seqNP3_end.txt")
seqNP3_begin_end = read.fasta("seqNP3_begin_end.txt")
seqNP3_no = read.fasta("seqNP3_no.txt")
seqNP3_begin[[1]][1:15]
```

Note that in this simulation all the different censoring mechanisms are considered.

Values: The function `simulSM()` returns a list of simulated sequences. These sequences can be saved in a fasta file by using the parameter `File.out`.

```
seqNP3_begin[[1]][1:15]
 [1] "c" "g" "g" "g" "g" "g" "g" "c" "c" "c" "a" "a" "a" "a" "a"
```

Estimation of semi-Markov models

In this subsection we explain and illustrate the estimation of a semi-Markov model in the parametric and non-parametric cases.

Parametric estimation of semi-Markov models

We will consider the distributions $f_{ij}(k) = f_{ij}(k;\theta_{ij})$ depending on unknown parameters $\theta_{ij} \in \mathbb{R}^{m_{ij}}$, where the dimension of parameters set m_{ij} is known; no assumptions are made on $\left(p_{ij}\right)_{ij}$. From the data, we want to estimate p_{ij} and θ_{ij}, $i, j \in E$. Let us assume that we have at our disposal several independent sample paths of a semi-Markov chain, say L, each of them of length n_l, $l = 1, \ldots, L$, censored at the beginning and at the end of the trajectory, i.e.,

$$y_1^l, y_2^l, \ldots, y_{n_l}^l,$$

or, equivalently,

$$j_0^l, k_0^l, j_1^l, k_1^l, j_2^l, k_2^l, \ldots, j_{t^l}^l, k_{t^l}^l, j_{t^l+1}^l, k_{t^l+1}^l$$

with $\sum_{i=0}^{t^l+1} k_i^l = n_l$, where $j_0^l, \ldots, j_{t^l+1}^l$ are the successive distinct visited states, k_0^l is the first sojourn time, assumed to be right censored, $k_{t^l+1}^l$ is the last sojourn time, assumed also to be right censored, while $k_1^l, \ldots, k_{t^l}^l$ are the other successive sojourn times, assumed to be complete (observed, non censored).

To estimate the parameters of model, we use the maximum likelihood estimation:

$$\begin{aligned} &\underset{p_{uv},\theta_{uv};u,v\in E}{\operatorname{argmax}} \quad (l(p_{uv}, \theta_{uv}; u, v \in E)) \qquad (6)\\ &= \left(\underset{p_{uv},\theta_{uv};v\in E}{\operatorname{argmax}} \left(\sum_{v\in E} N_{uv}(L, n_{1:L}) \log(p_{uv}) + \sum_{v\in E} \sum_{k=1}^{\max_l(n_l)} N_{uv}(k; L, n_{1:L}) \log(f_{uv}(k; \theta_{uv})) \right.\right.\\ &+ \sum_{v\in E} \sum_{k=1}^{\max_l(n_l)} \overline{N}_{uv}^b(k; L) \log(\overline{F}_{uv}(k; \theta_{uv}))\\ &\left.\left. + \sum_{k=1}^{\max_l(n_l)} \overline{N}_{u\bullet}^e(k; L) \log\left(1 - \sum_{m=1}^{k} \sum_{v\in E} p_{uv} f_{uv}(m; \theta_{uv})\right)\right)\right)_{u\in E}, \end{aligned}$$

where we introduced the following counting processes

$$\begin{aligned} N_{ij}(L, n_{1:L}) &= \sum_{l=1}^{L} \sum_{m=1}^{N^l(n_l)-1} \mathbb{1}_{\{J_m^l=i; J_{m+1}^l=j\}},\\ N_{ij}(k; L, n_{1:L}) &= \sum_{m=1}^{N^l(n_l)-1} \mathbb{1}_{\{J_m^l=i; J_{m+1}^l=j; T_{m+1}^l - T_m^l = k\}},\\ \overline{N}_{ij}^b(k; L) &= \sum_{l=1}^{L} \mathbb{1}_{\{J_0^l=i; J_1^l=j; T_1^l - T_0^l > k\}},\\ \overline{N}_{i\bullet}^e(k; L) &= \sum_{l=1}^{L} \mathbb{1}_{\{J_{T_{N^l(n_l)}^l}^l = i, X_{T_{N^l(n_l)+1}^l}^l > k\}}, \end{aligned}$$

where

$$N^l(n_l) = \max\{m \in \mathbb{N} \mid T_m^l \le n_l\}$$

is the counting process of jump number in $[1; n_l]$ of the trajectory l.

Note that:

- $N_{ij}(L, n_{1:L})$ represents the number of transitions from state i to state j along the L sample paths;

- $N_{ij}(k; L, n_{1:L})$ represents the number of transitions from state i to state j along the L sample paths, with sojourn time of length k in state i;

- $\overline{N}_{ij}^b(k; L)$ represents the number of trajectories starting in state i, with a next transition to state j and censored sojourn time in state i greater than k;

- $\overline{N}_{i\bullet}^e(k; L)$ represents the number of trajectories ending in state i with a censored sojourn time in state i greater than k.

Note also that in the expression (6) of the log-likelihood, the first two terms correspond to the transition probabilities and the observed (non censored) sojourn times, the third term is the contribution to the likelihood of the first sojourn times, assumed to be right censored, while the last term is the contribution to the likelihood of the last sojourn times, assumed to be right censored.

Up to here we presented the estimation for the general case, taking into account the censoring at the beginning and at the end and the sojourn times depending on the current state and on the next state. Thus the maximization problem (6) is written with the sojourn times depending on the current state and on the next state (the general model of the type $q_{ij}(k) = p_{ij} f_{ij}(k)$ given in (2)), but the different cases of sojourn times are written and coded in the package. Note also that different types of censoring are also written and coded in the package.

Parameters: The estimation is carried out by the function `estimSM()`. The different parameters of the function are:

- `file`: Path of the fasta file which contains the sequences from which to estimate
- `seq`: List of the sequence(s) from which to estimate
- `E`: Vector of state space of length S
- `TypeSojournTime`: Type of sojourn time; it can be `"fij"`, `"fi"`, `"fj"` or `"f"` according to the four cases previously discussed
- `distr`: Sojourn time distributions:
  - is a matrix of distributions of size $S \times S$ if `TypeSojournTime` is equal to `"fij"`,
  - is a vector of distributions of size S if `TypeSojournTime` is equal to `"fi"` or `"fj"`,
  - is a distribution if `TypeSojournTime` is equal to `"f"`,

  where the distributions to be used can be one of `"uniform"`, `"geom"`, `"pois"`, `"weibull"` or `"nbinom"`.
- `cens.beg`: Type of censoring at the beginning of sample paths; 1 (if the first sojourn time is censored) or 0 (if not). All the sequences must be censored in the same way.
- `cens.end`: Type of censoring at the end of sample paths; 1 (if the last sojourn time is censored) or 0 (if not). All the sequences must be censored in the same way.

Note that the sequences from which we estimate can be given either as an `R` list (`seq` argument) or as a file in fasta format (`file` argument).

```
## data
seq3 = read.fasta("seq3.txt")
E = c("a","c","g","t")
## creation of the distribution matrix
distr.matrix = matrix(c("dweibull", "pois", "geom", "nbinom",
                        "geom", "nbinom", "pois", "dweibull",
                        "pois", "pois", "dweibull", "geom",
                        "pois","geom", "geom", "nbinom"),
                      nrow = S, ncol = S, byrow = TRUE)
## estimation of simulated sequences
estSeq3 = estimSM(seq = seq3, E = E, TypeSojournTime = "fij",
distr = distr.matrix, cens.end = 0, cens.beg = 0)
```

Here, we estimate simulated sequences with no censoring. The estimation performed will correspond to the likelihood given in (6), without the third and forth terms.

Values: The function `estimSM()` returns a list containing:

- `init`: Vector of size S with estimated initial probabilities of the semi-Markov chain

  ```
  estSeq3$init
  [1] 0.0000000 0.0000000 0.3333333 0.6666667
  ```

- `Ptrans`: Matrix of size $S \times S$ with estimated transition probabilities of the embedded Markov chain $J = (J_m)_m$

  ```
  estSeq3$Ptrans
            [,1]      [,2]      [,3]      [,4]
  [1,] 0.0000000 0.2263948 0.4957082 0.2778970
  [2,] 0.2134472 0.0000000 0.2892209 0.4973319
  [3,] 0.2900627 0.4709042 0.0000000 0.2390331
  [4,] 0.4108761 0.2024169 0.3867069 0.0000000
  ```

- `param`: Array with estimated parameters of the sojourn time distributions

```
estSeq3$param
, , 1

          [,1]      [,2]     [,3]      [,4]
[1,] 0.0000000 1.7867299 0.411398 4.1358797
[2,] 0.6920415 0.0000000 4.745387 0.6131806
[3,] 2.0123457 3.0722433 0.000000 0.5729614
[4,] 3.8088235 0.3068702 0.392237 0.0000000

, , 2

     [,1] [,2] [,3]      [,4]
[1,]    0    0    0 2.1043710
[2,]    0    0    0 0.8425902
[3,]    0    0    0 0.0000000
[4,]    0    0    0 0.0000000
```

Note that, for example, `estSeq3$param[1,3,]` is the vector containing the parameters of the distribution $f_{13}(\cdot)$.

- q: Array of size $S \times S \times Kmax$ with estimated semi-Markov kernel

```
estSeq3$q[,,1:3]
, , 1

           [,1]       [,2]       [,3]       [,4]
[1,] 0.00000000 0.03792273 0.20393336 0.05070777
[2,] 0.14771431 0.00000000 0.00251382 0.19237763
[3,] 0.03877405 0.02181093 0.00000000 0.13695675
[4,] 0.00911087 0.06211573 0.15168076 0.00000000

, , 2

           [,1]       [,2]       [,3]       [,4]
[1,] 0.00000000 0.06775768 0.12003558 0.07072333
[2,] 0.04548987 0.00000000 0.01192905 0.09806611
[3,] 0.07802680 0.06700849 0.00000000 0.05848582
[4,] 0.03470170 0.04305426 0.09218596 0.00000000

, , 3

           [,1]       [,2]       [,3]       [,4]
[1,] 0.00000000 0.06053233 0.07065318 0.06124460
[2,] 0.01400899 0.00000000 0.02830398 0.06214172
[3,] 0.07850845 0.10293320 0.00000000 0.02497570
[4,] 0.06608632 0.02984219 0.05602722 0.00000000
```

Note that, for example, $q_{13}(2) = \mathbb{P}(J_{m+1} = 3, T_{m+1} - T_m = 2 | J_m = 1) = 0.12192187$.

Non-parametric estimation of semi-Markov models
Here we will consider two types of estimation for semi-Markov chains: a direct estimation, obtaining thus empirical estimators (in fact, approached MLEs), cf. Barbu and Limnios (2006), Barbu and Limnios (2008) and an estimation based on a couple Markov chain associated to the semi-Markov chain (see Trevezas and Limnios, 2011).

No censoring: direct estimation
Let $\{Y_1, Y_2, \ldots, Y_n\}$ be a trajectory of a semi-Markov chain $Y = (Y_n)_{n \in \mathbb{N}}$, censored at an arbitrary fixed time n.

- The case $q_{ij}(k) = p_{ij} f_{ij}(k)$: The maximum likelihood estimators are

$$\widehat{p_{ij}}(n) = \frac{N_{ij}(n)}{N_{i\bullet}(n)}, \quad \widehat{f_{ij}}(k;n) = \frac{N_{ij}(k;n)}{N_{ij}(n)}, \quad \widehat{q_{ij}}(k;n) = \frac{N_{ij}(k;n)}{N_{i\bullet}(n)}.$$

where $N_{ij}(n) = \sum_{m=1}^{N(n)-1} \mathbb{1}_{\{J_m=i;J_{m+1}=j\}}$, $N_{i\bullet}(n) = \sum_{m=1}^{N(n)-1} \mathbb{1}_{\{J_m=i\}}$,
$N_{ij}(k;n) = \sum_{m=1}^{N(n)-1} \mathbb{1}_{\{J_m=i;J_{m+1}=j;T_{m+1}-T_m=k\}}$.

- The case $q_{ij}(k) = p_{ij} f_{i\bullet}(k)$: The maximum likelihood estimators are

$$\widehat{p_{ij}}(n) = \frac{N_{ij}(n)}{N_{i\bullet}(n)}, \quad \widehat{f_{i\bullet}}(k;n) = \frac{N_{i\bullet}(k;n)}{N_{ij}(n)}, \quad \widehat{q_{ij}}(k;n) = \frac{N_{i\bullet}(k;n)}{N_{i\bullet}(n)},$$

where $N_{i\bullet}(k;n) = \sum_{m=1}^{N(n)-1} \mathbb{1}_{\{J_m=i;T_{m+1}-T_m=k\}}$.

- The case $q_{ij}(k) = p_{ij} f_{\bullet j}(k)$: The maximum likelihood estimators are

$$\widehat{p_{ij}}(n) = \frac{N_{ij}(n)}{N_{i\bullet}(n)}, \quad \widehat{f_{\bullet j}}(k;n) = \frac{N_{\bullet j}(k;n)}{N_{ij}(n)}, \quad \widehat{q_{ij}}(k;n) = \frac{N_{\bullet j}(k;n)}{N_{i\bullet}(n)},$$

where $N_{\bullet j}(k;n) = \sum_{m=1}^{N(n)-1} \mathbb{1}_{\{J_{m+1}=j;T_{m+1}-T_m=k\}}$.

- The case $q_{ij}(k) = p_{ij} f(k)$: The maximum likelihood estimators are

$$\widehat{p_{ij}}(n) = \frac{N_{ij}(n)}{N_{i\bullet}(n)}, \quad \widehat{f}(k;n) = \frac{N(k;n)}{N_{ij}(n)}, \quad \widehat{q_{ij}}(k;n) = \frac{N(k;n)}{N_{i\bullet}(n)},$$

where $N(k;n) = \sum_{m=1}^{N(n)-1} \mathbb{1}_{\{T_{m+1}-T_m=k\}}$.

Censoring: couple Markov chain For a semi-Markov chain $Y = (Y_n)_{n\in\mathbb{N}}$, let $U = (U_n)_{n\in\mathbb{N}}$ be the backward recurrence time of the SMC, defined by

$$U_n := n - T_{N(n)}. \tag{7}$$

We can show (cf. Limnios and Oprisan, 2001) that the chain $(Y,U) = (Y_n, U_n)_{n\in\mathbb{N}}$ is a Markov chain with state space $E \times \mathbb{N}$. We will denote its transition matrix by $\tilde{\mathbf{p}} := (p_{(i,t_1)(j,t_2)})_{i,j\in E, t_1,t_2\in\mathbb{N}}$.

The maximum likelihood estimators of $q_{ij}(k)$ (Trevezas and Limnios, 2011) are given by

$$\widehat{q_{ij}}(k;n) = \widehat{p}_{(i,k-1)(j,0)}(n) \prod_{t=0}^{k-2} \widehat{p}_{(i,t)(i,t+1)}(n), \tag{8}$$

where $\widehat{p}_{(i,t_1)(j,t_2)}(n)$ represents the classical MLE of the transition probability $p_{(i,t_1)(j,t_2)}$. Thus we obtain the corresponding estimator of p_{ij}

$$\widehat{p}_{ij} = \sum_{k=0}^{\infty} \widehat{q}_{ij}(k). \tag{9}$$

In order to compute the estimators of the sojourn times, we consider the four different types of semi-Markov kernels defined in Equations (2), (3), (4) and (5).

Parameters: The estimation is carried out by the function `estimSM()` and several parameters must be given.

- `file`: Path of the fasta file which contains the sequences from which to estimate
- `seq`: List of the sequence(s) from which to estimate
- `E`: Vector of state space of length S
- `TypeSojournTime`: Type of sojourn time; always equal to "NP" for the non-parametric estimation
- `cens.beg`: Type of censoring at the beginning of sample paths; 1 (if the first sojourn time is censored) or 0 (if not). All the sequences must be censored in the same way.
- `cens.end`: Type of censoring at the end of sample paths; 1 (if the last sojourn time is censored) or 0 (if not). All the sequences must be censored in the same way.

Note that the sequences from which we estimate can be given either as an R list (`seq` argument) or as a file in fasta format (`file` argument). The parameter `distr` is always equal to "NP".

```
 ## data
seqNP3_no = read.fasta("seqNP3_no.txt")
```

```
E = c("a","c","g","t")
## estimation of simulated sequences
estSeqNP3= estimSM(seq = seqNP3_no, E = E, TypeSojournTime = "fj",
                    distr = "NP", cens.end = 0, cens.beg = 0)
```

Here, we estimate simulated sequences with no censoring.

Values: The function `estimSM()` returns a list containing:

- `init`: Vector of size S with estimated initial probabilities of the semi-Markov chain

  ```
  estSeqNP3$init
  [1] 0.0000000 0.6666667 0.3333333 0.0000000
  ```

- `Ptrans`: Matrix of size $S \times S$ with estimated transition probabilities of the embedded Markov chain $J = (J_m)_m$

  ```
  estSeqNP3$Ptrans
            [,1]      [,2]      [,3]      [,4]
  [1,] 0.0000000 0.2051948 0.5090909 0.2857143
  [2,] 0.1938179 0.0000000 0.3107769 0.4954052
  [3,] 0.3010169 0.4874576 0.0000000 0.2115254
  [4,] 0.3881686 0.1936791 0.4181524 0.0000000
  ```

- `laws`: Array of size $S \times S \times Kmax$ with estimated values of the sojourn time distributions

  ```
  estSeqNP3$laws[,,1:2]
  , , 1

            [,1]      [,2]      [,3]      [,4]
  [1,] 0.0000000 0.3941423 0.4728997 0.2939271
  [2,] 0.1896104 0.0000000 0.4728997 0.2939271
  [3,] 0.1896104 0.3941423 0.0000000 0.2939271
  [4,] 0.1896104 0.3941423 0.4728997 0.0000000

  , , 2

            [,1]      [,2]      [,3]      [,4]
  [1,] 0.0000000 0.1949791 0.3089431 0.1959514
  [2,] 0.1073593 0.0000000 0.3089431 0.1959514
  [3,] 0.1073593 0.1949791 0.0000000 0.1959514
  [4,] 0.1073593 0.1949791 0.3089431 0.0000000
  ```

- q: Array of size $S \times S \times Kmax$ with estimated semi-Markov kernel

  ```
  estSeqNP3$q[,,1:3]
  , , 1

             [,1]       [,2]      [,3]       [,4]
  [1,] 0.00000000 0.07792208 0.2562771 0.06753247
  [2,] 0.03508772 0.00000000 0.1378446 0.15956558
  [3,] 0.05559322 0.18576271 0.0000000 0.06372881
  [4,] 0.07698541 0.08670989 0.1920583 0.00000000

  , , 2

             [,1]       [,2]      [,3]       [,4]
  [1,] 0.00000000 0.04329004 0.1411255 0.05627706
  [2,] 0.01670844 0.00000000 0.1019215 0.10025063
  [3,] 0.03796610 0.09762712 0.0000000 0.03864407
  [4,] 0.03889789 0.03160454 0.1385737 0.00000000

  , , 3

             [,1]       [,2]       [,3]       [,4]
  [1,] 0.00000000 0.02770563 0.07965368 0.04069264
  [2,] 0.07101086 0.00000000 0.05179616 0.03926483
  [3,] 0.09152542 0.05423729 0.00000000 0.02440678
  [4,] 0.10940032 0.01782820 0.05591572 0.00000000
  ```

Supplementary functions

In this package, others functions are available. These functions enable to compute the initial distribution for a semi-Markov model, the log-likelihood of a semi-Markov model and the AIC and BIC of a semi-Markov model.

■ `InitialLawSM()`: Estimation of initial distribution for a semi-Markov model

Parameters:

- `q`: Array of size $S \times S \times Kmax$ with estimated semi-Markov kernel

```
seq = list(c("a","c","c","g","t","a","a","a","a",
                         "g","c","t","t","t","g"))
res = estimSM(seq = seq, E = c("a","c","g","t"), distr = "NP")
Warning message:
In .comptage(J, L, S, Kmax): Warning: missing transitions
q = res$q
p = res$Ptrans

InitialLawSM(E = c("a","c","g","t"), seq = seq, q = q)
$init
[1] 0.2205882 0.2205882 0.2058824 0.3529412
```

Values: The function `InitialLawSM()` returns a list containing a vector of the initial distribution.

■ `LoglikelihoodSM ()`: Computation of the log-likelihood

Parameters:

- `E`: Vector of state space of length S
- `seq`: List of the sequence(s) from which to estimate
- `mu`: Vector of initial distribution of length S
- `Ptrans`: Matrix of transition probabilities of the embedded Markov chain $J = (J_m)_m$ of size $S \times S$
- `TypeSojournTime`: Type of sojourn time; it can be `"fij"`, `"fi"`, `"fj"` or `"f"` according to the four cases previously discussed
- `distr`: Sojourn time distributions:
  - is a matrix of distributions of size $S \times S$ if `TypeSojournTime` is equal to `"fij"`,
  - is a vector of distributions of size S if `TypeSojournTime` is equal to `"fi"` or `"fj"`,
  - is a distribution if `TypeSojournTime` is equal to `"f"`,

  where the distributions to be used can be one of `"uniform"`, `"geom"`, `"pois"`, `"weibull"` or `"nbinom"`.
- `param`: Parameters of sojourn time distributions:
  - is an array of parameters of size $S \times S \times 2$ if `TypeSojournTime` is equal to `"fij"`
  - is a matrix of parameters of size $S \times 2$ if `TypeSojournTime` is equal to `"fi"` or `"fj"`
  - is a vector of parameters if `TypeSojournTime` is equal to `"f"`
- `laws`: Sojourn time distributions introduced by the user:
  - is an array of size $S \times S \times Kmax$ if `TypeSojournTime` is equal to `"fij"`,
  - is a matrix of size $S \times Kmax$ if `TypeSojournTime` is equal to `"fi"` or `"fj"`,
  - is a vector of length $Kmax$ if `TypeSojournTime` is equal to `"f"`,

  where $Kmax$ is the maximum length for the sojourn times.

```
## state space
E = c("a","c","g","t")
S = length(E)
## creation of transition matrix
Pij = matrix( c(0,0.2,0.3,0.4,0.2,0,0.5,0.2,0.5,0.3,0,0.4,0.3,0.5,0.2,0),ncol=4)
## for the reproducibility of the results
set.seed(3)
## simulation
seq5000 = simulSM(E = E, NbSeq = 1, lengthSeq = 5000,
                                TypeSojournTime = "f", init = c(1/4,1/4,1/4,1/4),
                                Ptrans = Pij, distr = "pois", param = 2, File.out =
                                      "seq5000.txt")
## computation of the log-likelihood
LoglikelihoodSM(seq = seq5000, E = E, mu = rep(1/4,4),
                                Ptrans = Pij, distr = "pois", param = 2,
                                TypeSojournTime = "f")
$L
$L[[1]]
[1] -1748.431

$Kmax
[1] 10
```

Values: The function `likelihoodSM()` returns a list containing:

- `L`: List with the value of the likelihood for each sequence
- `Kmax`: Maximal sojourn time

We also consider model selection criteria in order to evaluate and choose among candidate models; the Akaike information criterion (AIC) and the Bayesian information criterion (BIC) are considered.

■ `AIC_SM()`: computation of the AIC

$$AIC(M) = -2\log\mathcal{L} + 2M,$$

where $\mathcal{L}$ is the log-likelihood, M is the number of parameters involved in the model

Parameters:

- `E`: Vector of state space of length S
- `seq`: List of the sequence(s) from which to estimate
- `mu`: Vector of initial distribution of length S
- `Ptrans`: Matrix of transition probabilities of the embedded Markov chain $J = (J_m)_m$ of size $S \times S$
- `TypeSojournTime`: Type of sojourn time; it can be `"fij"`, `"fi"`, `"fj"` or `"f"` according to the four cases previously discussed
- `distr`: Sojourn time distributions:
  - is a matrix of distributions of size $S \times S$ if `TypeSojournTime` is equal to `"fij"`,
  - is a vector of distributions of size S if `TypeSojournTime` is equal to `"fi"` or `"fj"`,
  - is a distribution if `TypeSojournTime` is equal to `"f"`,

  where the distributions to be used can be one of `"uniform"`, `"geom"`, `"pois"`, `"weibull"` or `"nbinom"`.
- `param`: Parameters of sojourn time distributions:
  - is an array of parameters of size $S \times S \times 2$ if `TypeSojournTime` is equal to `"fij"`
  - is a matrix of parameters of size $S \times 2$ if `TypeSojournTime` is equal to `"fi"` or `"fj"`
  - is a vector of parameters if `TypeSojournTime` is equal to `"f"`
- `laws`: Sojourn time distributions introduced by the user:
  - is an array of size $S \times S \times Kmax$ if `TypeSojournTime` is equal to `"fij"`,
  - is a matrix of size $S \times Kmax$ if `TypeSojournTime` is equal to `"fi"` or `"fj"`,

– is a vector of length *Kmax* if `TypeSojournTime` is equal to `"f"`,

where *Kmax* is the maximum length for the sojourn times.

```
## state space
E = c("a","c","g","t")
S = length(E)
lengthSeq3 = c(1000, 10000, 2000)
## creation of the initial distribution
vect.init = c(1/4,1/4,1/4,1/4)
## creation of transition matrix
Pij = matrix( c(0,0.2,0.3,0.4,0.2,0,0.5,0.2,0.5,0.3,0,0.4,0.3,0.5,0.2,0),ncol=4)
## creation of the distribution matrix
distr.matrix = matrix(c("dweibull", "pois", "geom", "nbinom",
                                                 "geom", "nbinom", "pois", "dweibull",
                                                 "pois", "pois", "dweibull", "geom",
                                                 "pois","geom", "geom", "nbinom"),
                                                 nrow = S, ncol = S, byrow = TRUE)
## creation of an array containing the parameters
param1.matrix = matrix(c(0.6,2,0.4,4,0.7,2,5,0.6,2,3,0.6,
                                                  0.6,4,0.3,0.4,4), nrow = S,
                                                ncol = S, byrow = TRUE)
param2.matrix = matrix(c(0.8,0,0,2,0,5,0,0.8,0,0,0.8,
                                                 0,4,0,0,4), nrow = S, ncol = S,
                                                byrow = TRUE)
param.array = array(c(param1.matrix, param2.matrix), c(S,S,2))
## for the reproducibility of the results
set.seed(4)
## simulation of 3 sequences
seq.crit = simulSM(E = E, NbSeq = 3, lengthSeq = lengthSeq3,
                                  TypeSojournTime = "fij", init = vect.init,
                                  Ptrans = Pij, distr = distr.matrix,
                                  param = param.array, File.out = "seq.crit.txt")
## computation of the AIC
AIC_SM(seq = seq.crit, E = E, mu = rep(1/4,4), Ptrans = Pij,
                 distr = distr.matrix, param = param.array,
                 TypeSojournTime = "fij")
[[1]]
[1] 1692.061

[[2]]
[1] 16826.28

[[3]]
[1] 3334.314
```

Values: The function `AIC_SM()` returns a list with the value of AIC for each sequence.

■ `BIC_SM()`: computation of the BIC

$$BIC(M) = -2\log\mathcal{L} + log(n)M$$

where $\mathcal{L}$ is the log-likelihood, M is the number of parameters involved in the model and n is the sample size.

Parameters:

- `E`: Vector of state space of length S
- `seq`: List of the sequence(s) from which to estimate
- `mu`: Vector of initial distribution of length S
- `Ptrans`: Matrix of transition probabilities of the embedded Markov chain $J = (J_m)_m$ of size $S \times S$
- `TypeSojournTime`: Type of sojourn time; it can be `"fij"`, `"fi"`, `"fj"` or `"f"` according to the four cases previously discussed

- `distr`: Sojourn time distributions:
  - is a matrix of distributions of size $S \times S$ if `TypeSojournTime` is equal to `"fij"`,
  - is a vector of distributions of size S if `TypeSojournTime` is equal to `"fi"` or `"fj"`,
  - is a distribution if `TypeSojournTime` is equal to `"f"`,

  where the distributions to be used can be one of `"uniform"`, `"geom"`, `"pois"`, `"weibull"` or `"nbinom"`.
- `param`: Parameters of sojourn time distributions:
  - is an array of parameters of size $S \times S \times 2$ if `TypeSojournTime` is equal to `"fij"`
  - is a matrix of parameters of size $S \times 2$ if `TypeSojournTime` is equal to `"fi"` or `"fj"`
  - is a vector of parameters if `TypeSojournTime` is equal to `"f"`
- `laws`: Sojourn time distributions introduced by the user:
  - is an array of size $S \times S \times Kmax$ if `TypeSojournTime` is equal to `"fij"`,
  - is a matrix of size $S \times Kmax$ if `TypeSojournTime` is equal to `"fi"` or `"fj"`,
  - is a vector of length $Kmax$ if `TypeSojournTime` is equal to `"f"`,

  where $Kmax$ is the maximum length for the sojourn times.

```
## computation of the BIC
BIC_SM(seq = seq3, E = E, mu = rep(1/4,4), Ptrans = Pij,
                 distr = distr.matrix, param = param.array,
                 TypeSojournTime = "fij")
[[1]]
[1] 1760.811

[[2]]
[1] 16927.22

[[3]]
[1] 3412.733
```

Values: The function `BIC_SM()` returns a list with the value of BIC for each sequence.

The Markov case

In the `SMM` R package, we have also implemented the estimation and the simulation of discrete-time multi-state Markov models. As in the semi-Markov case, other functions are available, enabling to estimate the initial distribution, to compute the log-likelihood and also the AIC and BIC of a Markov model.

■ `simulMk()`: Simulation of a Markov chain of order k

```
## state space
E <- c("a","c","g","t")
S = length(E)
vect.init <- c(1/4,1/4,1/4,1/4)
k<-2
p <- matrix(0.25, nrow = S^k, ncol = S)
## for the reproducibility of the results
set.seed(5)
## simulation of 3 sequences with the simulMk function
seq.markov = simulMk(E = E, nbSeq = 3, lengthSeq = c(1000, 10000, 2000),
           Ptrans = p, init = vect.init, k = 2, File.out= "seq.markov.txt")

seq.markov[[1]][1:25]
[1] "g" "g" "g" "g" "c" "c" "c" "a" "a" "a" "c" "c" "c" "g" "g" "c" "c" "c" "c"
 "c" "g" "g" "g" "a" "a"
```

■ `estimMk()`: Estimation of a Markov chain of order k

```
## state space
E <- c("a","c","g","t")
## for the reproducibility of the results
seq.markov = read.fasta("seq.markov.txt")
## estimation of simulated sequences
res.markov = estimMk(seq = seq.markov, E = E, k = 2)
```

Values: The function `estimMk()` returns a list containing:

- `init`: Vector of initial probabilities of the Markov chain

```
res.markov$init
[1] 0.2513810 0.2491905 0.2519048 0.2475238
```

- `Ptrans`: Matrix of transition probabilities of the Markov chain

```
res.markov$Ptrans
           [,1]      [,2]      [,3]      [,4]
 [1,] 0.2845283 0.2498113 0.2188679 0.2467925
 [2,] 0.2479645 0.2361214 0.2709104 0.2450037
 [3,] 0.2640625 0.2507813 0.2515625 0.2335937
 [4,] 0.2577475 0.2448980 0.2433862 0.2539683
 [5,] 0.2371988 0.2454819 0.2402108 0.2771084
 [6,] 0.2458629 0.2584712 0.2450749 0.2513790
 [7,] 0.2435897 0.2533937 0.2624434 0.2420814
 [8,] 0.2457887 0.2258806 0.2710567 0.2572741
 [9,] 0.2300296 0.2692308 0.2684911 0.2322485
[10,] 0.2767584 0.2194190 0.2484709 0.2553517
[11,] 0.2542248 0.2578986 0.2549596 0.2329170
[12,] 0.2377567 0.2535545 0.2440758 0.2646130
[13,] 0.2527473 0.2590267 0.2417582 0.2464678
[14,] 0.2448196 0.2578665 0.2509593 0.2463546
[15,] 0.2617602 0.2276176 0.2594841 0.2511381
[16,] 0.2390469 0.2790161 0.2559570 0.2259800
```

■ `InitialLawMk()`: Estimation of the initial distribution of a Markov chain of order k

```
seq = list(c("a","c","c","g","t","a","a","a","a","g","c","t","t","t","g"))
res = estimMk(seq = seq, E = c("a","c","g","t"), k = 1)
Warning message:
In estimMk(seq = seq, E = c("a", "c", "g", "t"), k = 1):
  missing transitions
p = res$Ptrans

InitialLawMk(E = c("a","c","g","t"), seq = seq, Ptrans = p, k = 1)
$init
[1] 0.2205882 0.2205882 0.2058824 0.3529412
```

■ `LoglikelihoodMk()`: Computation of the log-likelihood

Parameters:

- `mu`: Initial distribution
- `Ptrans`: Probability transition matrix
- `k`: Order of the Markov chain

```
## state space
E = c("a","c","g","t")
S = length(E)
## creation of transition matrix
p = matrix(rep(1/4,S*S),ncol=4)
## for the reproducibility of the results
set.seed(6)
## simulation of two sequences of length 20 and 50 respectively
seq.markov2 = simulMk(E = E, nbSeq = 2, lengthSeq = c(20,50),
                       Ptrans = p, init = rep(1/4,4), k = 1,
                       File.out = "seq.markov2.txt")
## computation of the log-likelihood
LoglikelihoodMk(seq = seq.markov2, E = E, mu = rep(1/4,4), Ptrans = p, k = 1)
$L
$L[[1]]
[1] -27.72589

$L[[2]]
[1] -69.31472
```

Values: The function `likelihoodSM()` returns a list containing the value of the likelihood for each sequence.

■ `AIC_Mk()`: Computation of the AIC for a Markov chain of order *k*

Parameters:

- `mu`: Initial distribution
- `Ptrans`: Probability transition matrix
- `k`: Order of the Markov chain

```
## for the reproducibility of the results
seq.markov2 = read.fasta("seq.markov2.txt")

## computation of the AIC
AIC_Mk(seq = seq.markov2, E = E, mu = rep(1/4,4), Ptrans = p, k = 1)
[[1]]
[1] 79.45177

[[2]]
[1] 162.6294
```

Values: The function `AIC_Mk()` returns a list containing the value of the AIC for each sequence.

■ `BIC_Mk()`: Computation of the BIC for a Markov chain of order *k*

Parameters:

- `mu`: Initial distribution
- `Ptrans`: Probability transition matrix
- `k`: Order of the Markov chain

```
## for the reproducibility of the results
seq.markov2 = read.fasta("seq.markov2.txt")

## computation of the AIC
BIC_Mk(seq = seq.markov2, E = E, mu = rep(1/4,4), Ptrans = p, k = 1)
[[1]]
[1] 91.40056

[[2]]
[1] 185.5737
```

Values: The function `BIC_Mk()` returns a list containing the value of the BIC for each sequence.

Concluding remarks

In this paper we have presented the **SMM**, an R package for the simulation and the estimation of discrete-time multi-state semi-Markov models.The conditional sojourn time can be modeled by an arbitrary distribution for a semi-Markov model, which enables a generalization with respect to Markov models, where the sojourn time is only modelled by a Geometric distribution (in discrete time) or an Exponential distribution (in continuous time). The **SMM** package offers a variety of conditional sojourn time distributions (Poisson, Uniform, Negative Binomial, Geometric and Discrete Weibull). This package provides also non-parametric estimation and simulation and takes into account censored data of several types.

To summarize, the importance and interest of the R package **SMM** that we have developed come from:

- considering versatile tools, namely discrete-time multi-state semi-Markov processes, that are of use in a variety of applied fields, like survival analysis, biology, reliability, DNA analysis, insurance and finance, earthquake modeling, meteorology studies, etc. An example of application can be the description of DNA sequences by using a 2-state process to distinguish between coding and non-coding regions in DNA sequences. It is well known that the length of such regions generally does not follow a Geometric distribution. Thus semi-Markov chains can represent adapted tools for this type of modeling.

- implementing parametric and non-parametric estimation/simulation;

- considering several censoring schemes, that important in various applications;

- taking into account one or several independent sample paths;

- considering different types of semi-Markov kernels: either of the general type $q_{ij}(k) = p_{ij}f_{ij}(k)$ with the holding time distributions $f_{ij}(k)$ depending on the current state and on the next state to be visited, or with the holding time distributions depending only on the current state, $q_{ij}(k) := p_{ij}f_{i\bullet}(k)$, or with the holding time distributions depending only on the next state to be visited, $q_{ij}(k) := p_{ij}f_{\bullet j}(k)$, or with the holding time distributions depending neither on the current, nor on the future state, $q_{ij}(k) := p_{ij}f(k)$. As already mentioned, it is important that these four types of models be considered separately.

In conclusion, the R package **SMM** that we have developed deals with an important and versatile tool, useful for researchers, practitioners and engineers in various fields.

Bibliography

V. S. Barbu and N. Limnios. Empirical estimation for discrete time semi-Markov processes with applications in reliability. *Journal of Nonparametric Statistics*, 18(4):483–498, 2006. URL https://doi.org/10.1080/10485250701261913. [p236]

V. S. Barbu and N. Limnios. *Semi-Markov Chains and Hidden Semi-Markov Models toward Applications*, volume 191 of *Lecture Notes in Statistics*. Springer-Verlag, New York, NY, 2008. ISBN 978-0-387-73171-1 978-0-387-73173-5. URL http://link.springer.com/10.1007/978-0-387-73173-5. [p226, 236]

V. S. Barbu, A. Karagrigoriou, and A. Makrides. Semi-Markov modelling for multi-state systems. *Methodology and Computing in Applied Probability*, 2016. ISSN 1573-7713. URL https://doi.org/10.1007/s11009-016-9510-y. [p226]

J. Bulla and I. Bulla. Stylized facts of financial time series and hidden semi-Markov models. *Comput. Statist. Data Anal.*, 51:2192–2209, 2006. URL https://doi.org/10.1016/j.csda.2006.07.021. [p226]

J. Bulla, I. Bulla, and O. Nenadić. Hsmm - An R package for analyzing hidden semi-Markov models. *Computational Statistics & Data Analysis*, 54(3):611–619, 2010. ISSN 0167-9473. URL https://doi.org/10.1016/j.csda.2008.08.025. [p226]

O. Chryssaphinou, M. Karaliopoulou, and N. Limnios. On discrete time semi-Markov chains and applications in words occurrences. *Comm. Statist. Theory Methods*, 37:1306–1322, 2008. URL https://doi.org/10.1080/03610920701713328. [p226]

G. D'Amico, F. Petroni, and F. Prattico. Wind speed modeled as an indexed semi-Markov process. *Environmetrics*, 24(6):367–376, 2013. ISSN 1099-095X. URL https://doi.org/10.1002/env.2215. [p226]

G. D'Amico, J. Janssen, and R. Manca. Downward migration credit risk problem: a non-homogeneous backward semi-Markov reliability approach. *Journal of the Operational Research Society*, 67(3):393–401, 2016a. ISSN 1476-9360. URL https://doi.org/10.1057/jors.2015.35. [p226]

G. D'Amico, R. Manca, C. Corini, F. Petroni, and F. Prattico. Tornadoes and related damage costs: Statistical modelling with a semi-Markov approach. *Geomatics, Natural Hazards and Risk*, 7(5): 1600–1609, 2016b. URL https://doi.org/10.1080/19475705.2015.1124462. [p226]

N. Heutte and C. Huber-Carol. Semi-Markov models for quality of life data with censoring. In M. Mesbah, M.-L. T. Lee, and B. F. Cole, editors, *Statistical Methods for Quality of Life Studies*, pages 207–218. Kluwer, Dordrecht, 2002. URL https://doi.org/10.1007/978-1-4757-3625-0_16. [p226]

J. Janssen and R. Manca. *Applied Semi-Markov Processes*. Springer-Verlag, 2006. URL https://doi.org/10.1007/0-387-29548-8. [p226]

A. Król and P. Saint-Pierre. **SemiMarkov** : An *R* Package for Parametric Estimation in Multi-State Semi-Markov Models. *Journal of Statistical Software*, 66(6), 2015. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v066.i06. [p226]

P. Lévy. Processus semi-markoviens. In *Proc. of International Congress of Mathematics, Amsterdam*, 1954. [p226]

N. Limnios and G. Oprisan. *Semi-Markov Processes and Reliability*. Birkhäuser, Boston, 2001. URL https://doi.org/10.1007/978-1-4612-0161-8. [p226, 237]

J. O'Connell and S. Højsgaard. Hidden Semi Markov Models for Multiple Observation Sequences: The **mhsmm** Package for *R*. *Journal of Statistical Software*, 39(4), 2011. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v039.i04. [p226]

B. Ouhbi and N. Limnios. Nonparametric reliability estimation for semi-Markov processes. *J. Statist. Plann. Inference*, 109(1-2):155–165, 2003. URL https://doi.org/10.1016/S0378-3758(02)00308-7. [p226]

J. Sansom and P. J. Thomson. Fitting hidden semi-Markov models to breakpoint rainfall data. *J. Appl. Probab.*, 38A:142–157, 2001. URL https://doi.org/10.1239/jap/1085496598. [p226]

W. L. Smith. Regenerative stochastic processes. *Proc. R. Soc. Lond. Ser. A Math. Phys. Eng.*, 232:6–31, 1955. URL https://doi.org/10.1098/rspa.1955.0198. [p226]

L. Takacs. Some investigations concerning recurrent stochastic processes of a certain type. *Magyar Tud. Akad. Mat. Kutato Int. Kzl.*, 3:115–128, 1954. [p226]

S. Trevezas and N. Limnios. Exact MLE and asymptotic properties for nonparametric semi-Markov models. *Journal of Nonparametric Statistics*, 23(3):719–739, 2011. URL https://doi.org/10.1080/10485252.2011.555543. [p236, 237]

I. Votsi, N. Limnios, G. Tsaklidis, and E. Papadimitriou. Estimation of the expected number of earthquake occurrences based on semi-Markov models. *Methodology and Computing in Applied Probability*, 14(3):685–703, 2012. ISSN 1573-7713. URL https://doi.org/10.1007/s11009-011-9257-4. [p226]

I. Votsi, N. Limnios, G. Tsaklidis, and E. Papadimitriou. Hidden semi-Markov modeling for the estimation of earthquake occurrence rates. *Communications in Statistics: Theory and Methods*, 43: 1484–1502, 2014. URL https://doi.org/10.1080/03610926.2013.857414. [p226]

Vlad Stefan Barbu
Université de Rouen-Normandie
Laboratoire de Mathématiques Raphaël Salem
UFR des Sciences et Techniques
Avenue de l'Université, BP. 12
76801 Saint-Étienne-du-Rouvray, France
E-mail: barbu@univ-rouen.fr
URL: http://lmrs.univ-rouen.fr/persopage/barbu

Caroline Bérard
Université de Rouen-Normandie

LITIS EA 4108
E-mail: caroline.berard@univ-rouen.fr

Dominique Cellier
Université de Rouen-Normandie
LITIS EA 4108
E-mail: dominique.cellier@laposte.net

Mathilde Sautreuil
Université de Rouen-Normandie
Laboratoire de Mathématiques Raphaël Salem
UFR des Sciences et Techniques
Avenue de l'Université, BP. 12
76801 Saint-Étienne-du-Rouvray, France
E-mail: mathilde.sautreuil@gmail.com

Nicolas Vergne
Université de Rouen-Normandie
Laboratoire de Mathématiques Raphaël Salem
UFR des Sciences et Techniques
Avenue de l'Université, BP. 12
76801 Saint-Étienne-du-Rouvray, France
E-mail: nicolas.vergne@univ-rouen.fr
URL: http://lmrs.univ-rouen.fr/persopage/nicolas-vergne

ggplot2 Compatible Quantile-Quantile Plots in R

by Alexandre Almeida, Adam Loy, Heike Hofmann

Abstract Q-Q plots allow us to assess univariate distributional assumptions by comparing a set of quantiles from the empirical and the theoretical distributions in the form of a scatterplot. To aid in the interpretation of Q-Q plots, reference lines and confidence bands are often added. We can also detrend the Q-Q plot so the vertical comparisons of interest come into focus. Various implementations of Q-Q plots exist in R, but none implements all of these features. **qqplotr** extends **ggplot2** to provide a complete implementation of Q-Q plots. This paper introduces the plotting framework provided by **qqplotr** and provides multiple examples of how it can be used.

Background

Univariate distributional assessment is a common thread throughout statistical analyses during both the exploratory and confirmatory stages. When we begin exploring a new data set we often consider the distribution of individual variables before moving on to explore multivariate relationships. After a model has been fit to a data set, we must assess whether the distributional assumptions made are reasonable, and if they are not, then we must understand the impact this has on the conclusions of the model. Graphics provide arguably the most common way to carry out these univariate assessments. While there are many plots that can be used for distributional exploration and assessment, a quantile-quantile (Q-Q) plot (Wilk and Gnanadesikan, 1968) is one of the most common plots used.

Q-Q plots compare two distributions by matching a common set of quantiles. To compare a sample, $y_1, y_2, \ldots, y_n$, to a theoretical distribution, a Q-Q plot is simply a scatterplot of the sample quantiles, $y_{(i)}$, against the corresponding quantiles from the theoretical distribution, $F^{-1}(F_n(y_{(i)}))$. If the empirical distribution is consistent with the theoretical distribution, then the points will fall on a line. For example, Figure 1 shows two Q-Q plots: the left plot compares a sample drawn from a lognormal distribution to a lognormal distribution, while the right plot compares a sample drawn from a lognormal distribution to a normal distribution. As expected, the lognormal Q-Q plot is approximately linear, as the data and model are in agreement, while the normal Q-Q plot is curved, indicating disagreement between the data and the model.

Additional graphical elements are often added to Q-Q plots in order to aid in distributional assessment. A reference line is often added to a Q-Q plot to help detection of departures from the proposed model. This line is often drawn either by tracing the identity line or by connecting two pairs of quantiles, such as the first and third quartiles, which is known as the *Q-Q line*. Pointwise or simultaneous confidence bands can be built around the reference line to display the expected degree of sampling error for the proposed model. Such bands help gauge how troubling a departure from the proposed model may be. Figure 2 adds Q-Q lines and 95% pointwise confidence bands to the Q-Q plots in Figure 1. While confidence bands help analysts interpret Q-Q plots, this practice is less

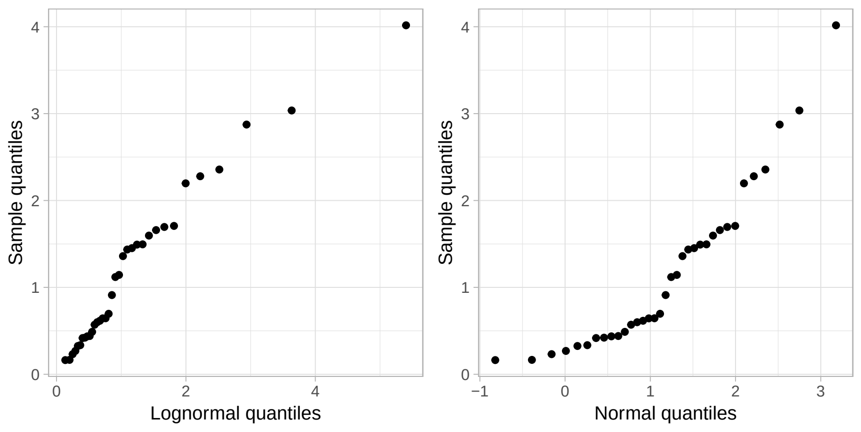


Figure 1: The left plot compares a sample of size $n = 35$ drawn from a lognormal distribution to a lognormal distribution, while the right plot compares this sample to a normal distribution. The curvature in the normal Q-Q plot highlights the disagreement between the data and the model.

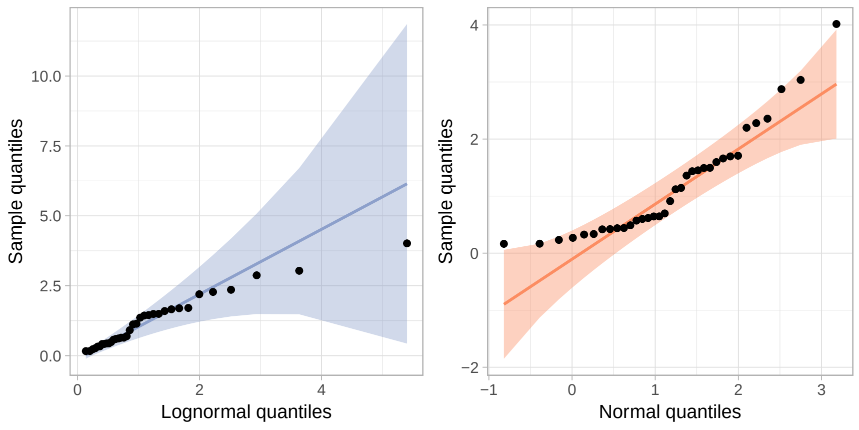


Figure 2: Adding reference lines and 95% pointwise confidence bands to the Q-Q plots in Figure 1.

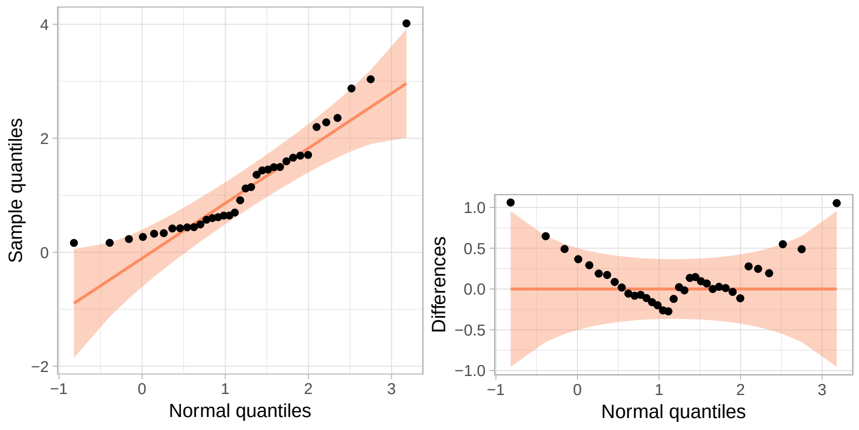


Figure 3: The left plot displays a traditional normal Q-Q plot for data simulated from a lognormal distribution. The right plot displays an adjusted detrended Q-Q plot of the same data, created by plotting the differences between the sample quantiles and the proposed model on the y-axis.

commonplace than it ought to be. One possible cause is that confidence bands are not implemented in all statistical software packages. Further, manual implementation can be tedious for the analyst, breaking the data-analytic flow—for example, a common simultaneous confidence band relies on an inversion of the Kolmogorov-Smirnov test.

Different orientations of Q-Q plots have also been proposed, most notably the *detrended Q-Q plot*. To detrend a Q-Q plot, the y-axis is changed to show the difference between the observed quantile and the reference line. Consequently, the y-axis represents agreement with the theoretical distribution. This makes the de-trended version of a Q-Q plot easier to process: cognitive research (Vander Plas and Hofmann, 2015; Robbins, 2005; Cleveland and McGill, 1984) suggests that onlookers have a tendency to intuitively assess the distance between points and lines based on the shortest distance (i.e., the orthogonal distance) rather than the vertical distance appropriate for the situation. In the de-trended Q-Q plot, the line to compare to points is rotated parallel to the x-axis, which makes assessing the vertical distance equal to assessing orthogonal distance. This is further investigated in Loy et al. (2016), who find that detrended Q-Q plots are more powerful than other designs as long as the x- and y-axes are adjusted to ensure that distances in the x- and y-directions are on the same scale. This Q-Q plot design is called an *adjusted detrended Q-Q plot*. Without this adjustment to the range of the axes, *ordinary detrended Q-Q plots* are produced, which were found to have lower power than the standard Q-Q plot in some situations (Loy et al., 2016), while the adjusted detrended Q-Q plots were found to be consistently more powerful. Figure 3 displays the normal Q-Q plot from Figure 2 along with its adjusted detrended version.

Various implementations of Q-Q plots exist in R. Normal Q-Q plots, where a sample is compared to the Standard Normal Distribution, are implemented using `qqnorm` and `qqline` in **base** graphics. `qqplot` provides a more general approach in **base** R that allows a specification of a second vector of quantiles, enabling comparisons to distributions other than a Normal. Similarly, the **lattice** package provides a general framework for Q-Q plots in the `qqmath` function, allowing comparison between a sample and any theoretical distribution by specifying the appropriate quantile function (Sarkar, 2008). qqPlot in the **car** package also allows for the assessment of non-normal distributions and adds pointwise confidence bands via normal theory or the parametric bootstrap (Fox and Weisberg, 2011). The **ggplot2** package provides geom_qq and geom_qq_line, enabling the creation of Q-Q plots with a reference line, much like those created using qqmath (Wickham, 2016). None of these general-use packages allow for easy construction of detrended Q-Q plots.

The **qqplotr** package extends **ggplot2** to provide a complete implementation of Q-Q plots. The package allows for quick construction of all Q-Q plot designs without sacrificing the flexibility of the **ggplot2** framework. In the remainder of this paper, we introduce the plotting framework provided by **qqplotr** and provide multiple examples of how it can be used.

Implementing Q-Q plots in the ggplot2 framework

qqplotr provides a **ggplot2** layering mechanism for Q-Q points, reference lines, and confidence bands by implementing separate statistical transformations (stats). In this section, we describe each transformation.

stat_qq_point

This modified version of stat_qq / geom_qq (from **ggplot2**) plots the sample quantiles against the theoretical quantiles (as in Figure 1). The novelty of this implementation is the ability to create a detrended version of the plotted points. All other transformations in **qqplotr** also allow for the detrend option. Below, we present a complete call to stat_qq_point and highlight the default values of its parameters:

```
stat_qq_point(data = NULL,
              mapping = NULL,
              geom = "point",
              position = "identity",
              na.rm = TRUE,
              show.legend = NA,
              inherit.aes = TRUE,
              distribution = "norm",
              dparams = list(),
              detrend = FALSE,
              identity = FALSE,
              qtype = 7,
              qprobs = c(0.25, 0.75),
              ...)
```

- Parameters such as data, mapping, geom, position, na.rm, show.legend, and inherit.aes are commonly found among several **ggplot2** transformations.
- distribution is a character string that sets the theoretical probability distribution. Here, we followed the nomenclature from the **stats** package, but rather than requiring the full function name for a distribution (e.g., "dnorm"), only the suffix is required (e.g., "norm"). If you wish to provide a custom distribution, then you must first create its density (PDF), distribution (CDF), quantile, and simulation functions, following the nomenclature outlined in **stats**. For example, to create the "custom" distribution, you must provide the appropriate dcustom, pcustom, qcustom, and rcustom functions. A detailed example is given in the User-provided distributions section.
- dparams is a named list specifying the parameters of the proposed distribution. By default, maximum likelihood etimates (MLEs) are used, so specifying this argument overrides the MLEs. Please note that MLEs are currently only supported for distributions available in **stats**, so if a custom distribution is provided to distribution, then *all* of its parameters must be estimated and passed as a named list to dparams.
- detrend is a logical that controls whether the points should be detrended (as in Figure 3), producing ordinary detrended Q-Q plots. For additional details on how to use this parameter and produce the more powerful adjusted detrended Q-Q plots, see the Detrending Q-Q plots section.
- identity is a logical value only used in the case of detrending (i.e., if detrend = TRUE). If identity = FALSE (default), then the points will be detrended according to the traditional Q-Q line that intersects the two data quantiles specified by qprobs (see below). If identity = TRUE, the identity line will be used instead as the reference line when constructing the detrended Q-Q plot.

- qtype and qprobs are only used when detrend = TRUE **and** identity = FALSE. These parameters are passed on to the type and probs parameters of the quantile function from **stats**, both of which are used to specify which quantiles are used to form the Q-Q line.

stat_qq_line

The stat_qq_line statistical transformation draws a reference line in a Q-Q plot.

```
stat_qq_line(data = NULL,
             mapping = NULL,
             geom = "path",
             position = "identity",
             na.rm = TRUE,
             show.legend = NA,
             inherit.aes = TRUE,
             distribution = "norm",
             dparams = list(),
             detrend = FALSE,
             identity = FALSE,
             qtype = 7,
             qprobs = c(0.25, 0.75),
             ...)
```

Nearly all of the parameters for stat_qq_line are identical to those for stat_qq_point. Hence, with the exception of identity, all other parameters have the same interpretation. For stat_qq_line, the identity parameter is *always* used, regardless of the value of detrend. This parameter controls which reference line is drawn:

a) When identity = FALSE (default), the *Q-Q line* is drawn. By default the Q-Q line is drawn through two points, the .25 and .75 quantiles of the theoretical and empirical distributions. This line provides a robust estimate of the empirical distribution, which is of particular advantage for small samples (Loy et al., 2016).
b) When identity = TRUE, the *identity line* is drawn. By definition of a Q-Q plot the identity line represents the theoretical distribution.

Both of these reference lines have a special meaning in the context of Q-Q plots. By comparing these two lines we learn about how well the parameters estimated from the sample match the theoretical parameters. For a distributional family that is invariant to linear transformations, the parameters specified in the theoretical distribution only have an effect on the Q-Q line and the Q-Q points. That is, the parameters get shifted and scaled in the plot, but relative relationships do not change aside from a change of scale on the x-axis. For other distributions, such as a lognormal distribution, re-specifications of the parameters result in non-linear transformations of the Q-Q line and Q-Q points (see Figure 4 for an example).

stat_qq_band

Confidence bands can be drawn around the reference line using one of four methods: simultaneous Kolmogorov-type bounds, a pointwise normal approximation, the parametric bootstrap (Davison and Hinkley, 1997), or the tail-sensitive procedure (Aldor-Noiman et al., 2013).

```
stat_qq_band(data = NULL,
             mapping = NULL,
             geom = "qq_band",
             position = "identity",
             show.legend = NA,
             inherit.aes = TRUE,
             na.rm = TRUE,
             distribution = "norm",
             dparams = list(),
             detrend = FALSE,
             identity = FALSE,
             qtype = 7,
             qprobs = c(0.25, 0.75),
             bandType = "pointwise",
             B = 1000,
             conf = 0.95,
             mu = NULL,
             sigma = NULL,
             ...)
```

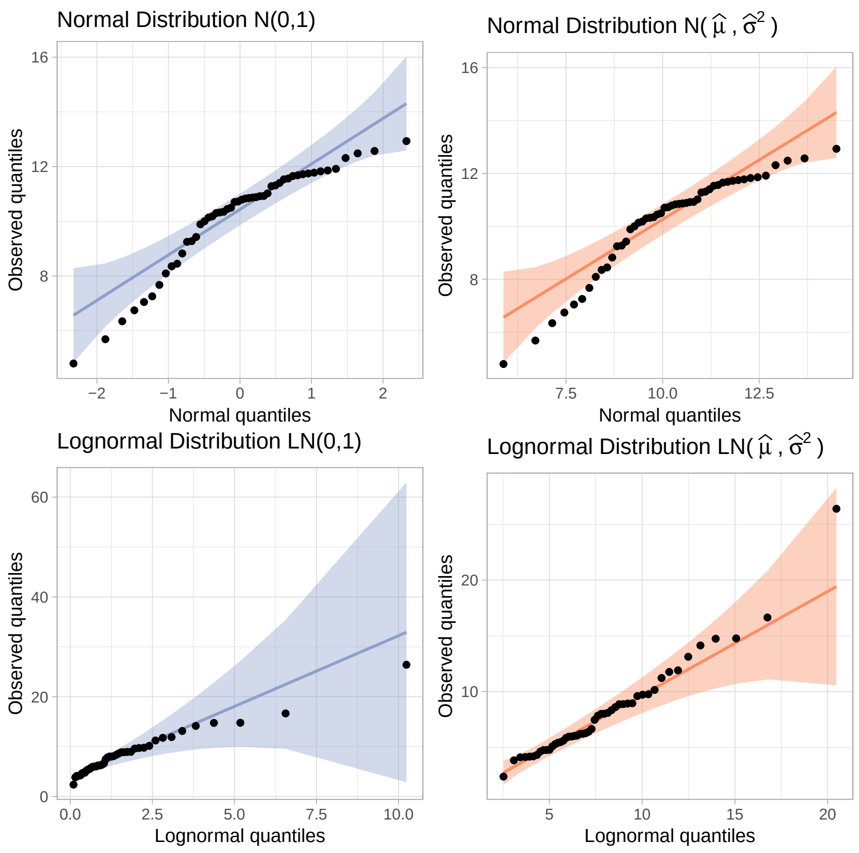


Figure 4: Q-Q plots of a sample of size $n = 50$ drawn from a normal distribution setting as theoretical the Standard Normal distribution (top-left) and a normal distribution with ML parameter estimates (top-right). Note how only the scales on the axes change between those plots. The bottom two plots show Q-Q plots of a sample of size $n = 50$ drawn from a lognormal distribution. On the left, the mean and variance of the theoretical are 0 and 1, respectively, on the log scale. On the right, ML estimates for mean and variance are used. 95% pointwise confidence bands are displayed on all Q-Q plots.

- `bandType` is a character string controlling the method used to construct the confidence bands:
  - **Simultaneous**: Specifying `bandType = "ks"` constructs simultaneous confidence bands based on an inversion of the Kolmogorov-Smirnov test. For an i.i.d. sample from CDF F, the Dvoretzky-Kiefer-Wolfowitz (DKW) inequality (Dvoretzky et al., 1956; Massart et al., 1990) states that $P(\sup_x |F(x) - \widehat{F}(x)| \geq \varepsilon) \leq 2\exp\left(-2n\varepsilon^2\right)$. Thus, lower and upper $(1-\alpha)100\%$ confidence bounds for $\widehat{F}(n)$ are given by $L(x) = \max\{\widehat{F}(n) - \varepsilon, 0\}$ and $U(x) = \min\{\widehat{F}(n) + \varepsilon, 1\}$, respectively. Confidence bounds for the points on a Q-Q plot are then given by $F^{-1}(L(x))$ and $F^{-1}(U(x))$.
  - **Pointwise:** Specifying `bandType = "pointwise"` constructs pointwise confidence bands based on a normal approximation to the distribution of the order statistics. An approximate 95% confidence interval for the ith order statistic is $\widehat{X}_{(i)} \pm \Phi^{-1}(.975) \cdot SE(X_{(i)})$, where $\widehat{X}_{(i)}$ denotes the value along the fitted reference line, $\Phi^{-1}(\cdot)$ denotes the quantile function for the Standard Normal Distribution, and $SE(X_{(i)})$ is the standard error of the ith order statistic.
  - **Bootstrap:** Specifying `bandType = "boot"` constructs pointwise confidence bands using percentile confidence intervals from the parametric bootstrap.

  - **Tail-sensitive:** Specifying `bandType = "ts"` constructs the simulation-based tail-sensitive simultaneous confidence bands proposed by Aldor-Noiman et al. (2013). Currently, tail-sensitive bands are only implemented for `distribution = "norm"`.

- `B` is a dual-purpose integer parameter. If `bandType = "boot"`, it specifies the number of bootstrap replicates. If `bandType = "ts"`, it specifies the number of simulated samples necessary to construct the tail-sensitive bands.
- `conf` is a numerical variable bound between 0 and 1 that sets the confidence level of the bands.
- `mu` and `sigma` are only used when `bandType = "ts"`. They represent the center and scale parameters, respectively, used to construct the simulated tail-sensitive confidence bands. If either is `NULL`, then *both* of the parameters are estimated using robust estimates via the **robustbase** package (Maechler et al., 2016). Currently, `bandType = "ts"` is only implemented for `distribution = "norm"`, which is the only distribution discussed by Aldor-Noiman et al. (2013).

Groups in qqplotr

`qqplotr` is implemented in accordance with the **ggplot2** concept of groups. When the user maps values to aesthetics that explicitly (by using `group`) or implicitly (such as `shape` or discrete values of `colour`, `size` etc.) introduce groups, the corresponding calculations respect the grouping in the data. All groups are compared to the same distributional family, but the parameters are estimated *separately* for each of the groups if `dparams` is not specified (which is the default for all transformations). If the user wants to fit the *same* distribution (i.e., the same parameter estimates) to each group, then the estimates must be manually calculated and passed to `dparams` as a named list for each of the desired **qqplotr** transformations. The use of groups is illustrated in more detail in the BRFSS example section.

Examples

In this section, we demonstrate the capabilities of **qqplotr** by providing multiple examples of how the package can be used. We start by loading the package:

```
library(qqplotr)
```

Constructing Q-Q plots with qqplotr

To give a brief introduction on how to use **qqplotr** and its transformations, consider the `urine` dataset from the **boot** package. This small dataset consists of 79 urine specimens that were analyzed to determine if certain physical characteristics of urine (e.g., pH or urea concentration) might be related to the formation of calcium oxalate crystals. In this example, we focus on the distributional assessment of pH measurements made on the samples.

We start by creating a normal Q-Q plot of the data. The top-left plot in Figure 5 shows a Q-Q plot comparing the pH measurements to the normal distribution. The code used to create this plot is shown below. As previously noted, the parameters of the normal distribution are automatically estimated using the MLEs when the parameters are not otherwise specified in `dparams`. The shaded region represents the area between the normal pointwise confidence bands. As we can see, the distribution of urine pH measurements is somewhat right-skewed.

```
library(dplyr) # for using `%>%` and later data transformation
data(urine, package = "boot")
urine %>%
  ggplot(aes(sample = ph)) +
  stat_qq_band(bandType = "pointwise", fill = "#8DA0CB", alpha = 0.4) +
  stat_qq_line(colour = "#8DA0CB") +
  stat_qq_point() +
  ggtitle("Normal") +
  xlab("Normal quantiles") +
  ylab("pH measurements quantiles") +
  theme_light() +
  ylim(3.2, 8.7)
```

Figure 5 also provides an overview of **qqplotr**'s capabilities:

- The left column displays Q-Q plots with 95% pointwise confidence bands obtained from a normal approximation.

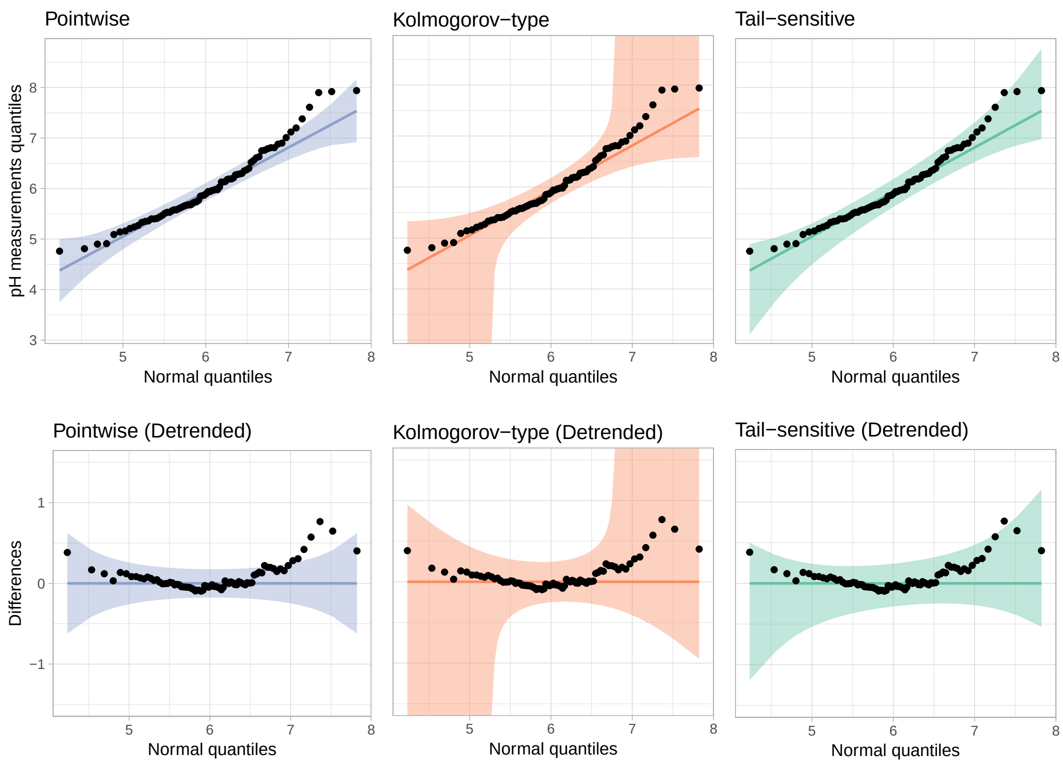


Figure 5: Normal Q-Q plots of pH measurements from urine samples using different confidence bands. Depending on the type of confidence band used, we come to different conclusions.

- The center column displays Q-Q plots with 95% Kolmogorov-type simultaneous confidence bands.
- The right column displays Q-Q plots with 95% tail-sensitive simultaneous confidence bands. Notice that these are substantially narrower in the tails than the Kolmogorov-type bands.
- The bottom row shows the detrended versions of the Q-Q plots in the top row.

User-provided distributions

Using the capabilities of **qqplotr** with the distributions implemented in **stats** is relatively straight-foward, since the implementation allows you to specify the suffix (i.e., distribution or abbreviation) via the `distribution` argument and the parameter estimates via the `dparams` argument. However, there are times when the distributions in **stats** are not sufficient for the demands of the analysis. For example, there is no left-skewed distribution listed aside from the beta distribution, which has a restrictive support. User-coded distributions, or distributions from other packages, can be used with **qqplotr** as long as the distributions are defined following the conventions laid out in **stats**. Specfically, for some distribution there must be density/mass (`d` prefix), CDF (`p` prefix), quantile (`q` prefix), and simulation (`r` prefix) functions. In this section, we illustrate the use of the smallest extreme value distribution (SEV).

To qualify for the 2012 Olympics in the men's long jump, athletes had to meet/exceed the 8.1 meter standard or place in the top twelve. During the qualification events, each athlete was able to jump up to three times, using their best (i.e., longest) jump as the result. Figure 6 shows a density plot of the results, which is clearly left skewed.

We start by loading the `longjump` dataset included in **qqplotr** and removing any `NA`s:

```
data("longjump", package = "qqplotr")
longjump <- na.omit(longjump)
```

Next, we define the suite of distributional functions necessary to utilize the SEV distribution.

```
# CDF
psev <- function(q, mu = 0, sigma = 1) {
  z <- (q - mu) / sigma
  1 - exp(-exp(z))
```

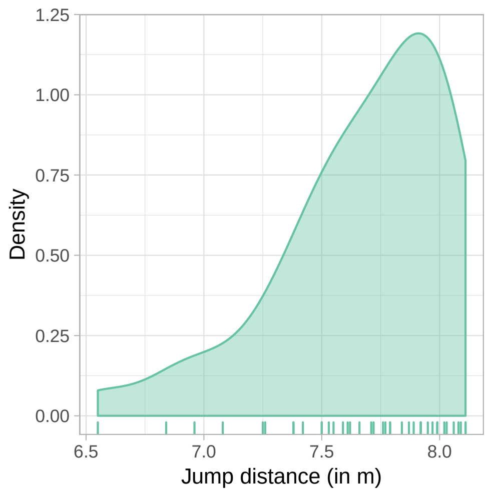


Figure 6: Density and rug plot of the 2012 men's long jump qualifying round. The distances are clearly left skewed.

```
}

# PDF
dsev <- function(x, mu = 0, sigma = 1) {
  z <- (x - mu) / sigma
  (1 / sigma) * exp(z - exp(z))
}

# Quantile function
qsev <- function(p, mu = 0, sigma = 1) {
  mu + log(-log(1 - p)) * sigma
}

# Simulation function
rsev <- function(n, mu = 0, sigma = 1) {
  qsev(runif(n), mu, sigma)
}
```

With the *sev distribution functions in hand, we can create a Q-Q plot to assess the appropriateness of the SEV model (Figure 7). The Q-Q plot shows that the distances do not substantially deviate from the SEV model, so we have found an adequate representation of the distances. The code used to create Figure 7 is given below:

```
ggplot(longjump, aes(sample = distance)) +
  stat_qq_band(distribution = "sev",
               bandType = "ks",
               dparams = list(mu = 0, sigma = 1),
               fill = "#8DA0CB",
               alpha = 0.4) +
  stat_qq_line(distribution = "sev",
               colour = "#8DA0CB",
               dparams = list(mu = 0, sigma = 1)) +
  stat_qq_point(distribution = "sev",
                dparams = list(mu = 0, sigma = 1)) +
  xlab("SEV quantiles") +
  ylab("Jump distance (in m)") +
  theme_light()
```

Detrending Q-Q plots

To illustrate how to construct an adjusted detrended Q-Q plot using **qqplotr**, consider detrending Figure 7. This is done by adding the argument `detrend = TRUE` to `stat_qq_point`, `stat_qq_line`, and `stat_qq_band`. To adjust the aspect ratio to ensure that vertical and horizontal distances are on the

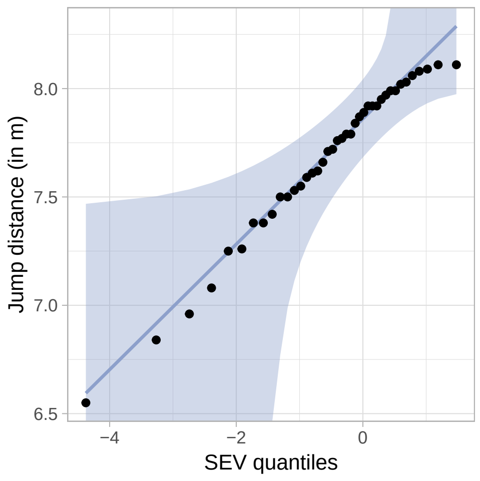


Figure 7: Q-Q plot comparing the long jump distances to the standard SEV distribution with 95% simultaneous confidence bands. The SEV distribution appears to adequately model the distances.

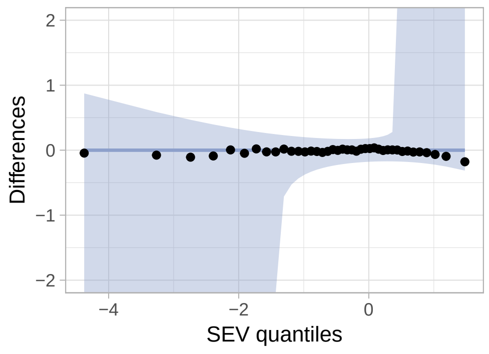


Figure 8: An adjusted detrended Q-Q plot assessing the appropriateness of the SEV distribution for the long jump data.

same scale we further add `coord_fixed(ratio = 1)`. We leave it to the user to adjust the y-axis limits on a case-by-case basis. The full command to construct Figure 8 is given below:

```
ggplot(longjump, aes(sample = distance)) +
  stat_qq_band(distribution = "sev",
               bandType = "ks",
               detrend = TRUE,
               dparams = list(mu = 0, sigma = 1),
               fill = "#8DA0CB",
               alpha = 0.4) +
  stat_qq_line(distribution = "sev",
               detrend = TRUE,
               dparams = list(mu = 0, sigma = 1),
               colour = "#8DA0CB") +
  stat_qq_point(distribution = "sev",
                detrend = TRUE,
                dparams = list(mu = 0, sigma = 1)) +
  xlab("SEV quantiles") +
  ylab("Differences") +
  theme_light() +
  coord_fixed(ratio = 1, ylim = c(-2, 2))
```

BRFSS example

The Center for Disease Control and Prevention runs an annual telephone survey, the Behavioral Risk Factor Surveillance System (BRFSS), to track "health-related risk behaviors, chronic health conditions, and use of preventive services" (Centers for Disease Control and Prevention, 2014). Close to half a million interviews are conducted each year. In this example, we focus on the responses for Iowa in 2012. The data set consists of 7166 responses across 359 questions and derived variables. To further illustrate the functionality of **qqplotr**, we focus on assessing the distributions of Iowan's heights and weights.

Figure 9 shows two Q-Q plots constructed from a sample of 200 men and 200 women drawn from the overall number of responses. On the left-hand side, individuals' heights are displayed in a Q-Q plot comparing raw heights to a normal distribution. We see that the distributions for both men and women show horizontal steps: this indicates that the distributional assessement is heavily dominated by the discreteness in the data, as most respondents provided their height to the nearest inch. On the right-hand side of Figure 9, we use jittering to remedy this situation. That is, we add a random number generated from a random uniform distribution on ± 0.5 inch to the reported height, as shown in the code below:

```
data("iowa", package = "qqplotr")
set.seed(3145)

sample_ia <- iowa %>%
  tidyr::nest(-SEX) %>%
  mutate(
    data = data %>%
    purrr::map(.f = function(x) sample_n(x, size = 200))
    ) %>%
  tidyr::unnest(data) %>%
  dplyr::select(SEX, WTKG3, HTIN4) %>%
  mutate(Gender = c("Male", "Female")[SEX])

params <- iowa %>%
  filter(!is.na(HTIN4)) %>%
  summarize(m = mean(HTIN4), s = sd(HTIN4))

customization <- list(scale_fill_brewer(palette = "Set2"),
                      scale_colour_brewer(palette = "Set2"),
                      xlab("Normal quantiles"),
                      ylab("Height (in.)"),
                      coord_equal(),
                      theme_light(),
                      theme(legend.position = c(0.8, 0.2), aspect.ratio = 1))

sample_ia %>%
  ggplot(aes(sample = HTIN4, colour=Gender, fill=Gender)) +
  stat_qq_band(bandType = "ts",
               alpha = 0.4,
               dparams = list(mean = params$m, sd = params$s)) +
  stat_qq_point(dparams = list(mean = params$m, sd = params$s)) +
  customization

sample_ia %>%
  mutate(HTIN4.jitter = jitter(HTIN4, factor = 2)) %>%
  ggplot(aes(sample = HTIN4.jitter, colour = Gender, fill = Gender)) +
  stat_qq_band(bandType = "ts",
               alpha = 0.4,
               dparams = list(mean = params$m, sd = params$s)) +
  stat_qq_line(dparams = list(mean = params$m, sd = params$s)) +
  stat_qq_point(dparams = list(mean = params$m, sd = params$s)) +
  customization +
  ylab("Jittered Height (in.)")
```

Notice that the same theoretical normal distribution was fit to both genders as specified in `dparams`. If we had used the default, then the MLEs for each gender would be used. As a result, we would be comparing the two genders over a different range of theoretical quantiles. By explicity providing parameter estimates for the mean and standard deviation via `dparams`, we force the Q-Q plots to use the same x-coordinates (theoretical quantiles), which is more useful when comparing the distribution of these groups.

As seen in Figure 9, by using jittering we diminish the effect that discreteness has on the distribution

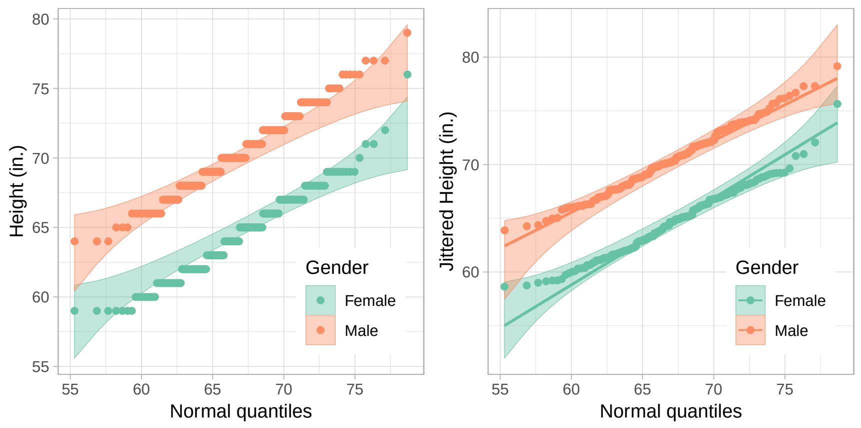


Figure 9: Q-Q plots comparing the raw (left) and jittered (right) heights to a normal distribution for a sample of 200 men and 200 women. The distribution on the left is dominated by the discreteness of the data. On the right, using a normal distribution to model people's height is not completely absurd, except for a few extreme outliers.

Table 1: Summary of Iowa residents' heights and weights with corresponding standard deviations by gender and for the total population.

SEX	mean height (in)	sd (in)	mean log weight (kg)	sd (kg)
Male	70.55	2.97	9.10	0.20
Female	64.51	2.91	8.89	0.23
Total	66.99	4.18	8.98	0.24

and brings the observed distribution much closer to a normal distribution. Unsurprisingly, the resulting distributions have different means (women are, on average, 6 inches shorter than men in this data set). Interestingly, the slope of the two genders is similar, indicating that the same scale parameter fits both genders' distributions (the standard deviation of height in the data set is 2.97 inch for men and 2.91 inch for women, see Table 1).

Unlike respondents' heights, their weights do not seem to be normally distributed. Figure 10 shows two Q-Q plots of these data. For both, distributional parameters are estimated separately for each group. This means that for each group we compare against its theoretical distribution shown as the identity line. The Q-Q plot on the left compares raw weights to a normal distribution. We see that tails of the observed distribution are heavier than expected under a normal distribution. On the right, weights are log-transformed. We see that using a normal distribution for each gender appears to be reasonable, with the exception of a few extreme outliers. The code used to create Figure 10 is shown below:

```
sample_ia %>%
  ggplot(aes(sample = WTKG3 / 100, colour = Gender, fill = Gender)) +
  geom_abline(colour = "grey40") +
  stat_qq_band(bandType = "ts", alpha = 0.4) +
  stat_qq_line() +
  stat_qq_point() +
  customization +
  ylab("Weight (kg.)")

sample_ia %>%
  ggplot(aes(sample = log(WTKG3/100), colour=Gender, fill=Gender)) +
  geom_abline(colour = "grey40") +
  stat_qq_band(bandType = "ts", alpha = 0.4) +
  stat_qq_line() +
  stat_qq_point() +
  customization +
  ylab("log Weight (kg.)")
```

Instead of log-transforming the observed weights, we can change the theoretical distribution to a lognormal. Figure 11 shows two lognormal Q-Q plots, one for each gender. Note the MLEs are used to

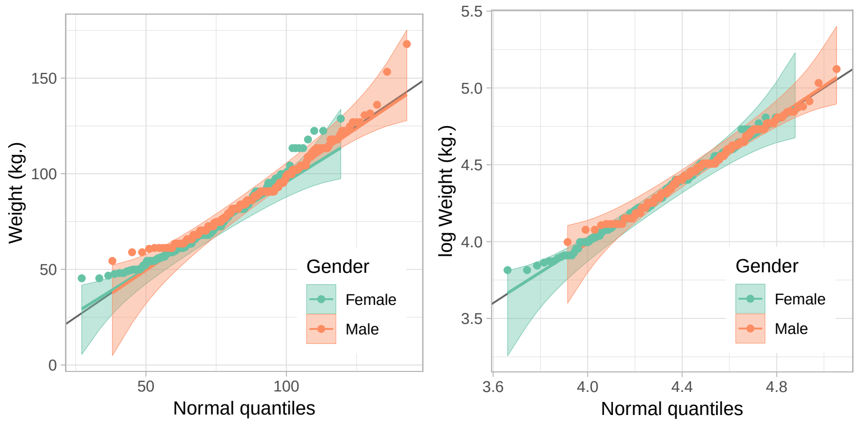


Figure 10: Q-Q plots comparing weights to a normal distribution for a sample of 200 men and 200 women. Unlike people's height, weight seems to be right skewed with some additional outliers in the left tail (left plot). On the right, weight was log-transfomed before its distribution is compared to a theoretical normal.

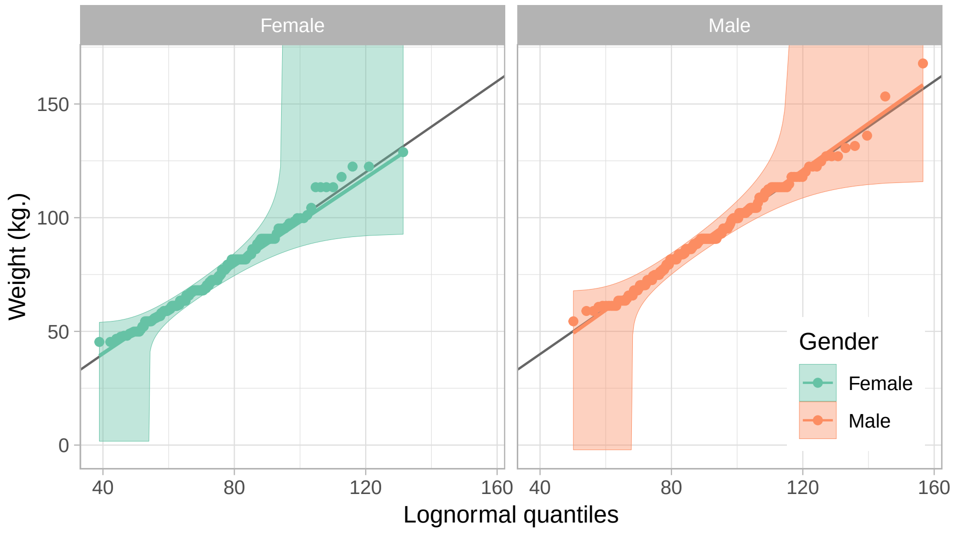


Figure 11: Q-Q plots comparing weights to a lognormal distribution for a sample of 200 men and 200 women. The parameters are estimated separately for each gender to specify the lognormal reference distribution for each gender.

parameterize the lognormal distribution for each group since dparams is not specified:

```
sample_ia %>%
  ggplot(aes(sample = WTKG3 / 100, colour = Gender, fill = Gender)) +
  geom_abline(colour = "grey40") +
  stat_qq_band(bandType = "ks", distribution = "lnorm", alpha = 0.4) +
  stat_qq_line(distribution = "lnorm") +
  stat_qq_point(distribution = "lnorm") +
  customization +
  facet_grid(. ~ Gender) +
  xlab("Lognormal quantiles") +
  ylab("Weight (kg.)") +
  theme(legend.position = c(0.9, 0.2))
```

Discussion

This paper presented the **qqplotr** package, an extension of **ggplot2** that implements Q-Q plots in both the standard and detrended orientations, along with reference lines and confidence bands. The examples illustrated how to create Q-Q plots for non-standard distributions found outside of the **stats** package, how to create detrended Q-Q plots, and how to create Q-Q plots when data are grouped. Further, in the BRFSS example, we illustrated how jittering can be used in Q-Q plots to better compare discretized data to a continuous distribution.

While **qqplotr** provides a complete implementation of the Q-Q plot, there is room for development in future versions. For example, Q-Q plots are members of the larger probability plotting family, so future versions of **qqplotr** will likely include additional members of that family.

Finally, we have made design choices in **qqplotr** that we believe are in line with best practices for distributional assessment, but the implementation is flexible enough to allow for easy customization. For example, maximum likelihood is used to estimate the parameters of the proposed model, but if outliers are present robust estimators may be desirable, such as when comparing the empirical distribution to a normal distribution. In this scenario, robust estimates of the location and scale can be obtained using the **robustbase** package (Maechler et al., 2016), and specified using the `dparams` parameter directly. This is especially useful if you wish to use the parametric bootstrap to build confidence bands. Similarly, `stat_qq_line` implements two types of reference lines: the identity line, and the traditional Q-Q line that passes through two quantiles of the distributions, such as the first and third quartiles. While those are the most conventionally used reference lines, alternative ones can be quickly implemented using `ggplot2::geom_abline` by specifying the slope and intercept. By default, the Q-Q line is used; however, this is not always the most appropriate choice. In order to *test* whether the data follow a specific distribution, the reference line should be used rather than using the data twice: once to estimate the parameters, and once for comparison.

Bibliography

S. Aldor-Noiman, L. D. Brown, A. Buja, W. Rolke, and R. A. Stine. The power to see: A new graphical test of normality. *The American Statistician*, 67(4):249–260, 2013. URL https://doi.org/10.1080/00031305.2013.847865. [p251, 253]

Centers for Disease Control and Prevention. Behavioral risk factor surveillance system. https://www.cdc.gov/brfss/about/index.htm, 2014. Accessed: 2017-11-1. [p256]

W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984. ISSN 01621459. URL https://doi.org/10.2307/2288400. [p249]

A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Application*. Cambridge University Press, Cambridge, 1997. ISBN 9780521574716. [p251]

A. Dvoretzky, J. Kiefer, and J. Wolfowitz. Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator. *Annals of Mathematical Statistics*, 27(3):642–669, 9 1956. URL https://doi.org/10.1214/aoms/1177728174. [p252]

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, second edition, 2011. ISBN 9781412975148. [p250]

A. Loy, L. Follett, and H. Hofmann. Variations of Q–Q plots: The power of our eyes! *The American Statistician*, 70(2):202–214, 2016. URL https://doi.org/10.1080/00031305.2015.1077728. [p249, 251]

M. Maechler, P. Rousseeuw, C. Croux, V. Todorov, A. Ruckstuhl, M. Salibian-Barrera, T. Verbeke, M. Koller, E. L. T. Conceicao, and M. Anna di Palma. *robustbase: Basic Robust Statistics*, 2016. URL http://robustbase.r-forge.r-project.org/. R package version 0.92-7. [p253, 260]

P. Massart et al. The tight constant in the Dvoretzky-Kiefer-Wolfowitz inequality. *The Annals of Probability*, 18(3):1269–1283, 1990. URL https://doi.org/10.1214/aop/1176990746. [p252]

N. Robbins. *Creating More Effective Graphs*. Wiley, Hoboken, 2005. ISBN 0-471-27402-x. [p249]

D. Sarkar. *lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. ISBN 978-0-387-75968-5. [p249]

S. Vander Plas and H. Hofmann. Signs of the sine illusion – and why we need to care. *Journal of Computational and Graphical Statistics*, 25:1170–1190, 2015. URL https://doi.org/10.1080/10618600.2014.951547. [p249]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. [p250]

M. B. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55 (1):1–17, 1968. URL https://doi.org/10.2307/2334448. [p248]

Acknowledgements

This work was partially funded by Google Summer of Code 2017. We thank the reviewers for their helpful comments and suggestions.

Alexandre Almeida
University of Campinas
Institute of Computing
Campinas, Brazil 13083-852
almeida.xan@gmail.com

Adam Loy
Carleton College
Department of Mathematics and Statistics
Northfield, MN 55057
ORCiD: 0000-0002-5780-4611
aloy@carleton.edu

Heike Hofmann
Iowa State University
Department of Statistics
Ames, IA 50011-1210
ORCiD: 0000-0001-6216-5183
hofmann@iastate.edu

Forecast Combinations in R using the ForecastComb Package

by Christoph E. Weiss and Eran Raviv and Gernot Roetzer

Abstract This paper introduces the R package **ForecastComb**. The aim is to provide researchers and practitioners with a comprehensive implementation of the most common ways in which forecasts can be combined. The package in its current version covers 15 popular estimation methods for creating a combined forecasts – including simple methods, regression-based methods, and eigenvector-based methods. It also includes useful tools to deal with common challenges of forecast combination (e.g., missing values in component forecasts, or multicollinearity), and to rationalize and visualize the combination results.

Introduction

Model uncertainties and/or model instabilities are deeply-rooted in real-life forecasting challenges. More often than not, and particularly in soft sciences including econometrics, the underlying data-generating process is not well understood.

In the business of time series forecasting, accuracy is key. The advent of "black-box" statistical learning methods (such as *Artificial Neural Networks*) testifies to the fact that forecasting practitioners and researchers alike tend to favor accuracy over explainability of a forecast model. There is a strong consensus in the social sciences that the observed processes are too complex to be modelled perfectly. Excluding some natural sciences, it is generally undisputed that the underlying data-generating process is unknown[1]. Even in the unrealistic case where the structural form of a data-generating process is more or less known, except for the parameter values that has to be estimated, a misspecified model can produce better forecasts than a correctly specified model due to parameters' estimation uncertainty[2].

Hence, statistical models are habitually too simple, misspecified, and/or incomplete; a fact that is widely accepted in theory, but less widely applied in practice, in the field of econometrics, in which researchers often still hang on to the conceptual error of assuming one true data-generating process and putting too much focus on model selection to find the one true model. Hansen (2005) takes note of this and related misconceptions of econometric forecasting practice in his essay on the challenges for econometric model selection: "models should be viewed as approximations, and econometric theory should take this seriously". An obvious alternative to choosing a single 'best' forecasting method is to combine forecasts from different models. A combined forecast can also be thought of as one which originates from a very complex model, an overarching model with different underlying simpler models in order to achieve a better data-generating process approximation. By now it is well established, empirically but in many different settings, that combining different forecasts delivers results which are "better than the best".

Following the advice in Hansen (2005), we abandon the quest for the one *true* correctly specified model, so that we are free to include information from different models. While this freedom, as seen empirically, improve forecast accuracy it is generally more intricate to interpret a forecast combined from different models compared to a forecast generated from a single model. In the context of forecasting, this means combining forecasts from different sources (models, experts). This emphasizes the aforementioned shift in approaching practical econometric forecasting problems: model misspecification cannot be fully rectified, but strategies can be found to mitigate its adverse effects on forecast quality. Selecting a single "best" forecasting model bears the risk of ending up with a model which is only accurate when evaluated using some validation sample, yet might prove unreliable when applied to new data. In the past decades, ample empirical evidence on the merits of combining forecasts has piled up; it is generally accepted that the (mostly linear) combination of forecasts from different models is an appealing strategy to hedge against forecast risk. Even though reduction of forecast risk is the main argument for using combined forecasts, it should also be noted that there are cases when combined forecasts are more accurate than even its best component (e.g. Graefe et al., 2014). While this is not theoretically sound, it is somewhat intuitive that in a continuously changing environment different forecasting models deliver different results at different time periods. For example, it is reasonable for one method to perform well in a low-volatility regime, and for another method to perform poorly in that regime, but perform well in a high-volatility regime. Strong empirical support for this argument of change in relative performance of different methods over time is found among

[1]Referring for example to George's Box well rehearsed aphorism: "All models are wrong but some are useful".
[2]We thank an anonymous referee for pointing this out.

others by Elliott and Timmermann (2005).

"Hedging" against model risk by combining different forecasts is intuitive and appealing. Joined with accuracy gains, the idea is widely adopted in macroeconomics and finance, with application abound. Magnus and Wang (2014) explore growth determinants, Stock and Watson (2004) use forecast combination for output forecasting, Wright (2009) and Kapetanios et al. (2008) for inflation forecasting, Avramov (2002), Rapach et al. (2010) and Ravazzolo et al. (2007) for forecasting stock returns, Wright (2008) for exchange rate forecasts, Andrawis et al. (2011) for forecasting inbound tourism figures, Nowotarski et al. (2014) and Raviv et al. (2015) for electricity price forecasting, and Weiss (2017) for healthcare demand forecasting. More recently, forecast combinations have been applied not only in "first moment" applications, but for higher moments as well. For example in Christiansen et al. (2012) for volatility forecasting, and Opschoor et al. (2014) for Value-at-Risk forecasting.

The theoretical foundations of forecast combination date back five decades to the seminal papers of Crane and Crotty (1967) and Bates and Granger (1969). Yet even in the recent past, papers discussing new combination techniques are published in reputable journals and stimulate further research still (Hansen, 2007, 2008; Hansen and Racine, 2012; Elliott et al., 2013; Morana, 2015; Cheng and Yang, 2015). Two extensive reviews of the literature, techniques and applications of forecast combinations are Clemen (1989) and Timmermann (2006).

Given the topic's popularity in both theoretical and empirical research, it is somewhat surprising that very few combination techniques are readily available in R: There are some packages that cover specialized specific topics related to combination methods, e.g. the **BMA** package by Raftery and Painter (2005) for Bayesian model averaging, as well as the **opera** package by Gaillard and Goude (2016) and the **forecastHybrid** package by Shaub and Ellis (2017). However, as of now the only two R packages that are entirely devoted to forecast combination are the **ForecastCombinations** package by Raviv (2015), which focuses on regression-based combination methods, and the **GeomComb** package by Weiss and Roetzer (2016), which focuses on geometric, eigenvector-based combination methods.

Aiming to improve user experience, we have merged these last two aforementioned packages, providing one unified package for the widest range of forecast combination approaches available today in R. The package is flexible and provides enough guidance for users familiar or unfamiliar with the world of forecasting. We have made both regression-based and eigenvector-based combination methods available to users in a single standardized framework based on S3 classes and methods. The logic behind this choice is that comparing regression-based and eigenvector-based combination methods is often insightful – as pointed out by Hsiao and Wan (2014), the conditions under which these two approaches perform well differ from each other: regression-based methods tend to produce more accurate forecasts when one or a few of the individual forecasts are considerably better than the rest, while eigenvector-based methods perform better when the individual forecasts are in the same ballpark. This paper presents the functionalities made available in the package and demonstrates how to implement them in an empirical exercise.

In the following section we review the forecast combination methods that are available in the **ForecastComb** package. We then provides a detailed implementation description using the package, with a specific empirical example – we combine univariate time series forecasts for UK energy supply.

Forecast combinations

Since the seminal paper by Bates and Granger (1969), myriad combination strategies have been put forward in theoretical and empirical literature. The best way to combine different forecasts has no theoretical underpinnings, a lot depends on the specifics of the data at hand. Andrawis et al. (2011) even suggest to use hierarchical forecast combinations, i.e. combining combined forecasts. The purpose here is not to investigate which combination method is suitable in which context. Rather, we provide a broad range of options for the user to explore and experiment with. Again, owing to a lack of theoretical foundation or the empirical lack of evidence for a one dominant way in which forecasts should be combined.

To fix notations, denote $F_{T \times P}$ as the matrix of forecast with dimension $T \times P$ where T is the number of rows and P is the number of columns (so we have P forecasts at each point in time). Denote f_i as the forecast obtained using model i, $i \in \{1 \dots P\}$. When there is no danger of confusion, we omit the additional subscript t which denotes the time dimension of the forecast. Some combination methods require an ordering of the component forecasts. When this is the case, $f_{(i)}$ denotes the ith order statistic of the cross-section of component forecasts. Finally, the weight given to that forecast in the overall combined forecast is denoted as w_i, and the combined forecast as f^c.

Frequently used schemes for forecast combinations

Simple Combination Methods

We now present few simple ways in which forecasts can be combined. They are simple in that there is no need to exactly estimate the weight each forecast should be given in the overall combination. We just use some location measure (for example the average) of the cross-sectional distribution of the individual forecasts. In theory some location measures are the same if the cross-sectional distribution is symmetric, but if the distribution is asymmetric, and/or if there is one method which produces extreme forecasts then the following few combination methods become pertinent.

1. *Simple Average*. The most intuitive approach to combine forecasts is using the average of all those forecasts. Over the years this innocent approach has established itself as an excellent benchmark, despite or perhaps because of its simplicity (e.g. Genre et al., 2013). The combined forecast is straightforwardly given by

$$f^c = \frac{1}{P}\sum_{i=1}^{P} f_i. \tag{1}$$

   Clemen (1989) argues that this equal weighting of component forecasts is often the best strategy in this context. This is still true almost thirty years later and called the "forecast combination puzzle", a term coined by Stock and Watson (2004). A rigorous attempt to explain why simple average weights often outperform more sophisticated forecast combination techniques is provided in a simulation study by Smith and Wallis (2009), who ascribe this surprising empirical finding to the effect of finite-sample error in estimating the combination weights. Recently, Claeskens et al. (2016) provide a theoretical argumentation to these empirical findings. The authors make the case that lower estimation noise, when the weights are determined rather than estimated, goes a long way in explaning the puzzle. A more detailed overview of the empirical support for the "forecast combination puzzle" can be found in Graefe et al. (2014).

2. *Median*. Another fairly simple and appealing combination method is using the median of the component forecasts. The median is insensitive to outliers, which can be relevant for some applications. Palm and Zellner (1992) suggest that simple averaging may not be a suitable combination method when some of the component forecasts are biased. This calls for the use of another location measures which is robust to outliers. The median method is an appealing, rank-based combination method that has been used in a wide range of empirical studies (e.g. Armstrong, 1989; McNees, 1992; Hendry and Clements, 2004; Stock and Watson, 2004; Timmermann, 2006).

   For the median method, the combined forecast is given by:

   - For odd P:

$$f^c = f_{(\frac{P}{2}+0.5)} \tag{2}$$

   - For even P:

$$f^c = \frac{1}{2}\left(f_{(\frac{P}{2})} + f_{(\frac{P}{2}+1)}\right) \tag{3}$$

3. *Trimmed Mean*. Another outlier-robust location measure that is commonly used is the trimmed mean (e.g. Armstrong, 2001; Stock and Watson, 2004; Jose and Winkler, 2008).

   Using a trim factor λ (i.e. the top/bottom $100 \times \lambda\%$ are trimmed) the combined forecast is calculated as:

$$f^c = \frac{1}{P(1-2\lambda)}\sum_{i=\lambda P+1}^{(1-\lambda)P} f_{(i)} \tag{4}$$

   Typically, we use $\lambda = 0.1$ indicating we trim the top and bottom 10% of the most extreme component forecasts, excluding those from the computation of the combined forecast. The trimmed mean is an interpolation between the simple average ($\lambda = 0$) and the median ($\lambda = 0.5$).

4. *Winsorized Mean*. Like the trimmed mean, the winsorized mean is a robust statistic that is less sensitive to outliers than the simple average. It takes a softer line when handling outliers: Instead of altogether removing them as in the trimmed mean approach, it caps outliers at a

certain level. By capping outliers rather than removing them, we allow for at least some degree of influence. For this reason, the measure is sometimes preferred, for example by Jose and Winkler (2008).

Let λ be the trim factor (i.e. the top/bottom $100 \times \lambda\%$ are winsorized) and $K = \lambda P$. The combined forecast is then calculated as:

$$f^c = \frac{1}{P}\left[Kf_{(K+1)} + \sum_{i=K+1}^{P-K} Kf_{(P-K)}\right] \tag{5}$$

5. *Bates/Granger (1969).* In their seminal paper, Bates and Granger (1969) introduced the idea of combining forecasts. Their approach builds on portfolio diversification theory and uses the diagonal elements of the estimated mean squared prediction error matrix in order to compute combination weights:

$$f^c = \sum_{i=1}^{P} f_i' \times \frac{\hat{\sigma}^{-2}(i)}{\sum_{j=1}^{P} \hat{\sigma}^{-2}(j)} \tag{6}$$

where $\hat{\sigma}^2(i)$ is the estimated mean squared prediction error of model i.

The approach ignores correlation between component forecasts due to difficulties in precisely estimating the covariance matrix.

6. *Newbold/Granger (1974).* Building on the earlier research by Bates and Granger (1969), the methodology of Newbold and Granger (1974) also extracts the combination weights from the estimated mean squared prediction error matrix.

Let Σ be the mean squared prediction error matrix of $F_{N\times P}$ and e be a $P \times 1$ vector of $(1, 1, ..., 1)'$. Newbold and Granger (1974)'s method is a constrained minimization of the mean squared prediction error using the normalization condition $e'w = 1$. This yields the following combined forecast:

$$f^c = F_{N\times P} \times \frac{\Sigma^{-1}e}{e'\Sigma^{-1}e} \tag{7}$$

While the method dates back to Newbold and Granger (1974), the variant of the method we use in the **ForecastComb** package does not impose the prior restriction that Σ is diagonal. This approach, used by Hsiao and Wan (2014), is a generalization of the original method.

7. *Inverse Rank.* The inverse rank approach, suggested by Aiolfi and Timmermann (2006), ranks the forecast models based on their performance up to time N. The model with the lowest mean squared prediction error is assigned the rank 1, the model with the second lowest mean squared prediction error is assigned the rank 2, etc. The combined forecast is then calculated as follows:

$$f^c = \sum_{i=1}^{P} f_i' \times \frac{Rank_i^{-1}}{\sum_{j=1}^{P} Rank_j^{-1}} \tag{8}$$

Timmermann (2006) points out that this method, just like Bates and Granger (1969), also ignores correlations across forecast errors. However, the method is more robust to outliers, since total rankings are not likely to change dramatically by the presence of extreme forecasts.

Regression-based Combination Methods

8. *Ordinary Least Squares (OLS) regression.* The idea to use regression for combining forecasts was put forward by Crane and Crotty (1967) and successfully driven to the forefront by Granger and Ramanathan (1984). Using this approach, the combined forecast is a linear function of the individual forecasts where the weights are determined using a regression of the individual forecasts on the target itself:

$$y = \alpha + \sum_{i=1}^{P} w_i f_i + \varepsilon, \tag{9}$$

Using a portion of the forecasts to train the regression model, the OLS coefficients can be estimated by way of minimizing the sum of squared errors in equation(8). The combined forecast is then given by:

$$f^c = \widehat{\alpha} + \sum_{i=1}^{P} \widehat{w}_i f_i, \tag{10}$$

An advantage of the OLS forecast combinations is that the combined forecast is unbiased due to the intercept in the equation, even if one of the individual forecasts is biased. A disadvantage is that the method places no restriction on the combination weights (i.e. they do not add up to 1 and can be negative), which complicatesinterpretation, especially if the coefficients are non-convex.[3]

9. *Least Absolute Deviation (LAD) regression*. While the OLS regression estimates the coefficients in equation (10) by minimizing the sum of squared errors, we may want to estimate those coefficients differently, minimizing a different loss function[4], for example the absolute sum of squares. The reason is best explained using an example: Assume we have a model that performs well in general, yet every now and then misses the target by a very large margin. Such a model would be weighted more heavily under the LAD scheme than under the OLS scheme since those large but infrequent errors will be more heavily penalized using OLS. Whether this is beneficial depends on the user's preference and/or the cost of missing the target given the problem at hand. It should be noted that this lower sensitivity to outliers has another advantage: OLS weights can be unstable when predictors are highly correlated, which is the norm in forecast combination. Minor fluctuations in the sample can cause major shifts in the coefficient vector ('bouncing betas'), often leading to poor out-of-sample performance. This suggests that LAD combination should be favored in the presence of highly correlated component forecasts (Nowotarski et al., 2014).

10. *Constrained Least Squares (CLS) regression*. Like the LAD approach, CLS addresses the issue of 'bounding betas'. It does so by minimizing the sum of squared errors under some additional constraints. Specifically, we constrain the estimated coefficients $\{w_i\}$, allowing only for positive solutions: $w_i \geq 0 \; \forall i$, and to sum up to one: $\sum_{i=1}^{P} w_i = 1$. The solution requires numerical minimization, but good optimization algorithms are readily available: The **ForecastComb** package relies on the function `solve.QP` available from the **quadprog** package (Turlach and Weingessel, 2013). To tackle problems with high (but imperfect) collinearity that can cause errors in the CLS estimation, we also implement a revised Cholesky decomposition based on Ridge regression which has been propsed by Babaie-Kafaki and Roozbeh (2017) and can mitigate issues with multicollinearity.

    Theoretically, the additional constraints set CLS sub-optimally compared to the OLS. It lacks the good asymptotic properties admitted by OLS. However, in practice it is often found to perform better, especially so when the individual forecasts are highly correlated. In addition, the CLS weights are more easily interpretable. It is hard to justify a non-convex linear combination of two forecasts, while CLS weights can be conveniently interpreted for example as percentages devoted to each of the individual forecasts.

11. *Complete subset regression.* The **ForecastComb** package allows the relatively new idea of computing forecast combination weights using complete subset regression. The underlying idea is relatively straightforward: With P component forecasts that can serve as predictors in the regression model, we can form n regression models, each with a unique subset of predictors. n, the total number of combined forecasts from regression models is given by

$$n = \sum_{i=1}^{P} \binom{P}{i} = \sum_{i=1}^{P} \frac{P!}{i!(P-i)!} = 2^P - 1 \tag{11}$$

    In the most basic variant, the final combined forecast is obtained by taking the simple average over the cross-section of these n combined forecasts from complete subset regressions. The method is proposed by Elliott et al. (2013) who develop the theory behind this estimator and present favourable results from simulations and empirical application for US stock returns. Admittedly, the scheme is computationally expensive, and thus additional computational resources may be required if P is in the dozens (Elliott et al. (2013), Section 2.5.1 proposes a workaround based on random sampling from P). Additionally, since all n forecasts are returned,

[3]if the combination of two individual forecasts is not convex, the resulting combined forecast will not necessarily lie between those individual forecasts. As an example, we can look at the first quarter (2014 or 2015) GDPplus series estimate, published by the Federal Reserve Bank of Philadelphia. The combined prediction of two individual estimates of the US GDP; one based on the expenditure-side and one based on the income-side, lied above the sum of those two estimate. Intuitively, this is hard to communicate (`https://www.philadelphiafed.org/research-and-data/real-time-center/gdpplus`).

[4]For a discussion of optimal forecast combinations under general loss functions, see Elliott and Timmermann (2004)

the user can freely refine the technique further – for example by choosing not to average over all n forecasts, but some partial subset. Using the median instead of the mean is another option that comes to mind.

Obtaining n combined forecasts from the complete subset regression also allows us to use a frequentist approach to forecast combination, also known as information-theoretic forecast combinations. In the **ForecastComb** package, several information criteria are available in the complete subset regression method, each with its own merits and weaknesses: By far the most common are the AIC (Akaike's information criterion) and the BIC (Bayesian information criterion, also known as the Schwarz information criterion). Both are supplied in addition to the corrected AIC (Hurvich and Tsai, 1989) and the Hannan Quinn information criterion (Hannan and Quinn, 1979).

Formally, the weight given to each forecast based on the information-theoretic forecast combinations is the following:

$$w_i = \frac{\exp(-1/2\varrho_i)}{\sum_{j=1}^{n}\exp(-1/2\varrho_j)}, \tag{12}$$

where ϱ_i is the information criterion for forecast i obtained using a regression with a specific combination of forecasts. The value of n is fixed as the number of possible combinations, and the combined forecast is given by:

$$f^c = \sum_{i=1}^{n} w_i \tilde{f}_i. \tag{13}$$

It is worth noting that this is a two-step combination method. The first step is the computation of n combined forecasts $\tilde{f}_i$ using the complete subset regression method with the original P forecasts as predictors; the second step is the combination of these combined forecasts using the weights based on information criteria.

One advantage of this frequentist approach to model averaging is that the amount of shrinkage enforced on each individual forecast is data driven. The specification of a shrinkage hyperparameter, which is required in the corresponding Bayesian framework (e.g. Raftery and Painter, 2005) is spared from the user in this case.

Eigenvector-based Combination Methods

The eigenvector-based forecast combination methods, proposed by Hsiao and Wan (2014), are based on the idea of minimizing the mean squared prediction error subject to a normalization condition.

The most commonly used normalization condition for this purpose is to require the combination weights to add up to one, i.e. $\sum_{i=1}^{P} w_i = 1$ (e.g. Newbold and Granger, 1974; Timmermann, 2006). Hsiao and Wan (2014) show that this normalization condition leads to a constrained minimum of the mean squared prediction error (MSPE), and propose a normalization condition that leads to an unconstrained minimum of the MSPE:

$$\sum_{i=1}^{P} w_i^2 = 1 \tag{14}$$

This unconstrained minimum of the MSPE is the basis of the four eigenvector-based approaches in the **ForecastComb** package.

12. *Standard Eigenvector Approach.* The standard eigenvector approach retrieves combination weights from the estimated MSPE matrix as follows: The P positive eigenvalues of the MSPE matrix are arranged in increasing order $(\Phi_1 = \Phi_{min}, \Phi_2, ..., \Phi_P)$, and κ_i denotes the eigenvector corresponding to Φ_i. Let $d_i = e'\kappa_i$ with e being a $P \times 1$ vector of $(1, 1, ...1)'$. The combination weights w are then chosen corresponding to the minimum of $(\frac{\Phi_1}{d_1^2}, \frac{\Phi_2}{d_2^2}, ..., \frac{\Phi_P}{d_P^2})$, denoted as κ_l, as:

$$w = \frac{1}{d_l}\kappa_l \tag{15}$$

The combined forecast is then obtained as usual:

$$f^c = \sum_{i=1}^{P} f_i' w_i \tag{16}$$

13. *Bias-Corrected Eigenvector Approach.* The bias-corrected eigenvector approach builds on the idea that if one or more of the component models yield biased forecasts, the accuracy of the

standard eigenvector approach can be improved by eliminating the bias. It modifies the standard approach by decomposing forecast errors into three parts: model-specific bias, omitted common factors of all component models, as well as an idiosyncratic part that is uncorrelated across the component models.

The optimization procedure to obtain combination weights coincides with the standard approach, except that we use as an input the centered MSPE matrix, i.e. after extracting the bias by subtracting the column means of the MSPE:

$$w = \frac{1}{\tilde{d}_l}\tilde{\kappa}_l \tag{17}$$

where $\tilde{d}_i$ and $\tilde{\kappa}_i$ are defined analogously to d_i and κ_i in the standard eigenvector approach with the only difference that they correspond to the spectral decomposition of the centered MSPE matrix rather than the original (uncentered) MSPE matrix.

The combined forecast is then obtained by:

$$f^c = \alpha + \sum_{i=1}^{P} f_i' w_i \tag{18}$$

where the intercept α corrects for the potential bias.

14. *Trimmed Eigenvector Approach.*

 The standard approach is highly sensitive to the disparities in performance of different predictive models, i.e. the standard eigenvector approach's performance could be severely impaired by one or more component models that produce poor forecasts. This is due to treating uncertainties in the actual series, y, and the uncertainties of the component models, $F_{N\times P}$, symmetrically. For a detailed discussion of this so-called orthogonality principle, see Section 3 in Hsiao and Wan (2014). The trimmed eigenvector approach takes note of this issue.

 The idea of trimming the pool of input forecasts has been used by Aiolfi and Timmermann (2006) and is picked up by Hsiao and Wan (2014) using the eigenvector framework – the weights are computed exactly as in the standard eigenvector approach, but based on the MSPE matrix of the trimmed forecasts, after discarding particularly bad component models.

15. *Trimmed Bias-Corrected Eigenvector Approach.* The underlying methodology of the trimmed bias-corrected eigenvector approach is the same as the bias-corrected eigenvector approach: The weights are retrieved through the spectral decomposition of the centered MSPE matrix.

 The only difference to the bias-corrected eigenvector approach is that this method, like the trimmed eigenvector approach, pre-selects component models that serve as input for the forecast combination; only a subset of the available forecast models is retained, while the models with the worst performance are discarded, thereby combining the favourable modifications of the previous two methods.

There are three more related functions. The function `auto_combine` computes the fit for all the available forecast combination methods and is based on a grid-search optimization. It returns the combined forecast with the best in-sample accuracy. The default accuracy metric is the RMSE and MAE or MAPE are also available. The function `rolling_combine` is simply a dynamic variant, where the computation is done in an expanding window fashion rather than a single in- and out-of-sample split. Finally, the function `predict.foreccomb_res` allows to obtain the forecasts using previously trained forecast combination model. It is a simple utility function to quickly get the forecasts in an uncomplicated manner.

Implementation

The main functions provided in the **ForecastComb** package can be classified in 3 categories:

- Data Preparation
- Estimation of Forecast Combination
- Post-Fit Presentation of Results

In addition, some auxiliary functions are provided. Table 1 gives the list of the main functions.

Function	**Description**
Data Preparation Functions	
`foreccomb`	Transform raw input data for forecast combination
`cs_dispersion`	Compute cross-sectional dispersion
Forecast Combination Functions	
Simple Forecast Combination Functions	
`comb_BG`	Bates/Granger (1969) forecast combination
`comb_InvW`	Inverse Rank forecast combination
`comb_MED`	Median Forecast Combination
`comb_NG`	Newbold/Granger (1974) forecast combination
`comb_SA`	Simple Average forecast combination
`comb_TA`	Trimmed Mean forecast combination
`comb_WA`	Winsorized Mean forecast combination
Regression-Based Forecast Combination Functions	
`comb_CLS`	Constrained Least Squares (CLS) forecast combination
`comb_CSR`	Complete Subset Regression forecast combination
`comb_LAD`	Least Absolute Deviation (LAD) forecast combination
`comb_OLS`	Ordinary Least Squares (OLS) forecast combination
Eigenvector-Based Forecast Combination Functions	
`comb_EIG1`	Standard Eigenvector forecast combination
`comb_EIG2`	Bias-Corrected Eigenvector forecast combination
`comb_EIG3`	Trimmed Eigenvector forecast combination
`comb_EIG4`	Trimmed Bias-Corrected Eigenvector forecast combination
Other Forecast Combination Functions	
`auto_combine`	Automated grid-search forecast combination
`rolling_combine`	Rolling forecast combination (time-varying combination weights)
Post-Fit Functions	
`plot.foreccomb_res`	`S3` method to plot results from a forecast combination model
`summary.foreccomb_res`	`S3` method – summary of the forecast combination estimation
`predict.foreccomb_res`	`S3` method to create forecasts using weights from previously trained forecast combination model

Table 1: Brief description of main functions available in the **ForecastComb** package

Data Preparation

The **ForecastComb** package considers as a starting point that the user has already obtained a set of component forecasts, either from survey data or using statistical techniques, and now seeks to improve accuracy by combining those component forecasts into one. If the user only has the actual time series data, other packages in R can be used to create a set of component models. For instance, the **forecastHybrid** package by Shaub and Ellis (2017) which creates several univariate forecasts using methods available in the mature and popular **forecast** package (Hyndman, 2017).

The method `foreccomb` is the workhorse in the data preparation step. It supports the user with transforming the raw input data to make sure that the estimation of the forecast combinations will run smoothly.

The call of `foreccomb` is:

```
foreccomb(observed_vector, prediction_matrix, newobs = NULL,
  newpreds = NULL, byrow = FALSE, na.impute = TRUE, criterion = "RMSE")
```

The function requires user input for the parameters `observed_vector` (a vector, the actual data) and `prediction_matrix` (a matrix, the set of component forecasts to be combined). The format of the input matrix is as follows: Each column contains the forecasts from one of the P component models. Each row corresponds to the cross-section of component forecasts at a specific point in time. A situation where the format of the data is reversed, meaning that rows correspond to forecast models and columns correspond to the time index, is handled by setting the argument `byrow = TRUE`.

The `foreccomb` function includes some convenient features that take note of the fact that in many cases combination methods are applied to survey forecasts and the challenges that come along with this:

- *Split into Training Set and Test Set*. The function allows the user to specify a training set (`observed_vector` and `prediction_matrix`)and a test set (`newobs` and `newpreds`) separately. This is useful since most combination functions have to estimate the weights (requiring part of the sample to be dedicated to that task), while it is recommended that a test set is available to evaluate the model's performance on "new" data.
- *Missing Value Imputation*. Survey forecasts usually include missing values. This can be either because some of the survey participants did not respond or because the set of survey participants is changed. The `foreccomb` function provides two alternatives to deal with missing values: The default option (`na.impute = TRUE`) uses the missing value imputation algorithm from the **mtsdi** by Junger and de Leon (2012). It is a modified version of the EM algorithm for imputation that is specifically adjusted for multivariate time series data, accounting for correlation between the forecasting models and the time structure of the series itself. A smoothing spline is fitted to each of the time series at every iteration and the degrees of freedom of each spline are chosen by cross-validation. Alternatively, the argument can be set to `na.impute = FALSE`, which means the component forecast models that include any missing values are dropped prior to estimating forecast combinations, and the user is notified in the console if so relevant.
- *Handling Multicollinearity*. More often than not component forecasts that are used in the forecast combination are highly correlated. This can trouble the estimation process which does not handle well perfect collinearity. The `foreccomb` function has an inherent algorithm that checks the set of component forecasts for perfect multicollinearity, and if detected, drops one of the component models from the input data. The algorithm is designed to minimize the cost of dropping one or more models from the input data, in the sense that out of the models that cause perfect multicollinearity, it drops the least accurate forecast model. Only perfect multicollinearity is handled. By default, Root Mean Squared Error (RMSE) is used as the accuracy metric, but alternatively the user may choose the Mean Absolute Error (MAE) or the Mean Absolute Percentage Error (MAPE) by changing the argument `criterion`.

The output of the `foreccomb` function is an object of `S3` class `foreccomb` that can be passed on to the estimation functions or the other auxiliary functions, for instance the function `cs_dispersion` which computes the cross-sectional dispersion of the set of component forecasts.

This is often helpful for selecting a suitable combination method: One of the main findings of Hsiao and Wan (2014) is that regression-based methods produce more accurate forecasts when one or a few of the component forecasts are much better than the rest, while eigenvector-based methods perform better when there is low dispersion among the component forecasts. The `cs_dispersion` function can be used to compute and plot this cross-sectional dispersion using standard deviation (default), interquartile range, or range.

Estimation of Forecast Combination

The package provides the user with functions for the 15 estimation techniques for combined forecasts, which were described in previous Section. The estimation functions require an object of S3 class `foreccomb` as input, which is obtained using the methods from the previous subsection.

Four of the methods include trimming, i.e. a pre-selected subset of the full set of component models that should be used in the estimation of combination weights. These are:

- Trimmed Mean (`comb_TA`)
- Winsorized Mean (`comb_WA`)
- Trimmed Eigenvector Approach (`comb_EIG3`)
- Trimmed Bias-Corrected Eigenvector Approach (`comb_EIG4`)

For these methods, the user has some flexibility. The package provides the option to set the trimming factor (or, for the eigenvector methods, the number of retained component models) manually. Otherwise, an inbuilt optimization algorithm is used for choosing the trimming factor such that the combined forecast has the best possible fit. Again, this optimization can be based on either RMSE, MAE, or MAPE, which are controlled by the argument `criterion`.

A simple simulation example:

```
R> actual <- rnorm(100)
R> forecasts <- matrix(rnorm(1000, 1), 100, 10)

R> input_data <- foreccomb(actual, forecasts)

R> # Manual Selection of Trimming Factor:
R> model1 <- comb_TA(input_data, trim_factor = 0.3)

R> # Assess accuracy of the combined forecast:
R> model1$AccuracyTrain

                   ME     RMSE      MAE     MPE     MAPE       ACF1 Theil's U
Training Set -1.07312 1.520668 1.274116 146.411 486.3501 -0.0480562  1.698456

R> # Algorithm-Optimized Selection of Trimming Factor:
R> model2 <- comb_TA(input_data, criterion = "RMSE")

Optimization algorithm chooses trim factor for trimmed mean approach...
Algorithm finished. Optimized trim factor: 0.1

R> # Assess accuracy of the combined forecast:
R> model2$AccuracyTrain

                    ME     RMSE     MAE      MPE     MAPE        ACF1 Theil's U
Training Set -1.063363 1.515091 1.27304 134.7211 489.4353 -0.03678255  1.692617
```

As can be seen, the automated selection of the trimming factor leads to an improved accuracy of the combined forecast.

The 15 methods included in the package all produce static combination weights, i.e. the models use the training set data to estimate combination weights, which will in turn be applied to all periods of the test set. The research community in the forecasting field is strongly divided in the assessment of the value of time-varying combination weights, since putting higher weights on more recent data tends to increase the parameter variance. Section 4.1 in Timmermann (2006) reviews the advantages and challenges of allowing for time-varying weights.

While the **ForecastComb** mainly uses time-invariant combination weights, the user is provided with some flexibility. The `rolling_combine` function allows for the estimation of each of the methods with time-varying weights. The approach builds on the idea of time series cross-validation (Bergmeir et al., 2015), using the provided training set as a departure point to estimate starting weights, and then increasing the training set one step at a time and re-estimating the weights for the remaining test set. However, this approach requires that the user provides a full test set, i.e. also providing observed values for the test set.

The function `auto_combine` provides a quick and painless alternative. The function is based on a grid-search optimization that returns the combined forecast with the best in-sample accuracy (using

RMSE as accuracy metric in the default setting).

All of the estimation methods return an object of S3 class `foreccomb_res` with the components presented in Table 2. The object can subsequently be passed on to post-fit functions.

Return Values	Description
For all methods	
`Method`	Returns the used forecast combination method
`Models`	Returns the individual input models that were used for the forecast combinations
`Weights`	Returns the combination weights obtained by applying the combination method to the training set
`Fitted`	Returns the fitted values of the combination method for the training set
`Accuracy_Train`	Returns a range of accuracy measures for the training set
`Forecasts_Test`	Returns forecasts produced by the combination method for the test set. Only returned if input included a forecast matrix for the test set
`Accuracy_Test`	Returs a range of accuracy measures for the test set. Only returned if input included a forecast matrix and a vector of actual values for the test set
`Input_Data`	Returns the data forwarded to the method
Only for `comb_TA` *and* `comb_WA`	
`Trim Factor`	Returns the trim factor, λ
Only for `comb_OLS`, `comb_LAD`, `comb_EIG3` *and* `comb_EIG4`	
`Intercept`	Returns the intercept (bias correction)
Only for `comb_EIG3` *and* `comb_EIG4`	
`Top_Predictors`	Number of retained predictors
`Ranking`	Ranking of predictors that determines which models are removed in the trimming step

Table 2: Output Components of the Forecast Combination Estimation Metods

Post-Fit Functions

Results from the estimation of combined forecasts can be passed on to two post-fit convenience functions: `summary` and `plot`, which are S3 methods specific to the class `foreccomb_res`.

The summary function displays the output of the respective forecast combination in concise form, it displays the estimated combination weights (and the intercept, if the combination method includes one), as well as accuracy statistics for the training set and the test set.

The plot function will produce different plots based on the input data. If only a training set was provided, it plots the actual versus the fitted values; if a test set was also provided, it plots the combined forecasts as well. Another option for the user is a plot of the combination weights[5], obtained by setting `which = 2` in the function call.

[5]In case the combined forecast is produced using time-varying combination weights, the weights plot displays only the average weight of the respective component model over the test set period.

UK Electricity Supply: An Empirical Example

The **ForecastComb** package includes the dataset `electricity`, which is a multivariate time series of monthly UK electricity supply (in GWh) from January 2007 to March 2017, and 5 univariate time series forecasts for the same series and period. The observed data series is sourced from the International Energy Agency (IEA, 2017). The component models to be combined are the following cross-validated one-month univariate forecasts in the dataset:

- ARIMA (produced using the `auto.arima` function in the **forecast** package),
- ETS (produced using the `ets` function in the **forecast** package),
- Neural Network (produced using the `nnetar` function in the **forecast** package),
- Damped Trend (produced using the `ets` function in the **forecast** package),
- Dynamic Optimized Theta Model (produced using the `dotm` function in the **forecTheta** package by Fiorucci et al. (2016)).

To illustrate the functionalities of the package, we apply 4 combination techniques: the simple average (`comb_SA`), the OLS regression (`comb_OLS`), the standard eigenvector approach (`comb_EIG1`) and the trimmed bias-corrected eigenvector approach (`comb_EIG4`). The selected methods span all three categories of combination techniques (statistics-based, regression-based, and eigenvector-based) and includes trimmed and bias-corrected methods and are therefore suitable to show the full functionality of the **ForecastComb** package in this empirical context. using both the static and dynamic version of each. For the selected combination methods, we produce both static and dynamic forecasts, which gives us a total of 7 different time series of combined forecasts (not 8, since the static and dynamic versions of the simple average combination are identical).

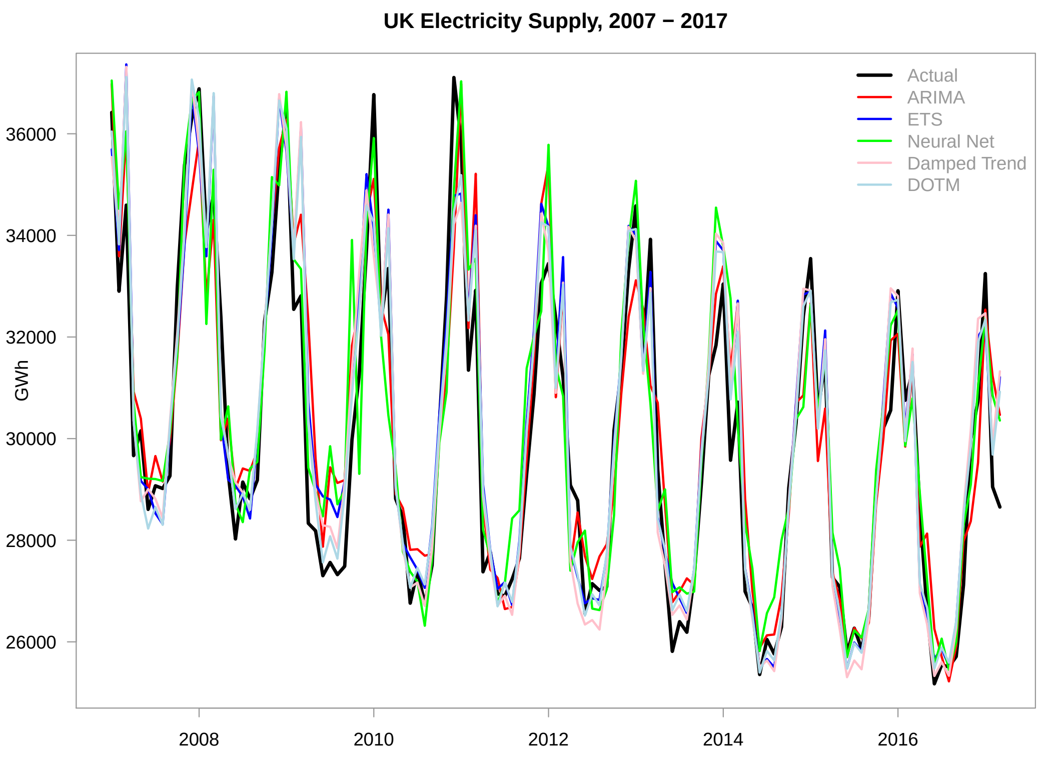


Figure 1: UK Electricity Supply, 2007 - 2017: Actual Value and Forecasts.

For the purpose of this exercise, we use the first 84 months as training set, which leaves us with a test set size of 39. Figure 1 plots the actual series and the univariate forecasts. The forecasts (which are 1-month forecasts obtained via time series cross-validation) track the actual series very well. None of them performs exceptionally poorly compared with the rest, which are conditions that tend to favour eigenvector approaches (Hsiao and Wan, 2014). As it is with most electricity data, the main difference between the individual models is in their ability to quickly recognize and adjust for turning points.

For example, the neural nets model handles turning points well, but sometimes also overshoots, while the ARIMA model has a smoother behaviour around turning points.

First, we format the data correctly for the estimation of combination weights. This step ensures later operations would proceed smoothly. Since there are no missing values and no perfectly collinear columns in our dataset, this is relatively straightforward:

```
R> data(electricity)
R> train.obs <- electricity[1:84, 6]
R> train.pred <- electricity[1:84, 1:5]
R> test.obs <- electricity[85:123, 6]
R> test.pred <- electricity[85:123, 1:5]
R> input_data <- foreccomb(train.obs, train.pred, test.obs, test.pred)
```

Once the object of S3 class `foreccomb` is created, it can be fed into the estimation functions. We can look at the cross-sectional dispersion, to get a better idea of variability in the univariate forecasts.

```
R> cs_dispersion(input_data, measure = "SD", plot = TRUE)
```

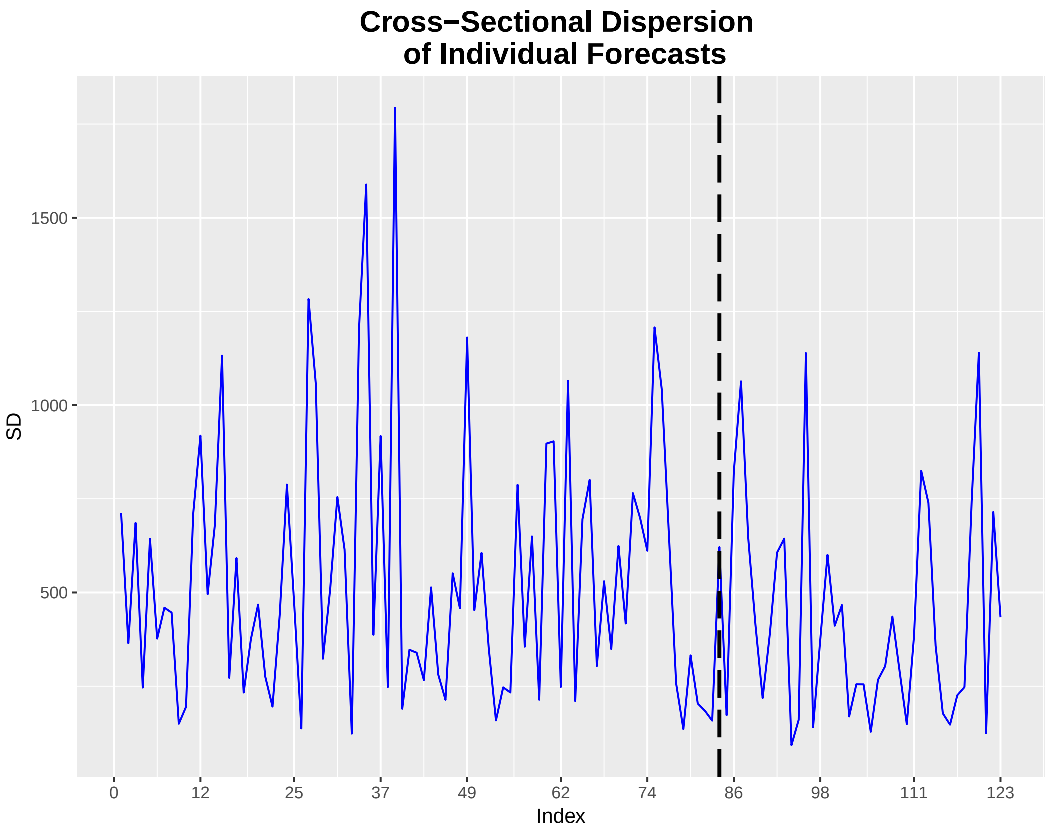


Figure 2: Cross-Sectional Standard Deviation of the Component Forecasts.

Figure 2 shows that apart from a brief period of increased dispersion around the end of 2009, the cross-sectional standard deviation of the component forecasts is rather stable and low given the level of around 25,000 to 35,000 GWh. This begs the question whether conditions have fluctuated enough during the test set so that estimation of time-varying weights is beneficial, yet we proceed with it for this demonstration.

```
R> ####### ESTIMATION OF STATIC FORECAST COMBINATIONS ########
```

```
R> SA <- comb_SA(input_data)
R> OLS_static <- comb_OLS(input_data)
R> EIG1_static <- comb_EIG1(input_data)
R> EIG4_static <- comb_EIG4(input_data, criterion = "MAE")

R> ###### ESTIMATION OF DYNAMIC FORECAST COMBINATIONS
R> OLS_dyn <- rolling_combine(input_data, "comb_OLS")
R> EIG1_dyn <- rolling_combine(input_data, "comb_EIG1")
R> EIG4_dyn <- rolling_combine(input_data, "comb_EIG4", criterion = "MAE")
```

The 7 combined forecasts can be evaluated separately by looking at their summary measures, which we present here for the static Ordinary Least Squares approach:

```
R> summary(OLS_static)

Summary of Forecast Combination
-------------------------------
Method:  Ordinary Least Squares Regression
Individual Forecasts & Combination Weights:
        Combination Weight
arima           0.02152869
ets            -0.20646266
nnet            0.20992792
dampedt        -1.04349858
dotm            1.97991049

Intercept (Bias-Correction):  962.3229

Accuracy of Combined Forecast:

                        ME     RMSE      MAE         MPE     MAPE
Training Set -4.417560e-12 888.1433 697.8645 -0.08262472 2.254470
Test Set     -4.007742e+01 671.5214 536.0331 -0.24705122 1.841961

Additional information can be extracted from the combination object:
For fitted values (training set):  OLS_static$Fitted
For forecasts (test set):  OLS_static$Forecasts_Test
See  str(OLS_static)  for full list.
```

The output shows that the OLS combination puts an extremely high relative weight on the forecast from the Dynamic Optimized Theta model, which seems to be the best component forecast, which is rather surprising given that seasonality is an important feature in the analyzed series and Theta models cannot incorporate seasonality into the estimation so far, relying on pre-estimation deseasonalizing and post-estimation reseasonalizing. Table 3 shows a comparison of the accuracies achieved by the combined forecasts. Since all forecasts are for the same series, it is reasonable to use MAE as accuracy metric.

Method	MAE Training Set	MAE Test Set
Simple Average	819.28	573.39
Ordinary Least Squares (static)	697.86	536.03
Ordinary Least Squares (dynamic)	697.86	533.47
Standard Eigenvector (static)	821.60	573.84
Standard Eigenvector (dynamic)	821.60	572.99
Trimmed Bias-Corrected Eigenvector (static)	785.30	540.18
Trimmed Bias-Corrected Eigenvector (dynamic)	785.30	541.98

Table 3: Mean Absolute Errors of Combined Forecasts

The evaluation of accuracy delivers some interesting insight: All of the combination models perform better in the test set than in the training set, which is counter-intuitive, but is likely due to the increased dispersion among the component forecasts in the early period. The results also clearly suggest that one or more of the component forecasts were biased. The two methods with an intercept (i.e. bias correction) perform best. Finally, allowing for time-varying combination weights does not seem to change test-set accuracy much compared with the models' static counterparts, suggesting that the estimated combination weights did not fluctuate a lot over time. There are some potential explanations why the OLS method performed extremely well in this case:

- With such stable conditions the risk of 'bouncing betas' (described in the previous section) is low,
- The OLS method produces unbiased forecasts even if one or more of the component forecasts are biased (which is why the trimmed bias-corrected eigenvector approach performed reasonably well too),
- One of the component forecasts is much better than the rest, a situation that is favorable for regression-based approaches, as pointed out by Hsiao and Wan (2014). These are also conditions under which sophisticated methods actually can largely improve upon a simple average combination.

Now that we learned that some of the combination models produced more accurate forecasts than the simple average, we address the next, very natural question: How well did the combination methods perform compared to the univariate component forecasts themselves? To shed more light on this question, Table 4 shows the MAE values for the univariate models during the test set period.

Method	**MAE Test Set**
ARIMA	770.32
ETS	615.88
Neural Network	730.35
Damped Trend	660.08
DOTM	540.24

Table 4: Mean Absolute Errors of Univariate Component Forecasts

The results of the accuracy evaluation speak for themselves. Not only did two of the combined forecasts perform better or equally well as the best univariate forecast over the test set period, but also the forecast risk is considerably lower indeed. In the test set the range of MAEs of the different combined forecast methods is only 40, while the corresponding value for the univariate forecasting methods is 230. It is worth noting that all of the combined forecasts perform considerably better than even the second-best univariate forecast, emphasizing the appeal of forecast combination: In the ideal case, it is possible to end up with a forecast that is better than the best univariate forecast. Even if this is not the case, using forecast combination considerably decreases the risk of ending up with a poorly performing model.

Finally, we take a closer look at the results from the best combined forecast, which is the dynamic OLS combination method in this exercise.

```
R> ##### ACTUAL VS FITTED PLOT #####
R> plot(OLS_dyn)
```

```
R> ##### COMBINATION WEIGHTS #####
R> OLS_static$Weights
 fhatarima     fhatets    fhatnnet fhatdampedt    fhatdotm
 0.02152869 -0.20646266  0.20992792 -1.04349858  1.97991049
R> colMeans(OLS_dyn$Weights)

  fhatarima      fhatets     fhatnnet  fhatdampedt     fhatdotm
 0.002750948 -0.123811218  0.184846358 -1.098381593  2.001988985
```

Figures 3 shows how well the combined forecast obtained from the dynamic OLS method predicts the monthly electricity supply series. Results confirm the conjecture that the stable conditions (low cross-sectional dispersion and one very dominant univariate component forecast) do not cause a lot of fluctuation in the combination weights even when allowing for time-varying weights.

The weights presented also confirms another thing: that weights obtained using the OLS combination methods can be hard to interpret. It seems obvious that the method should put a high weight on the DOTM forecast, since it is the best univariate forecast by far. However, the reason why it assigns

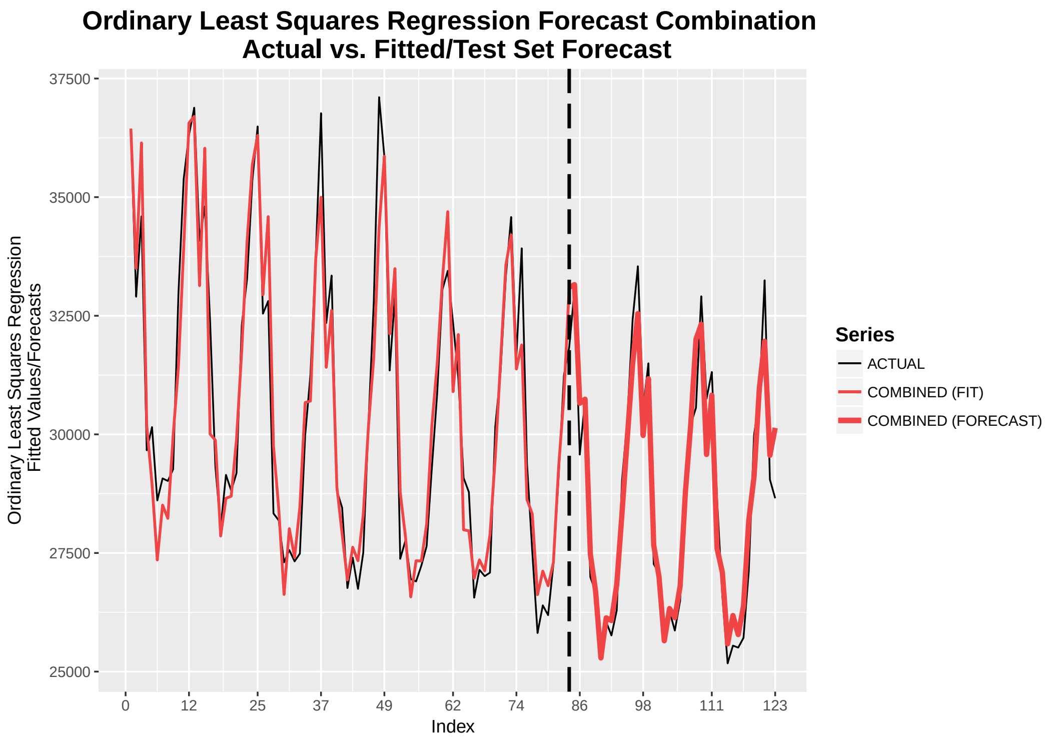


Figure 3: Static OLS Forecast Combination: Actual vs Fitted Plot.

negative weights to the ETS and Damped Trend forecasts (the second- and third-best univariate forecasts) is not very intuitive. A possible explanation might be that all three of these are exponential smoothing-type models, suggesting that the information obtained from the ETS and Damped Trend models is better captured by the DOTM model, while ARIMA and Neural Networks are not closely related modelling approaches to the Theta model, so that even though these models perform far worse on average, they capture information differently and might outperform the DOTM forecast in some periods for that reason, justifying their positive weights (however small) and explaining how the combined forecast can slightly outperform the best component forecast.

Discussion and Conclusions

Forecast combination is a useful strategy to hedge against model risk. Even if combined forecasts do not win over the most accurate component forecast, they generally avoid poor performance by circumventing the choice between individual methods (Timmermann, 2006). Instead of putting all eggs into one basket using model selection, these model averaging techniques are motivated by portfolio theory and diversify across component forecasts.

The **ForecastComb** package categorizes some of the most popular approaches into 3 groups: (a) simple statistics-based methods, (b) regression-based methods, and (c) eigenvector-based methods. Providing both regression-based combinations and eigenvector-based combinations to the users is considered useful, since the former tend to perform relatively better when one or a few component forecasts are much better than the rest, while the latter perform relatively better when all forecasts are in the same ballpark (Hsiao and Wan, 2014).

The package is designed to support users along the entire modelling process: data preparation, model estimation, and interpretation of results using summaries and plotting functionalities. It includes tools for data transformation that are designed to deal with two common issues in forecast combination prior to estimation – missing values and multicollinearity. The 15 combination methods are available in both static and dynamic variants, and users have the option to automate the selection algorithm so that a good combination method is found based on training set fit. Finally, the package offers specialized functions for summarizing and visualizing the combination results.

While the current version of the package already provides a comprehensive set of tools for forecast combination, there is scope for further extensions in future updates. First, additional combination methods that showed promising results recently can be added, for instance the factor-augmented regression approach by Cheng and Hansen (2015) and the AdaBoost algorithms reviewed by Barrow and Crone (2016). Second, additional algorithms for adaptive combination weights (cf. Timmermann, 2006) can be implemented to provide even more flexibility with dynamic estimation. There are few other possible extensions which are not handled here. For example the adaptation for a forecast combination context of the mean absolute scaled error (Hyndman and Koehler, 2006) – the current gold standard for accuracy evaluation – using the in-sample MAE of the best component forecast as scaling factor. Another extension is of course the combination of forecast densities, while in this paper the scope was limited to point-forecasts only.

Acknowledgements

We thank four anonymous referees for their insightful comments and remarks which helped to improve both the package and this paper.

Bibliography

M. Aiolfi and A. Timmermann. Persistence in forecasting performance and conditional combination strategies. *Journal of Econometrics*, 135(1):31–53, 2006. URL https://doi.org/10.1016/j.jeconom.2005.07.015. [p265, 268]

R. R. Andrawis, A. F. Atiya, and H. El-Shishiny. Combination of long term and short term forecasts, with application to tourism demand forecasting. *International Journal of Forecasting*, 27(3):870–886, 2011. URL https://doi.org/10.1016/j.ijforecast.2010.05.019. [p263]

J. S. Armstrong. Combining forecasts: The end of the beginning or the beginning of the end? *International Journal of Forecasting*, 5(4):585–588, 1989. URL https://doi.org/10.1016/0024-6301(90)90317-w. [p264]

J. S. Armstrong. Combining forecasts. In J. S. Armstrong, editor, *Principles of Forecasting: A Handbook for Researchers and Practitioners*, pages 417–439. Kluwer Academic Publisher, 2001. URL https://doi.org/10.1016/s0040-1625(02)00180-4. [p264]

D. Avramov. Stock return predictability and model uncertainty. *Journal of Financial Economics*, 64(3):423–458, 2002. URL https://doi.org/10.2139/ssrn.260591. [p263]

S. Babaie-Kafaki and M. Roozbeh. A revised cholesky decomposition to combat multicollinearity in multiple regression models. *Journal of Statistical Computation and Simulation*, 87(12):2298–2308, 2017. URL https://doi.org/10.1080/00949655.2017.1328599. [p266]

D. K. Barrow and S. F. Crone. A comparison of adaboost algorithms for time series forecast combination. *International Journal of Forecasting*, 32(4):1103–1119, 2016. URL https://doi.org/10.1016/j.ijforecast.2016.01.006. [p278]

J. M. Bates and C. W. J. Granger. The combination of forecasts. *Operations Research Quarterly*, 20:451–468, 1969. URL https://doi.org/10.1016/b978-0-12-295180-0.50027-6. [p263, 265]

C. Bergmeir, R. J. Hyndman, and B. Koo. A note on the validity of cross-validation for evaluating time series prediction. Working Paper No. 10/15, Monash University, Department of Econometrics and Business Statistics, 2015. URL https://doi.org/10.1016/j.csda.2017.11.003. [p271]

G. Cheng and Y. Yang. Forecast combination with outlier protection. *International Journal of Forecasting*, 31(2):223–237, 2015. URL https://doi.org/10.1016/j.ijforecast.2014.06.004. [p263]

X. Cheng and B. E. Hansen. Forecasting with factor-augmented regression: A frequentist model averaging approach. *Journal of Econometrics*, 186(2):280–293, 2015. URL https://doi.org/10.2139/ssrn.2343103. [p278]

C. Christiansen, M. Schmeling, and A. Schrimpf. A comprehensive look at financial volatility prediction by economic variables. *Journal of Applied Econometrics*, 27(6):956–977, 2012. URL https://doi.org/10.1002/jae.2298. [p263]

G. Claeskens, J. R. Magnus, A. L. Vasnev, and W. Wang. The forecast combination puzzle: A simple theoretical explanation. *International Journal of Forecasting*, 32(3):754–762, 2016. URL https://doi.org/10.1016/j.ijforecast.2015.12.005. [p264]

R. T. Clemen. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5(4):559–583, 1989. URL https://doi.org/10.1016/0169-2070(89)90012-5. [p263, 264]

D. B. Crane and J. R. Crotty. A two-stage forecasting model: Exponential smoothing and multiple regression. *Management Science*, 13(8):501–507, 1967. URL https://doi.org/10.1287/mnsc.13.8.b501. [p263, 265]

G. Elliott and A. Timmermann. Optimal forecast combinations under general loss functions and forecast error distributions. *Journal of Econometrics*, 122(1):47–79, 2004. URL https://doi.org/10.1016/j.jeconom.2003.10.019. [p266]

G. Elliott and A. Timmermann. Optimal forecast combinations under regime switching. *International Economic Review*, 46(4):1081–1102, 2005. URL https://doi.org/10.1111/j.1468-2354.2005.00361.x. [p263]

G. Elliott, A. Gargano, and A. Timmermann. Complete subset regressions. *Journal of Econometrics*, 2013. URL https://doi.org/10.1016/j.jeconom.2013.04.017. [p263, 266]

J. A. Fiorucci, F. Louzada, and B. Yiqi. *forecTheta: Forecasting Time Series by Theta Models*, 2016. URL https://CRAN.R-project.org/package=forecTheta. R package version 2.2. [p273]

P. Gaillard and Y. Goude. *Opera: Online Prediction by Expert Aggregation*, 2016. URL https://CRAN.R-project.org/package=opera. R package version 1.0. [p263]

V. Genre, G. Kenny, A. Meyler, and A. Timmermann. Combining expert forecasts: Can anything beat the simple average? *International Journal of Forecasting*, 29(1):108–121, 2013. URL https://doi.org/10.1016/j.ijforecast.2012.06.004. [p264]

A. Graefe, J. S. Armstrong, R. J. Jones, and A. G. Cuzán. Combining forecasts: An application to elections. *International Journal of Forecasting*, 30(1):43–55, 2014. URL https://doi.org/10.1016/j.ijforecast.2013.02.005. [p262, 264]

C. W. Granger and R. Ramanathan. Improved methods of combining forecasts. *Journal of Forecasting*, 3 (2):197–204, 1984. URL https://doi.org/10.1002/for.3980030207. [p265]

E. J. Hannan and B. G. Quinn. The determination of the order of an autoregression. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 190–195, 1979. URL http://www.jstor.org/stable/2985032. [p267]

B. E. Hansen. Challenges for econometric model selection. *Econometric Theory*, 21(1):60–68, 2005. URL https://doi.org/10.1017/s0266466605050048. [p262]

B. E. Hansen. Least squares model averaging. *Econometrica*, pages 1175–1189, 2007. URL https://doi.org/10.1111/j.1468-0262.2007.00785.x. [p263]

B. E. Hansen. Least-squares forecast averaging. *Journal of Econometrics*, 146(2):342–350, 2008. URL https://doi.org/10.1016/j.jeconom.2008.08.022. [p263]

B. E. Hansen and J. S. Racine. Jackknife model averaging. *Journal of Econometrics*, 167(1):38–46, 2012. URL https://doi.org/10.1016/j.jeconom.2011.06.019. [p263]

D. F. Hendry and M. P. Clements. Pooling of forecasts. *The Econometrics Journal*, 7(1):1–31, 2004. URL ttps://doi.org/10.1111/j.1368-423x.2004.00119.x. [p264]

C. Hsiao and S. K. Wan. Is there an optimal forecast combination? *Journal of Econometrics*, 178(2): 294–309, 2014. URL https://doi.org/10.1016/j.jeconom.2013.11.003. [p263, 265, 267, 268, 270, 273, 276, 277]

C. M. Hurvich and C.-L. Tsai. Regression and time series model selection in small samples. *Biometrika*, 76(2):297–307, 1989. URL https://doi.org/10.2307/2336663. [p267]

R. J. Hyndman. *forecast: Forecasting Functions for Time Series and Linear Models*, 2017. URL http://github.com/robjhyndman/forecast. R package version 8.1. [p270]

R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. URL https://doi.org/10.1016/j.ijforecast.2006.03.001. [p278]

IEA. *Monthly Electricity Statistics*, 2017. International Energy Agency. [p273]

V. R. R. Jose and R. L. Winkler. Simple robust averages of forecasts: Some empirical results. *International Journal of Forecasting*, 24(1):163–169, 2008. URL https://doi.org/10.1016/j.ijforecast.2007.06.001. [p264, 265]

W. Junger and A. P. de Leon. *Mtsdi: Multivariate Time Series Data Imputation*, 2012. URL https://CRAN.R-project.org/package=mtsdi. R package version 0.3.3. [p270]

G. Kapetanios, V. Labhard, and S. Price. Forecasting using bayesian and information-theoretic model averaging: An application to uk inflation. *Journal of Business & Economic Statistics*, 26(1):33–41, 2008. URL https://doi.org/10.1198/073500107000000232. [p263]

J. R. Magnus and W. Wang. Concept-based bayesian model averaging and growth empirics. *Oxford Bulletin of Economics and Statistics*, 76(6):874–897, 2014. URL https://doi.org/10.1111/obes.12068. [p263]

S. K. McNees. The uses and abuses of 'consensus' forecasts. *Journal of Forecasting*, 11(8):703–710, 1992. URL https://doi.org/10.1016/0169-2070(93)90039-p. [p264]

C. Morana. Model averaging by stacking. Working Paper 310, University of Milan Bicocca, Department of Economics, Management and Statistics, 2015. URL https://doi.org/10.2139/ssrn.2665704. [p263]

P. Newbold and C. W. J. Granger. Experience with forecasting univariate time series and the combination of forecasts. *Journal of the Royal Statistical Society A*, 137(2):131–165, 1974. URL https://doi.org/10.2307/2344546. [p265, 267]

J. Nowotarski, E. Raviv, S. Trück, and R. Weron. An empirical comparison of alternative schemes for combining electricity spot price forecasts. *Energy Economics*, 46:395–412, 2014. URL https://doi.org/10.2139/ssrn.2313553. [p263, 266]

A. Opschoor, D. J. Van Dijk, and M. Van der Wel. Improving density forecasts and value-at-risk estimates by combining densities. Discussion Paper 14-090/III, Tinbergen Institute, 2014. URL https://doi.org/10.2139/ssrn.2469024. [p263]

F. C. Palm and A. Zellner. To combine or not to combine? issues of combining forecasts. *Journal of Forecasting*, 11(8):687–701, 1992. URL https://doi.org/10.1002/for.3980110806. [p264]

A. E. Raftery and I. S. Painter. Bma: An r package for bayesian model averaging. *R news*, 5(2):2–8, 2005. [p263, 267]

D. E. Rapach, J. K. Strauss, and G. Zhou. Out-of-sample equity premium prediction: Combination forecasts and links to the real economy. *Review of Financial Studies*, 23(2):821–862, 2010. [p263]

F. Ravazzolo, M. Verbeek, and H. K. Van Dijk. Predictive gains from forecast combinations using time varying model weights. Report 2007-26, Econometric Institute, 2007. URL https://doi.org/10.2139/ssrn.1012574. [p263]

E. Raviv. *ForecastCombinations: Forecast Combinations in R*, 2015. URL https://CRAN.R-project.org/package=ForecastCombinations. R package version 1.1. [p263]

E. Raviv, K. E. Bouwman, and D. van Dijk. Forecasting day-ahead electricity prices: Utilizing hourly prices. *Energy Economics*, 50:227 – 239, 2015. ISSN 0140-9883. URL https://doi.org/10.1016/j.eneco.2015.05.014. [p263]

D. Shaub and P. Ellis. *forecastHybrid: Convenient Functions for Ensemble Time Series Forecasts*, 2017. URL https://CRAN.R-project.org/package=forecastHybrid. R package version 0.4.1. [p263, 270]

J. Smith and K. F. Wallis. A simple explanation of the forecast combination puzzle. *Oxford Bulletin of Economics and Statistics*, 71(3):331–355, 2009. URL https://doi.org/10.1111/j.1468-0084.2008.00541.x. [p264]

J. H. Stock and M. W. Watson. Combination forecasts of output growth in a seven-country data set. *Journal of Forecasting*, 23(6):405–430, 2004. URL https://doi.org/10.1002/for.928. [p263, 264]

A. Timmermann. Forecast combinations. *Handbook of economic forecasting*, 1:135–196, 2006. URL https://doi.org/10.1016/s1574-0706(05)01004-9. [p263, 264, 265, 267, 271, 277, 278]

B. A. Turlach and A. Weingessel. *Quadprog: Functions to Solve Quadratic Programming Problems.*, 2013. URL https://CRAN.R-project.org/package=quadprog. R package version 1.5-5. [p266]

C. E. Weiss. Hierarchical time series and forecast combination in healthcare demand forecasting. Working Paper, Judge Business School, University of Cambridge, 2017. [p263]

C. E. Weiss and G. R. Roetzer. *GeomComb: (Geometric) Forecast Combination Methods*, 2016. URL https://CRAN.R-project.org/package=GeomComb. R package version 1.0. [p263]

J. H. Wright. Bayesian model averaging and exchange rate forecasts. *Journal of Econometrics*, 146(2): 329–341, 2008. URL https://doi.org/10.2139/ssrn.457345. [p263]

J. H. Wright. Forecasting us inflation by bayesian model averaging. *Journal of Forecasting*, 28(2):131–144, 2009. URL https://doi.org/10.1002/for.1088. [p263]

Christoph E. Weiss
Judge Business School
University of Cambridge
Cambridge CB2 1AG
United Kingdom
christoph.weiss@cantab.net

Eran Raviv
APG - Asset Management
Gustav Mahlerplein 3, 1082 Amsterdam
The Netherlands
eran.raviv@apg-am.nl

Gernot Roetzer
Trinity College Dublin
College Green, Dublin 2
Ireland
roetzeg@tcd.ie

testforDEP: An R Package for Modern Distribution-free Tests and Visualization Tools for Independence

by Jeffrey C. Miecznikowski, En-shuo Hsu, Yanhua Chen, Albert Vexler

Abstract This article introduces **testforDEP**, a portmanteau R package implementing for the first time several modern tests and visualization tools for independence between two variables. While classical tests for independence are in the base R packages, there have been several recently developed tests for independence that are not available in R. This new package combines the classical tests including Pearson's product moment correlation coefficient method, Kendall's τ rank correlation coefficient method and Spearman's ρ rank correlation coefficient method with modern tests consisting of an empirical likelihood based test, a density-based empirical likelihood ratio test, Kallenberg data-driven test, maximal information coefficient test, Hoeffding's independence test and the continuous analysis of variance test. For two input vectors of observations, the function `testforDEP` provides a common interface for each of the tests and returns test statistics, corresponding p values and bootstrap confidence intervals as output. The function `AUK` provides an interface to visualize Kendall plots and computes the area under the Kendall plot similar to computing the area under a receiver operating characteristic (ROC) curve.

Introduction

In this article, we present the **testforDEP** R package, a package for testing dependence between variables. Since in nonparametric settings there are no most powerful tests for independence, it is important to be able to implement various reasonable independence tests tuned for different classes of alternatives. This package addresses a need for implementing *both* classical and novel tests of independence in a single easy to use function. The function `cor.test` offered in the base R package gives classical tests for association/correlation between two samples, using the Pearson product moment correlation coefficient (Pearson, 1920), Kendall τ rank correlation coefficient (Kendall, 1938) and Spearman ρ rank correlation coefficient (Spearman, 1904). The function `cor.test` is helpful to test for independence between two samples when the samples are linearly dependent or monotonically associated. However the function `cor.test` is less powerful to detect general structures of dependence between two random variables, including non-linear and/or random-effect dependence structures. Many modern statistical methodologies have been proposed to detect these general structures of dependence. These methods include an empirical likelihood based test (Einmahl and McKeague, 2003), a density-based empirical likelihood ratio test for independence (Vexler et al., 2014), data-driven rank test for independence (Kallenberg and Ledwina, 1999), maximal information coefficient (Reshef et al., 2011) and a continuous analysis of variance test (Wang et al., 2015). These methods are useful to detect complex structures of dependence and until now there were no R packages available that implement these modern tests.

We propose the new package **testforDEP**, combining both classical and modern tests. The package **testforDEP** also provides dependence visualization tools such as the Kendall plot with the area under Kendall plot (Vexler et al., 2017) which previously has not been available in R packages. Moreover, we develop an exact test based on the maximal information coefficient to detect dependence between two random variable and we perform a power analysis for these different tests. The **testforDEP** package is available from the Comprehensive R Archive Network and also available for download at the author's department webpage.

We will focus on tests of bivariate independence of two random variables X and Y where we have n sample measurements or observations $(X_1, X_2, \dots, X_n)$ and $(Y_1, Y_2, \dots, Y_n)$, respectively. If X and Y have cumulative distribution functions $F_X(x)$ and $F_Y(y)$ and probability density functions $f_X(x)$ and $f_Y(y)$ we say X and Y are independent if and only if the combined random variable (X, Y) has a joint cumulative distribution function $F(x, y) = F_X(x)F_Y(y)$ or equivalently, if the joint density exists, $f(x, y) = f_X(x)f_Y(y)$ for all $x, y \in \mathbb{R}$. We are interested in testing the null hypothesis:

$$H_0 : X \text{ and } Y \text{ are independent,} \tag{1}$$

which is equivalent to

$$H_0 : F(x, y) = F_X(x)F_Y(y) \text{ for all } x, y \in \mathbb{R}. \tag{2}$$

We note there are no most powerful testing strategies in this setting. Thus a strategy for choosing a test depends on the type of correlation that a user is trying to detect. The following sections detail the different tests for independence, outline the components of the **testforDEP** package and provides real data examples demonstrating the power of the modern methods to detect nonlinear dependence. Finally, we conclude with a brief summary and future directions.

Classical tests of independence

In the following subsections we outline the classical tests of independence.

Pearson product moment correlation coefficient

The Pearson product-moment correlation coefficient γ is a well known measure of linear correlation (or dependence) between two random variables with a full presentation in Pearson (1920) and Hauke and Kossowski (2011). Using the standard Pearson estimator (denoted $\hat{\gamma}$ we have that $T_{\hat{\gamma}}$ asymptotically follows a t distribution with $n-2$ degrees of freedom under independence. Accordingly, a size α rejection rule is:

$$|T_{\hat{\gamma}}| > t_{1-\alpha/2}, \tag{3}$$

where $t_{1-\alpha/2}$, is the $1-\alpha/2$ quantile for the t distribution with $n-2$ degree of freedom. Advantages of the Pearson test are that it is simple to execute, has high power to detect linear dependency and also when the underlying data are normally distributed in which it also has the form of a maximum likelihood estimator. A weakness of the Pearson test is that it has very weak power to detect dependency that is not characterized by first moments and is often not powerful in many nonparametric settings. For example, if X_i is a normally distributed random variable and $Y_i = 1/X_i$ then the Pearson test will have very lower power to detect this structure (see, Miecznikowski et al. (2017)).

Kendall rank correlation coefficient

The Kendall rank correlation coefficient τ is a well known method proposed in Kendall (1938) as a nonparametric measure of monotonic association between two variables. Using the standard estimator $\hat{\tau}$, the test statistic Z_τ is given by:

$$Z_\tau = \frac{3\tau\sqrt{n(n-1)}}{\sqrt{2(2n+5)}}, \tag{4}$$

where Z_τ approximately follows a standard normal distribution under independence. A level α rejection rule for the null hypothesis is as follows:

$$|Z_\tau| > z_{1-\alpha/2} \tag{5}$$

where $z_{1-\alpha/2}$ is $1-\alpha/2$ quantile for a standard normal distribution. A weakness of the Kendall rank correlation coefficient test is that it has low power to detect non-monotonic dependency structures.

Spearman rank correlation coefficient

Spearman's rank correlation coefficient ρ proposed by Spearman (1904) is a nonparametric measure of statistical dependence between two variables. Spearman's rank correlation measures the monotonic association between two variables (Spearman, 1904; Hauke and Kossowski, 2011). The statistic ρ is defined as:

$$\rho = 1 - \frac{6\sum_{i=1}^{n} d_i^{\,2}}{n(n^2-1)}, \quad d_i = R_i - S_i, \tag{6}$$

where R_i, S_i are the ranks of X_i and Y_i, respectively. Note $\rho = 0$ suggests X and Y are independent. To conduct a test for statistical independence under the null hypothesis, we use the test statistic T_ρ defined as:

$$T_\rho = \rho\sqrt{\frac{n-2}{1-\rho^2}}, \tag{7}$$

which is distributed approximately as a Student's t distribution with $n-2$ degrees of freedom under null hypothesis. Accordingly, a level α rejection rule is:

$$|T_\rho| > t_{1-\alpha/2}. \tag{8}$$

Similar to the Kendall rank correlation coefficient, a weakness of the Spearman rank correlation coefficient test is that it has low power to detect non-monotonic dependency structures.

Hoeffding's test for independence

Hoeffding's test for dependence is proposed in Hoeffding (1948) as a test for two random variables with continuous distribution functions. Hoeffding's D is a nonparametric measure of the distance between joint distribution, $F(x,y)$ and product of marginal distributions, $F_X(x)F_Y(y)$ (Harrell Jr and Dupont, 2006). The coefficient D is defined as:

$$D(x,y) = F(x,y) - F_X(x)F_Y(y). \tag{9}$$

We can also define Δ as:

$$\Delta = \Delta(F) = \int D^2(x,y)dF(x,y), \tag{10}$$

where F is the joint distribution of X and Y and the integral is over $\mathbb{R}^2$.

We denote Ω' as the class of X, Y's such that their joint density function $F(x,y)$ is continuous. Denote Ω'' as the class of X, Y's such that their joint and marginal probability density functions are continuous. It has been shown that if $F(x,y)$ belongs to Ω'', $\Delta(F) = 0 \iff D(x,y) = 0$. The random variables X and Y are independent if and only if $D(x,y) = 0$.

It can be shown that D ranges between $-\frac{1}{60}$ and $\frac{1}{30}$ where larger values of D suggest dependence. An estimator of D, $\hat{D}$ is defined as:

$$\hat{D}(x,y) = \hat{F}(x,y) - \hat{F}_X(x)\hat{F}_Y(y). \tag{11}$$

We use $\hat{D}$ as the test statistic and note that it only depends on the rank order of observations (Hoeffding, 1948). For computing the test statistic $\hat{D}$, we use the R package **Hmisc** (Harrell Jr et al., 2017). Note the returned test statistic $\hat{D}'$ is 30 times the original $\hat{D}$ in Hoeffding (1948). A size α test can be given by:

$$\hat{D}' > C_\alpha, \tag{12}$$

where C_α is a size α critical value.

Due to the limiting computing tools available for the original publication in 1948, the author only provided cutoff tables for small sample sizes. With advanced computing power, we compute the C_α cutoffs for $n = 10, 20, 50, 100, 200, 500$ in Table 1.

	$n=10$	$n=20$	$n=50$	$n=100$	$n=200$	$n=500$
$\alpha = 0.01$	0.2976	0.1145	0.0377	0.0191	0.0086	0.0037
$\alpha = 0.05$	0.1548	0.0589	0.0209	0.0098	0.0045	0.0020
$\alpha = 0.1$	0.0952	0.0378	0.0131	0.0061	0.0028	0.0013
$\alpha = 0.2$	0.0437	0.0184	0.0060	0.0028	0.0011	0.0006
$\alpha = 0.8$	-0.0635	-0.0232	-0.0078	-0.0037	-0.0018	-0.0007
$\alpha = 0.9$	-0.0794	-0.0294	-0.0097	-0.0045	-0.0021	-0.0008

Table 1: Hoeffding cutoff values for $\hat{D}'$ based on 5000 Monte Carlo simulations.

We note this test is not as powerful as Pearson's test when the dependency is linear.

Modern tests of independence

In the following subsections we outline the modern tests of independence.

Density-based empirical likelihood ratio test for independence

A density-based empirical likelihood ratio test is proposed in Vexler et al. (2014) as a nonparametric test of dependence for two variables. Following Vexler et al. (2014), to test the null hypothesis we use the test statistic VT_n defined as:

$$VT_n = \max_{0.5n^{\beta_2} \le m \le \gamma_n} \max_{0.5n^{\beta_2} \le r \le \gamma_n} \prod_{i=1}^{n} \frac{n\widetilde{\Delta}_i(m,r)}{2m}, \tag{13}$$

where $\gamma_n = \min(n^{0.9}, n/2), 0.75 < \beta_2 < 0.9$ and $\widetilde{\Delta}_i(m,r)$ is defined as,

$$\widetilde{\Delta}_i(m,r) \equiv \frac{\hat{F}(X_{(s_i+r)},Y_{(i+m)})-\hat{F}(X_{(s_i-r)},Y_{(i+m)})-\hat{F}(X_{(s_i+r)},Y_{(i-m)})+\hat{F}(X_{(s_i-r)},Y_{(i-m)})+n^{-\beta_1}}{\hat{F_X}(X_{(s_i+r)})-\hat{F_X}(X_{(s_i-r)})}, \tag{14}$$

where $\hat{F}$ is the empirical joint distribution of X, Y and $Y_{(1)} \leq Y_{(2)}, \ldots, \leq Y_{(n)}$ are the order statistics of $Y_1, Y_2, \ldots Y_n$ and $X_{t(i)}$ is the concomitant of the ith order statistic defined in David and Nagaraja (1970). $\hat{F}_X$ is the empirical marginal distribution of X, s_i is the integer number such that $X_{(s)} = X_{t(i)}$, $\beta_1 \in (0, 0.5)$.

The statistic VT_n reaches its maximum with respect to $m \geq 0.5n^{\beta_2}$ and $r \geq 0.5n^{\beta_2}$ at $m = 0.5n^{\beta_2}$ and $r = 0.5n^{\beta_2}$ (Vexler et al., 2014). Thus, we simplify (13) to

$$VT_n = \prod_{i=1}^{n} n^{1-\beta_2}\widetilde{\Delta}_i\left([0.5n^{\beta_2}], [0.5n^{\beta_2}]\right), \tag{15}$$

where the function $[x]$ denotes the nearest integer to x. Accordingly, a size α rejection rule of the test is given by:

$$\log(VT_n) > C_\alpha, \tag{16}$$

where C_α is an α-level test threshold. It is established in Vexler et al. (2014) that VT_n is distribution free under (1) and the critical values C_α are estimated by 50,000 Monte Carlo simulations from $X_1, \ldots, X_n \sim Uniform[0,1]$ and $Y_1, \ldots, Y_n \sim Uniform[0,1]$.

This test is very powerful for many nonparametric alternatives as it is designed to approximate corresponding nonparametric likelihood ratios, however, when the dependency is linear with normal data other methods such as Pearson and Spearman tests will outperform this method.

Kallenberg data-driven test for independence

Kallenberg and Ledwina (1999) propose two data-driven rank tests for independence, $TS2$ and V. The $TS2$ statistic uses the intermediate statistic T_k defined as:

$$T_k = \sum_{j=1}^{k}\left\{\frac{1}{\sqrt{n}}\sum_{i=1}^{n} b_j\left(\frac{R_i - \frac{1}{2}}{n}\right) b_j\left(\frac{S_i - \frac{1}{2}}{n}\right)\right\}^2, \tag{17}$$

where b_j denotes the jth orthonormal Legendre polynomial. The selection of the order k in T_k is done by the modified Schwarz's rule, given by

$$S2 = \min\{1 \leq k \leq d(n), T_k - k\log n \geq T_j - j\log n,\ j = 1, \ldots, d(n)\}, \tag{18}$$

where $d(n)$ is a sequence of numbers tending to infinity as $n \to \infty$. The test statistic is,

$$TS2 = \sum_{j=1}^{S2}\left\{\frac{1}{\sqrt{n}}\sum_{i=1}^{n} b_j\left(\frac{R_i - \frac{1}{2}}{n}\right) b_j\left(\frac{S_i - \frac{1}{2}}{n}\right)\right\}^2. \tag{19}$$

It is reported in Kallenberg and Ledwina (1999) that there is almost no change in the critical value of $TS2$ for $d(n) > 2$. By default, we choose $d(n) = 4$. The decision rule to reject the null hypothesis in (2) is

$$TS2 > C_\alpha, \tag{20}$$

where C_α is an α-level test threshold.

In Kallenberg and Ledwina (1999) the $TS2$ test statistic in (19) is called the "diagonal" test statistic and the other test statistic, V is called the "mixed" statistic due to the fact that it involves "mixed" products. To derive the V statistic, we only consider the case $d(n) = 2$ and have sets of indexes $\{(1,1)\}$, $\{(1,1),(i,j)\}$, where $(i,j) \neq (1,1)$. Let Λ be one of these sets and define

$$T_\Lambda = \sum_{(r,s)\in\Lambda} V(r,s), \tag{21}$$

where

$$V(r,s) = \left\{\frac{1}{\sqrt{n}}\sum_{i=1}^{n} b_r\left(\frac{R_i - \frac{1}{2}}{n}\right) b_s\left(\frac{S_i - \frac{1}{2}}{n}\right)\right\}^2. \tag{22}$$

Letting $|\Lambda|$ denote the cardinality of Λ, we now search for a model, say $\Lambda^{(max)}$, for which $T_\Lambda - |\Lambda|\ \log n$

is maximized. Having obtained $\Lambda^{(max)}$, the second test statistic of (2) is

$$V = T_{\Lambda^{(max)}}. \tag{23}$$

It can be easily seen that the test statistic V can be rewritten (for $d(n) = 2$) in the simple form,

$$V = \begin{cases} V(1,1) & \text{, if } \max\left\{V(1,2), V(2,1), V(2,2)\right\} < \log n \\ V(1,1) + \max\left\{V(1,2), V(2,1), V(2,2)\right\} & \text{, otherwise.} \end{cases}$$

A size α rejection rule for the test is given by

$$V > C_\alpha, \tag{24}$$

where C_α is a size α critical value.

This test is very powerful when data are from an exponential family, however the statistic is rather complicated and a required assumption is that the data distributions belong to semiparametrically defined families.

Maximal information coefficient

The maximal information coefficient (MIC) proposed in Reshef et al. (2011) is a measure of strength of the linear and non-linear association between two variables X and Y. The maximal information coefficient uses binning as a means to apply mutual information on continuous random variables. Defining $\mathcal{D}$ as a finite set of ordered pairs, we can partition the x-values of $\mathcal{D}$ into x bins and the y-values of $\mathcal{D}$ into y bins, allowing empty bins. We call such partition an x-by-y grid, denoted G. For a fixed $\mathcal{D}$, different grids G results in different distributions $\mathcal{D}|G$. The MIC of a set $\mathcal{D}$ of two-variable data with sample size n and grid size less than $B(n)$ is defined as:

$$MIC(\mathcal{D}) = \max_{xy<B(n)} \{M(\mathcal{D})_{x,y}\}, \tag{25}$$

where x and y are observed values of variables X and Y, $\omega_{(1)} < B(n) \le o(n^{1-\varepsilon})$ for some $0 < \varepsilon < 1$. Note $M(\mathcal{D})_{x,y}$ is called the characteristic matrix of a set $\mathcal{D}$ of two-variable data x, y and defined as:

$$M(\mathcal{D})_{x,y} = \frac{I^*(\mathcal{D}, x, y)}{\log \min\{x, y\}}, \tag{26}$$

where $I^*(\mathcal{D}, x, y)$ is defined as:

$$I^*(\mathcal{D}, x, y) = \max I(\mathcal{D}|G), \tag{27}$$

for a finite set $\mathcal{D} \subset \mathbb{R}^2$ and positive integers x, y. We define the mutual information of two discrete random variables X and Y as:

$$I(\mathcal{D}|G) = \sum_{y\in Y} \sum_{x\in X} \hat{F}(x,y) \log\left(\frac{\hat{F}(x,y)}{\hat{F}_X(x)\hat{F}_Y(y)}\right), \tag{28}$$

The MIC statistic in (25) is computed using the R package **minerva** (see Albanese et al. (2013)). To our knowledge there is no hypothesis tests published to detect general structures of dependence based on MIC. We use an approach similar to the one in Simon and Tibshirani (2014) to develop an exact test based on the MIC statistic. A size α MIC test can be given by,

$$MIC(\mathcal{D}) > C_\alpha \tag{29}$$

where C_α is a size α critical value. To evaluate our approach, we simulated 5000 Monte Carlo sets of independent random variables X and Y of size n from standard normal distribution, exponential distribution and reverse of the standard normal distribution. The cutoff results are shown in Table 2. Regardless of the data distribution, the cutoff values for a given sample size are very similar.

It has been noted in Simon and Tibshirani (2014) that the MIC approach has serious power deficiencies and when used for large-scale exploratory analysis will result in too many false positives.

Empirical likelihood based test for independence

An empirical likelihood based test is proposed by Einmahl and McKeague (2003). The approach is via localizing the empirical likelihood with one or more "time" variables implicit in the given null hypothesis and then constructing an omnibus test statistic by integrating the log-likelihood ratio over

	$n = 10$	$n = 35$	$n = 75$	$n = 100$
	N ǀ E ǀ RN	N ǀ E ǀ RN	N ǀ E ǀ RN	N ǀ E ǀ RN
$\alpha = 0.01$	0.61 ǀ 0.61 ǀ 0.61	0.50 ǀ 0.52 ǀ 0.50	0.38 ǀ 0.37 ǀ 0.38	0.33 ǀ 0.33 ǀ 0.33
$\alpha = 0.05$	0.61 ǀ 0.61 ǀ 0.61	0.43 ǀ 0.43 ǀ 0.43	0.33 ǀ 0.33 ǀ 0.33	0.30 ǀ 0.30 ǀ 0.30
$\alpha = 0.1$	0.61 ǀ 0.61 ǀ 0.61	0.41 ǀ 0.40 ǀ 0.41	0.31 ǀ 0.31 ǀ 0.31	0.28 ǀ 0.28 ǀ 0.28
$\alpha = 0.5$	0.24 ǀ 0.24 ǀ 0.24	0.31 ǀ 0.30 ǀ 0.31	0.25 ǀ 0.25 ǀ 0.25	0.23 ǀ 0.24 ǀ 0.23
$\alpha = 0.75$	0.24 ǀ 0.24 ǀ 0.24	0.27 ǀ 0.26 ǀ 0.27	0.23 ǀ 0.23 ǀ 0.23	0.21 ǀ 0.21 ǀ 0.21
$\alpha = 0.9$	0.12 ǀ 0.12 ǀ 0.12	0.24 ǀ 0.23 ǀ 0.24	0.21 ǀ 0.21 ǀ 0.21	0.20 ǀ 0.20 ǀ 0.20
$\alpha = 0.95$	0.12 ǀ 0.12 ǀ 0.12	0.23 ǀ 0.22 ǀ 0.23	0.20 ǀ 0.20 ǀ 0.20	0.19 ǀ 0.19 ǀ 0.19
$\alpha = 0.99$	0.11 ǀ 0.11 ǀ 0.11	0.20 ǀ 0.20 ǀ 0.20	0.18 ǀ 0.18 ǀ 0.18	0.17 ǀ 0.17 ǀ 0.17

Table 2: Cutoff values from 5000 Monte Carlo simulations for a normal distribution (N), exponential distribution (E) and reverse normal distribution (RN).

those variables. We first consider a null hypothesis,

$$H_0 : F_X = F_0, \tag{30}$$

where F_0 is a fully specified distribution function. We define the localized empirical likelihood ratio $R(x)$ as:

$$R(x) = \frac{sup\{L(\tilde{F_X}) : \tilde{F}_X(x) = F_0(x)\}}{sup\{L(\tilde{F}_X)\}}, \tag{31}$$

where $\tilde{F}$ is an arbitrary distribution function, $L(\tilde{F}_X) = \prod_{i=1}^{n}(\tilde{F}_X(X_i) - \tilde{F}_X(X_i-))$. The supremum in the denominator is achieved when $\tilde{F} = \hat{F}$, the empirical distribution function. While the supremum in the numerator is attained by putting mass $F_0/(n\hat{F}(x))$ on each observation up to and including x and mass $(1 - F_0(x))/(n(1 - \hat{F}(x)))$ on each observation beyond x (Einmahl and McKeague, 2003). This gives the log localized empirical likelihood ratio:

$$\log R(x) = nF(x)\log\frac{F_0(x)}{\hat{F}(x)} + n(1 - \hat{F}(x))\log\frac{1 - F_0(x)}{1 - \hat{F}(x)}. \tag{32}$$

Note that $0\log(a/0)$ is defined as 0. For an independence test, the local likelihood ratio test statistic is:

$$R(x,y) = \frac{sup\{L(\tilde{F}) : \tilde{F}(x,y) = \tilde{F}_X(x)\tilde{F}_Y(y)\}}{sup\{L(\tilde{F})\}}, \tag{33}$$

for $(x,y) \in \mathbb{R}^2$, with $L(\tilde{F}) = \prod_{i=1}^{n}\tilde{P}(\{X_i\})$, where $\tilde{P}$ is the probability measure corresponding to $\tilde{F}$. The log local likelihood ratio test statistic is then:

$$\begin{aligned} \log R(x,y) =& n\hat{P}(A_{11})\log\frac{\hat{F}_X(x)\hat{F}_Y(y)}{\hat{P}(A_{11})} \\ &+ n\hat{P}(A_{12})\log\frac{\hat{F}_X(x)(1-\hat{F}_Y(y))}{\hat{P}(A_{12})} \\ &+ n\hat{P}(A_{21})\log\frac{(1-\hat{F}_X(x))\hat{F}_Y(y)}{\hat{P}(A_{21})} \\ &+ n\hat{P}(A_{22})\log\frac{(1-\hat{F}_X(x))(1-\hat{F}_Y(y))}{\hat{P}(A_{22})}, \end{aligned} \tag{34}$$

where $\hat{P}$ is the empirical measure of joint probability and

$$\begin{aligned} A_{11} &= (-\infty, x] \times (-\infty, y], \\ A_{12} &= (-\infty, x] \times (y, \infty), \\ A_{21} &= (x, \infty) \times (-\infty, y], \\ A_{22} &= (x, \infty) \times (y, \infty). \end{aligned} \tag{35}$$

The test statistic T_{el} is defined as:

$$T_{el} = -2\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\log R(x,y)d\hat{F}_X(x)\hat{F}_Y(y), \tag{36}$$

where T_{el} is distribution-free. We reject H_0 when

$$T_{el} > C_\alpha, \tag{37}$$

where C_α is a size α critical value.

This test is characterized by moments and is very powerful for many nonparametric alternatives. However, when the dependency is linear with normally distributed data, this test is outperformed by Pearson and Spearman tests.

Kendall plot and area under Kendall plot

The Kendall plot, also called K-plot, is a visualization of dependence in a bivariate random sample. It is proposed in Genest and Boies (2012). Similar to a Chi-plot which detects association in random samples from continuous bivariate distributions, the K-plot adapts the concept of a probability plot to detect dependence.

The horizontal axis in a K-plot, $W_{i:n}$ is the expectation of the ith order statistics in a random sample of size n from the distribution K_0 of the H_i (defined below) under the null hypothesis of independence. (Genest and Boies, 2012). $W_{i:n}$ can be computed as:

$$W_{i:n} = n\binom{n-1}{i-1}\int_0^1 w\{K_0(w)\}^{i-1} \times \{1-K_0(w)\}^{n-i} dK_0(w), \tag{38}$$

for all $1 \le i \le n$. Note K_0 is given as:

$$K_0(w) = w - w\log(w), 0 \le w \le 1. \tag{39}$$

The vertical axis is the sorted H_i. Unlike $W_{i:n}$, H_i is computed base on information provided by data.

Note H_i is defined as:

$$H_i = \frac{1}{n-1}\sum_{i=1}^{n} Ind(X \le X_i, Y \le Y_i), \tag{40}$$

where Ind is indicator function.

Let $0 \le p \le 1$ then some properties of the K-plot are:

1. when Y = X, all points fall on curve $K^{-1}(p)$,
2. when Y = $-X$, all points fall on a horizontal line,
3. when X and Y are independent, the graph is linear.

The area under the Kendall plot (AUK) is proposed in Vexler et al. (2017) as an index to independence. It applies area-under-curve analysis and computes the area under the Kendall plot. Some properties are listed below:

1. when Y = X, $AUK = 0.75$,
2. when Y = $-X$, $AUK = 0$,
3. when X and Y are independent, $AUK = 0$.

Note $AUK = 0$ does not imply independence. The AUK is similar to computing the area under the curve (AUC) for a receiver operating curve (ROC). In an ROC curve the true positive rate (sensitivity) is plotted as a function of the false positive rate for different cut-off points. A large AUC (near 1) suggests a good prediction model whereas a small AUC (near 0.5) suggests a model that is no better than chance. Similarly, a large AUK suggests dependence while an AUK near 0 may be suggestive of independence and should be further examined using one of the proposed tests. Further, more complicated correlation patterns such as a mixture of positive and negative correlations as in Example 2 (see Data analysis examples) can be detected with the K-plot.

What is the package testforDEP

The R Package `testforDEP` includes two functions, `testforDEP()`, `AUK()` and one data frame, `LSAT`.

testforDEP

The function `testforDEP()` is the interface for the tests. The function `testforDEP()` takes two vectors X, Y as inputs and returns an S4 object containing the

1. test statistic,
2. p-value,
3. bootstrap confidence intervals.

The interface of `testforDEP()` is:

```
testforDEP(x, y, data, test = c("PEARSON", "KENDALL", "SPEARMAN",
         "VEXLER", "TS2", "V", "MIC", "HOEFFD", "EL"),
         p.opt = c("dist", "MC", "table"), num.MC = 10000,
         BS.CI = 0, rm.na = FALSE, set.seed = FALSE)
```

where `x` and `y` are two equal-length numeric vectors of input data. The parameter `data` is an alternative input that takes a data frame with two columns representing X and Y. When `x` or `y` are not provided, parameter `data` is taken as input. The parameter `test` specifies the hypothesis test to implement. A summary of the test options with the associated statistic returned is given below,

- "PEARSON", $T_{\hat{\gamma}}$,
- "KENDALL", Z_{τ},
- "SPEARMAN", T_{ρ},
- "VEXLER", $\log(VT_n)$,
- "TS2", $TS2$,
- "V", V,
- "MIC", $MIC(\mathcal{D})$,
- "HOEFFD", $\hat{D}$,
- "EL", T_{el}.

Parameter `p.opt` is the option for computing p-values in which p-values can be computed by the asymptotic null distribution of the test statistic (applicable for Pearson, Kendall and Spearman only), by an exact method (applicable for all tests) or by pre stored simulated tables based on an exact method. By default, `p.opt = "MC"`. The parameter `num.MC` gives the Monte Carlo simulation number and will only be used when `p.opt = "MC"`. When `p.opt = "dist"` or `p.opt = "table"`, `num.MC` will be ignored. To balance accuracy and computation time, `num.MC` must $\in [100, 10000]$, with `num.MC = 10000` as default. The parameter `BS.CI` specifies a $1-\alpha$ for bootstrap confidence intervals. When `BS.CI = 0`, the confidence intervals will not be computed. Parameter `rm.na` is a flag for removing rows with missing data. Parameter `set.seed` is a flag for setting the random number generator seed.

AUK

The function `AUK()` provides an interface for Kendall plots and AUK. It takes two vectors X, Y and returns a list containing:

1. AUK,
2. $W_{i:n}$,
3. sorted H_i,
4. bootstrap confidence intervals.

The interface of `AUK()` is:

```
AUK(x, y, plot = FALSE, main = "Kendall plot", Auxiliary.line = TRUE,
        BS.CI = 0, set.seed = FALSE)
```

where `x` and `y` are two equal-length numeric vectors of input data. Parameter `plot` is a flag for drawing the Kendall plot. Parameter `main` determines the title of the plot. If `plot = FALSE`, `main` will be ignored. Parameter `Auxiliary.lie` is a flag for auxiliary line. Parameter `BS.CI` specifies a $1-\alpha$ for bootstrap confidence intervals. The bootstrap confidence intervals were produced using a non-parametric bootstrap procedure with a normal, pivotal and percentile based confidence interval obtained as described in Wasserman (2013). The bootstrap confidence intervals are on the scale of the test statistics that are returned from each test in **testforDEP**. When `BS.CI = 0`, confidence intervals will not be computed. Parameter `set.seed` is a flag for setting seed.

School	LSAT	GPA	School	LSAT	GPA	School	LSAT	GPA
1	622	3.23	28	632	3.29	56	641	3.28
2	542	2.83	29	587	3.16	57	512	3.01
3	579	3.24	30	581	3.17	58	631	3.21
4	653	3.12	31	605	3.13	59	597	3.32
5	606	3.09	32	704	3.36	60	621	3.24
6	576	3.39	33	477	2.57	61	617	3.03
7	620	3.10	34	591	3.02	62	637	3.33
8	615	3.40	35	578	3.03	63	572	3.08
9	553	2.97	36	572	2.88	64	610	3.13
10	607	2.91	37	615	3.37	65	562	3.01
11	558	3.11	38	606	3.20	66	635	3.30
12	596	3.24	39	603	3.23	67	614	3.15
13	635	3.30	40	535	2.98	68	546	2.82
14	581	3.22	41	595	3.11	69	598	3.20
15	661	3.43	42	575	2.92	70	666	3.44
16	547	2.91	43	573	2.85	71	570	3.01
17	599	3.23	44	644	3.38	72	570	2.92
18	646	3.47	45	545	2.76	73	605	3.45
19	622	3.15	46	645	3.27	74	565	3.15
20	611	3.33	47	651	3.36	75	686	3.50
21	546	2.99	48	562	3.19	76	608	3.16
22	614	3.19	49	609	3.17	77	595	3.19
23	628	3.03	50	555	3.00	78	590	3.15
24	575c	3.01	51	586	3.11	79	558	2.81
25	662	3.39	52	580	3.07	80	611	3.16
26	627	3.41	53	594	2.96	81	564	3.02
27	608	3.04	54	594	3.05	82	575	2.74
			55	560	2.93			

Table 3: LSAT data from Efron and Tibshirani (1994).

Data analysis examples

Here we present two data analysis example to demonstrate the utility of the **testforDEP** package. In the first example, the data are average law school admission test (LSAT) and grade point average (GPA) from 82 law schools (details described in Efron and Tibshirani (1994)). The data frame "LSAT" is available in **testforDEP** package. The aim is to determine the dependence between LSAT and GPA. Table 3 gives the data and a scatter plot is shown in Figure 1 (Top Left).

Figure 1 (Top Left) suggests a linear positive relationship between LSAT and GPA. To confirm this, we draw the Kendall plot and compute AUK (see Figure 1 (Top Right)). Figure 1 (Top Right) shows a curve above the diagonal and AUK of 0.665, both of which suggest a positive dependence.

Now consider the dependence tests provided in package **testforDEP**. Table 4 shows the test statistics and p-values. All tests, classical and modern, suggest dependence between LSAT and GPA. From this analysis, it is clear that LSAT and GPA variables are dependent.

To further examine this data and explore the comparable power of the modern methods, we also examine a subset of 15 randomly chosen points from the full dataset as shown in Figure 1 (Bottom Left). An analysis of this subset reveals significant p-values for the empirical likelihood, density based empirical likelihood and Kallenberg tests (see Table 4). The small p-values, especially for the density-based empirical likelihood method, indicate the power of modern methods to detect non linear dependence in small sample sizes. We further explore this point in the next example.

In the second example, we demonstrate the power of the modern methods to detect non-linear dependence. This example was originally published in Rohrer et al. (2004) and re-examined in Chen et al. (2010). It involves analyzing microarray expression data collected to identify critical genes in photoreceptor degeneration. The data in Figure 2 are from 2 genes (Pcp2 and Pla2g7) with each point representing another gene with its x-coordinate being the Euclidean distance from Pla2g7 and the y-coordinate as the Euclidean distance from Pcp2. The Euclidean distance is calculated from expression profiles between wild type (*wt*) and rod degeneration (*rd*) mice. The correlation shown

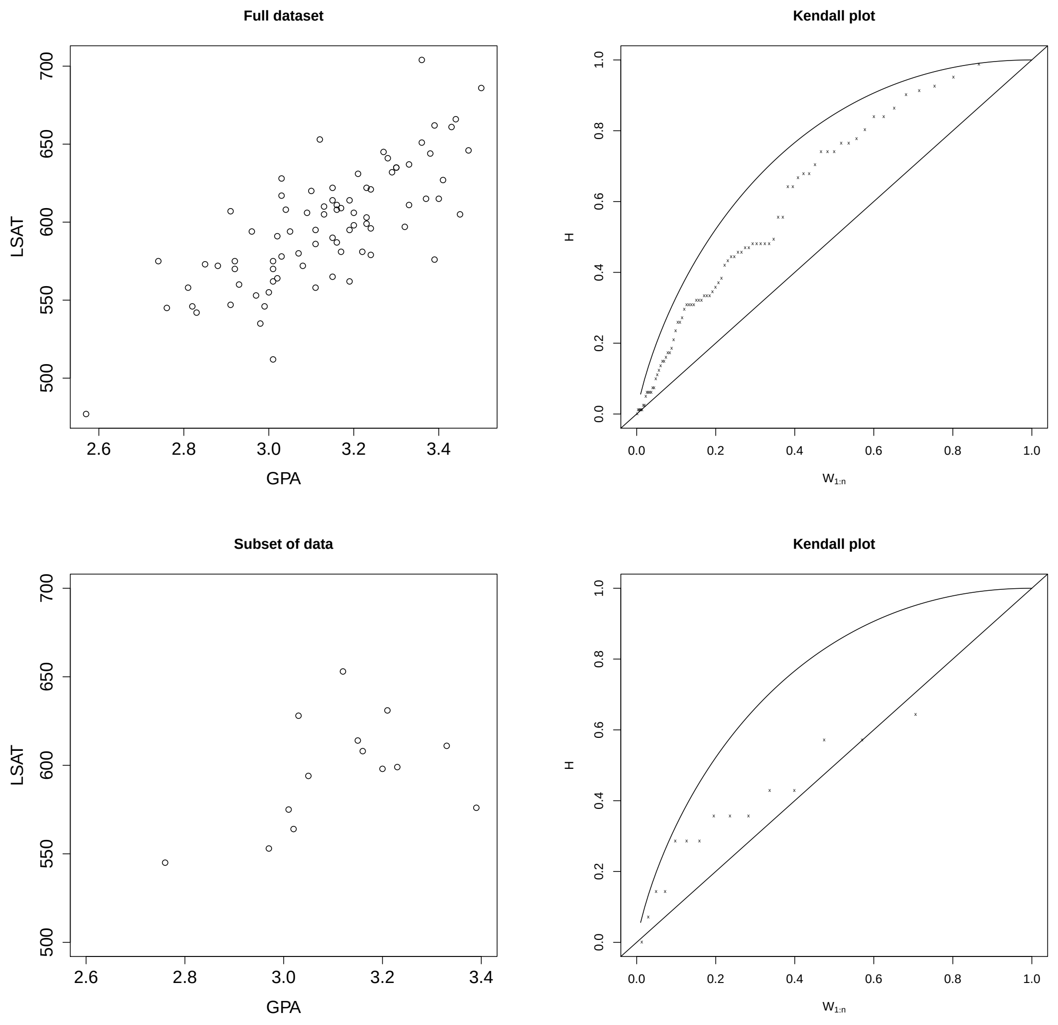


Figure 1: Top Left: Scatter plot based on data in Table 3. Top Right: Kendall plot of the full dataset. Bottom Left: A scatter plot of a subset of the LSAT/GPA data. Bottom Right: A Kendall plot of the subset of data.

	Full Data		Subset Data	
test	statistic	p-value	statistic	p-value
Pearson	10.46	< 0.0001	1.77	0.1074
Kendall	7.46	< 0.0001	69.00	0.0990
$\log(VT_n)$	65.70	< 0.0001	12.05	**0.0067**
TS_2	80.76	< 0.0001	3.16	0.2003
V	69.04	< 0.0001	8.51	**0.0146**
MIC	0.53	< 0.0001	0.38	0.0993
Hoeffding	0.22	< 0.0001	0.08	0.0658
T_{el}	13.64	< 0.0001	2.15	**0.0238**

Table 4: Example 1: Test results for the full LSAT/GPA data and for a randomly chosen subset of 15 data points.

test	statistic	p-value
Pearson	-1.015	0.3156
Kendall	-2.321	0.0196
Spearman	-2.900	0.0050
$\log(VT_n)$	140.554	< 0.0001
TS_2	13.347	0.0042
V	82.712	< 0.0001
MIC	0.666	< 0.0001
Hoeffding	0.085	< 0.0001
T_{el}	12.254	< 0.0001

Table 5: Example 2: Dependence test results for the microarray study with genes Pla2g7 and Pcp2 displayed in Figure 2 and published in Rohrer et al. (2004) with re-examination in Chen et al. (2010).

in Figure 2 measures the relationship between how the Pla2g7 gene interacts with all the other genes vs. how the Pcp2 gene interacts with all the other genes. From Figure 2 we see the genes are negatively correlated when their transformed levels are both less than 5. Otherwise, these genes are positively correlated. We proceed to analyze the data using the **testforDEP** package. The results are presented in Table 5. From this analysis, we see a large gain in power using the modern methods such as $\log(VT_n), TS_2, V, MIC, Hoeffding, T_{el}$ compared to the traditional methods (Pearson, Kendall, Spearman). Taken together, these examples demonstrate the power advantage of using modern methods to detect nonlinear dependence while still maintaining comparable power to traditional methods in detecting linear dependence.

Conclusion

The R package **testforDEP** is a new package that allows users for the first time to analyze complex structures of dependence via modern test statistics and visualization tools. Moreover, a novel exact method based on the MIC measurement have been proposed in the package **testforDEP**. Future work is necessary to further develop a formal testing structure based on the MIC statistic. We note that users of this package should carefully consider the strengths and weaknesses of the proposed tests as outlined in each section when deciding what test to apply. Further, a topic for future work is to develop a complete set of consistent criteria to determine which test is most appropriate for a specific setting. We note extensive simulations comparing each of the tests is provided in Miecznikowski et al. (2017) which may help guide users to choose among the tests. Ultimately, we believe that the package **testforDEP** will help investigators identify dependence using the cadre of modern tests implemented to detect dependency.

Acknowledgments

Albert Vexler's efforts were supported by the National Institutes of Health (NIH) grant 1G13LM012241-01. We are grateful to the Editor, Associate Editor and the two referees for helpful comments that clearly improved this manuscript.

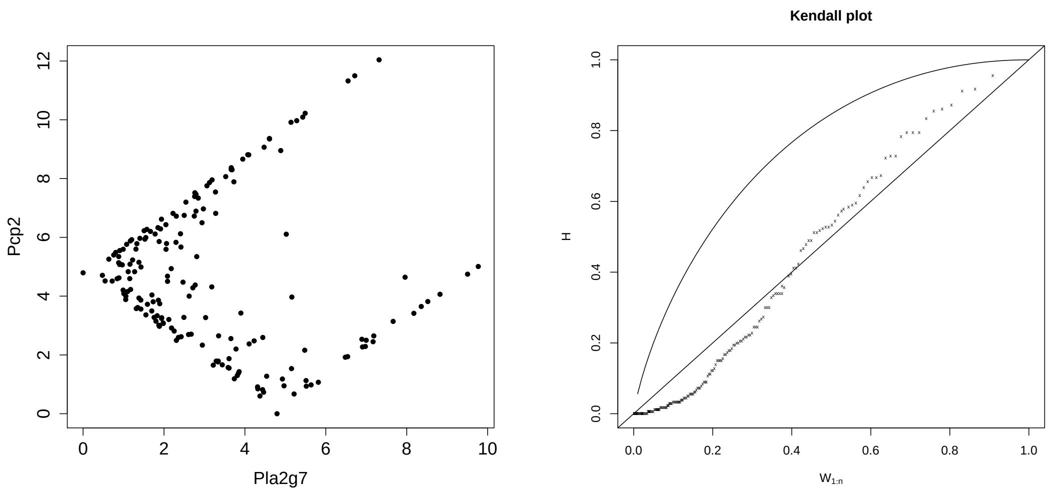


Figure 2: Left: Scatter plot of the transformed expression levels of Pla2g7 and Pcp2 genes studied in Chen et al. (2010). Right: Kendall plot of the dataset highlighting the negative correlation (points below $y = x$ line) for smaller expression values and positive dependence (points above $y = x$ line) for larger values.

Bibliography

D. Albanese, M. Filosi, R. Visintainer, S. Riccadonna, G. Jurman, and C. Furlanello. Minerva and minepy: a C engine for the MINE suite and its R, Python and MATLAB wrappers. *Bioinformatics*, 29 (3):407–408, 2013. URL `https://doi.org/10.1093/bioinformatics/bts707`. [p286]

Y. A. Chen, J. S. Almeida, A. J. Richards, P. Müller, R. J. Carroll, and B. Rohrer. A nonparametric approach to detect nonlinear correlation in gene expression. *Journal of Computational and Graphical Statistics*, 19(3):552–568, 2010. URL `https://doi.org/10.1198/jcgs.2010.08160`. [p290, 292, 293]

H. A. David and H. N. Nagaraja. *Order Statistics*. John Wiley & Sons, 1970. [p285]

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. CRC Press, 1994. [p290]

J. H. Einmahl and I. W. McKeague. Empirical likelihood based hypothesis testing. *Bernoulli*, pages 267–290, 2003. URL `https://doi.org/10.3150/bj/1068128978`. [p282, 286, 287]

C. Genest and J.-C. Boies. Detecting dependence with Kendall plots. *The American Statistician*, 2012. URL `https://doi.org/10.1198/0003130032431`. [p288]

F. E. Harrell Jr and M. C. Dupont. The hmisc package. *R package version*, 3:0–12, 2006. [p284]

F. E. Harrell Jr, with contributions from Charles Dupont, and many others. *Hmisc: Harrell Miscellaneous*, 2017. R package version 4.0-3. [p284]

J. Hauke and T. Kossowski. Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data. *Quaestiones Geographicae*, 30(2):87–93, 2011. URL `https://doi.org/10.2478/v10117-011-0021-1`. [p283]

W. Hoeffding. A non-parametric test of independence. *The Annals of Mathematical Statistics*, pages 546–557, 1948. URL `https://doi.org/10.1214/aoms/1177730150`. [p284]

W. C. Kallenberg and T. Ledwina. Data-driven rank tests for independence. *Journal of the American Statistical Association*, 94(445):285–301, 1999. URL `https://doi.org/10.1080/01621459.1999.10473844`. [p282, 285]

M. G. Kendall. A new measure of rank correlation. *Biometrika*, pages 81–93, 1938. URL `https://doi.org/10.1093/biomet/30.1-2.81`. [p282, 283]

J. Miecznikowski, E. Hsu, Y. Chen, and A. Vexler. testfordep: An R package for distribution-free tests and visualization tools for independence. *SUNY University at Buffalo Biostatistics Technical Reports*, 1701, 2017. [p283, 292]

K. Pearson. Notes on the history of correlation. *Biometrika*, pages 25–45, 1920. URL https://doi.org/10.1093/biomet/13.1.25. [p282, 283]

D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti. Detecting novel associations in large data sets. *Science*, 334(6062): 1518–1524, 2011. URL https://doi.org/10.1126/science.1205438. [p282, 286]

B. Rohrer, F. R. Pinto, K. E. Hulse, H. R. Lohr, L. Zhang, and J. S. Almeida. Multidestructive pathways triggered in photoreceptor cell death of the rd mouse as determined through gene expression profiling. *Journal of Biological Chemistry*, 279(40):41903–41910, 2004. URL https://doi.org/10.1074/jbc.M405085200. [p290, 292]

N. Simon and R. Tibshirani. Comment on "Detecting Novel Associations in Large Data Sets" by Reshef et al, *Science* Dec 16, 2011. *arXiv Preprint arXiv:1401.7645*, 2014. [p286]

C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. URL https://doi.org/10.2307/1412159. [p282, 283]

A. Vexler, W.-M. Tsai, and A. D. Hutson. A simple density-based empirical likelihood ratio test for independence. *The American Statistician*, 68(3):158–169, 2014. URL https://doi.org/10.1080/00031305.2014.901922. [p282, 284, 285]

A. Vexler, X. Chen, and A. D. Hutson. Dependence and independence: Structure and inference. *Statistical Methods in Medical Research*, 26(5):2114–2132, 2017. URL https://doi.org/10.1177/0962280215594198. [p282, 288]

Y. Wang, Y. Li, H. Cao, M. Xiong, Y. Y. Shugart, and L. Jin. Efficient test for nonlinear dependence of two continuous variables. *BMC Bioinformatics*, 16(1):1, 2015. URL https://doi.org/10.1186/s12859-015-0697-7. [p282]

L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer-Verlag, 2013. [p289]

Jeffrey Miecznikowski
Department of Biostatistics
University at Buffalo
Kimball Tower Rm 723
3435 Main Street, Buffalo NY 14214
E-mail: jcm38@buffalo.edu

En-shuo Hsu
Department of Biostatistics
University at Buffalo
3435 Main Street, Buffalo NY 14214
E-mail: daviden1013@gmail.com

Yanhua Chen
Department of Biostatistics
University at Buffalo
3435 Main Street, Buffalo NY 14214
E-mail: yanhua1108@gmail.com

Albert Vexler
Department of Biostatistics
University at Buffalo
Kimball Tower Rm 715
3435 Main Street, Buffalo NY 14214
E-mail: avexler@buffalo.edu

rFSA: An R Package for Finding Best Subsets and Interactions

by Joshua Lambert, Liyu Gong, Corrine F. Elliott, Katherine Thompson and Arnold Stromberg

Abstract Herein we present the R package **rFSA**, which implements an algorithm for improved variable selection. The algorithm searches a data space for models of a user-specified form that are statistically optimal under a measure of model quality. Many iterations afford a set of *feasible solutions* (or candidate models) that the researcher can evaluate for relevance to his or her questions of interest. The algorithm can be used to formulate new or to improve upon existing models in bioinformatics, health care, and myriad other fields in which the volume of available data has outstripped researchers' practical and computational ability to explore larger subsets or higher-order interaction terms. The package accommodates linear and generalized linear models, as well as a variety of criterion functions such as Allen's PRESS and AIC. New modeling strategies and criterion functions can be adapted easily to work with **rFSA**.

Introduction

In recent years, novel statistical modeling techniques have become more computationally intensive in an effort to accommodate the massive datasets afforded by advances in fields such as data mining and genetic sequencing. The volume and complexity of available data continue to grow, overwhelming even the fastest modern computers (Goudey, 2015). Efforts to analyze such complex data usually involve some level of data reduction (*e.g.*, Principle Component Analysis, Partial Least Squares, Sufficient Dimension Reduction), which can yield convoluted statistical models whose parameters researchers and statisticians alike struggle to interpret. Furthermore, many of these strategies are limited to specific model forms and lack the flexibility to explore higher-order interactions. Herein we present an alternative to data reduction that accommodates a variety of modeling strategies, including large-subset selection and identification of higher-order interaction terms. The resulting models, coefficient estimates, and predictions remain easily interpretable and flexible to traditional modes of statistical inference.

The Feasible Solutions Algorithm (FSA) overcomes the problems described above by searching a reduced data space to produce *feasible solutions* (Hawkins, 1994; Miller, 1984). Feasible models (Hawkins, 1994) are optimal under a given criterion function in the sense that no single exchange of an explanatory variable contained in the model for a variable outside the model will yield an improvement to the criterion. Miller (1984) first introduced the idea of a sequential-replacement algorithm such as the FSA. According to Miller, this computationally "cheap" method boasts rapid convergence; flexible implementation; and improved results compared to forward or backward selection. Miller also noted that the replacement algorithm could give too many solutions if repeated. However, Hawkins (1994) applied a similar exchange algorithm to the problem of finding robust regression estimators and minimum-volume ellipsoid estimators in multivariate data, demonstrating that with a sufficient number of random starts, the algorithm could find the optimal solution with arbitrarily high probability. Upon testing the algorithm, Hawkins also exhibited its superior performance relative to exhaustive search.

Exhaustively searching for an optimal model containing interactions is particularly demanding, computationally speaking, and not always reasonable or even attainable. For this reason, many published analyses do not even attempt to explore interactions. In other cases, interaction terms are identified by a primary investigator on the basis of his or her prior knowledge of the field, and then screened on an individual level by a statistician or data scientist. Such a process is tedious and time-consuming, and usually results in interactions being ignored or overlooked due to the sheer number of possibilities. These factors unite to afford a widespread lack of consideration for interactions, thereby undermining the predictive power of models attempting to capture complex relationships (Foster and Stine, 2004). We address these limitations by implementing an FSA with the capacity to explore higher-order terms, combined with the accessibility and ease of use associated with an R package.

The R implementation of our Feasible Solution Algorithm, **rFSA 0.9.1**, is now accessible *via* GitHub and CRAN.

Feasible Solutions Algorithm

Statisticians are often faced with the problem of identifying an informative subset of m explanatory variables from a set of p predictor terms, which we will denote $\mathbf{X}^p$. Consider selecting $p^+ \geq 0$ explanatory variables, denoted $\mathbf{X}^{p^+}$, to compose a preliminary model. Let $g(\mathbf{Y}, \mathbf{X})$ be an objective (criterion) function, generally a measure of model quality such as R^2, AIC, or PRESS. We wish to identify the m additional variables, $\mathbf{X}^m$, that when added to the existing model serve to optimize the objective function $g(\mathbf{Y}, \mathbf{X}^{(p^+ + m)})$.

The Feasible Solution Algorithm implements the following strategy, where we initially consider only first-order predictor terms:

1. Randomly select m variables to compose $\mathbf{X}^m$, and compute the objective function $g(\mathbf{Y}, \mathbf{X}^{(p^+ + m)})$.
2. Under each possible exchange of one of the m variables in the working model for a variable not contained in the model, compute the new value of the objective function.
3. Make the single exchange of variables from Step 2 that most improves the objective function.
4. Continue making exchanges until no single additional exchange improves the objective function. The variables $\mathbf{X}^{p+}, \mathbf{X}^{m^*}$ composing the final model constitute a single feasible solution.
5. Return to (1) to search for another feasible solution.

Miller (1984) illustrates the general procedure as follows: Suppose you have a dataset containing 26 predictor variables labeled A through Z, and imagine you wish to find the best subset of four predictor variables. Randomly select four initial predictors; suppose these are $ABCD$. Consider exchanging one of A, B, C, or D with each of the 22 remaining variables. Make the change that most improves the objective function; suppose we swap C for X, so that the working subset becomes $ABXD$. Next consider exchanging one of A, B, X, or D. (In point of fact, considering an exchange involving X is here redundant and unnecessary.) Repeat this process until no additional exchange of variables yields further improvement to the objective function.

If instead a user requests interaction terms of m^{th} order, our FSA randomly selects m effects to compose an initial interaction term in step (1). In step (2), the algorithm considers exchanging any one of the m variables involved in the interaction term. In accordance with the hierarchical paradigm, all associated lower-order interactions and main effects are exchanged as well, prior to calculating the associated objective function; no linear or lower-order terms are included in the model beyond those required to complete the hierarchy unless the user requests a variable be fixed in the model. The sequential-replacement procedure otherwise operates as described above, but this extension permits optimization with respect to the p-value of an interaction term as an alternative to standard model-building criteria.

Each iteration of the FSA yields a single feasible solution, which is the most-optimal model under a given criterion that can be reached from the (random) starting configuration by means of sequential replacement. Of course, depending on the initialized model, the algorithm may not converge to the *global* optimum with respect to the specified objective function. However, if a truly correct model exists and consists of explanatory variables present in the dataset, then under an appropriate objective function and with many iterations, **rFSA** is increasingly likely to discover it: A model containing correct explanatory variables should yield a value for the criterion superior to that of a model containing explanatory variables unrelated to the response. In our experience with **rFSA**, the models best supported by the data tend to manifest more often than inferior models, but it can be shown that with many random starts even relatively rare (locally or globally) optimal configurations may be identified (Hawkins, 1994). Thus, with multiple random starts, the FSA is likely to discover the optimal model, as well as additional models that would be overlooked if one considered only the 'best' model under a specified criterion, but which may contain additional information of interest from a clinical or scientific viewpoint.

Other algorithms for subset selection

Many existing methods attempt to identify the best subset of predictors that adequately explains a response variable. Common automatic variable-selection techniques include forward selection, backward elimination, and step-wise regression methods, while penalized regression techniques, such as LASSO, are commonly used for subset selection with many variables. Exhaustive search procedures simply examine all possible combinations of predictors under a viable model structure — but even on the most-powerful computers, this approach becomes impracticable for datasets containing many explanatory variables (Goudey, 2015). Algorithms implementing these procedures in the context of linear or generalized linear models, respectively, are currently available in the form of R packages **leaps** (Lumley and Miller, 2009), **glmulti** (Calcagno, 2013), and **glmnet** (Friedman, 2008).

Other algorithms for finding interactions

Computational methods for exploring potential interactions include Random Forest (Jiang, 2009) and Boosting (Lampa, 2014) methods. In the former approach, researchers examine branches of a regression tree to identify splits that contribute downstream to alternative classifications of the response variable. Bayesian methods also exist for identifying interactions, although they tend to specialize in exploration of large genetic datasets (Zhang and Liu, 2007). LASSO can be used to identify interactions by selecting informative variables from an initial pool containing all possible interactions as well as first-order terms (Bien et al., 2013), and LASSO for Hierarchical Interactions has been implemented in the R package **hierNet** (Bien and Tibshirani, 2014). Although all of these methods have been used previously for identifying interactions, the software for implementing them are limited with respect to the statistical methods they utilize and criterion functions they oblige. The statistical community would therefore benefit from a robust algorithm whose package can accommodate multiple statistical methods and criterion functions.

rFSA

Our R package **rFSA** implements the FSA described above for use in subset selection and identification of interaction terms. The primary function, `FSA`, accepts as two of its arguments any user-specified R functions for use in fitting models (*e.g.*, `lm`, `glm`) and calculating the model criterion (*e.g.*, `AIC`, `BIC`, `r.squared`), with only the restrictions that the criterion function must (1) accept as its argument the model object returned by the specified model-fitting function and (2) return a single numeric value. Additional arguments to `FSA` include the number of parameters to consider, whether to investigate interactions with or without quadratic terms, and whether to minimize or maximize the criterion function.

Architecture

Within the R package **rFSA**, the function `FSA` is used to identify feasible models. A full list of the parameters belonging to `FSA` are described below, for reference in illustrating the architecture of the package:

Parameters:

`formula`	symbolic description of the functional model form to be fitted. All subsequent analysis will model the response variable designated by this parameter.
`data`	data frame containing the set of predictor variables of interest.
`fitfunc`	function to fit the model. Defaults to `lm`.
`fixvar`	vector of one or more variable names to be fixed in the model as main effects. Defaults to NULL.
`quad`	logical: whether to consider higher-order interactions containing two instances of the same variable. Defaults to FALSE.
`m`	number of predictors to compose a model. If `interactions` is TRUE then *m* is the order of interactions to consider.
`numrs`	number of random starts to perform.
`cores`	number of cores to use while running. Restricted to 1 for a Windows machine.
`interactions`	logical: whether to include interactions in model. Defaults to TRUE.
`criterion`	vector of criterion function(s) to maximize or minimize. Defaults to `AIC`.
`minmax`	vector of strings `"min"` or `"max"` specifying whether to minimize or maximize each criterion function. Defaults to `"min"`.
`checkfeas`	vector of variable names composing a model to test for feasibility. If multiple random starts specified, test of feasibility will occupy the final random start. Defaults to NULL.
`var4int`	variable to be fixed in the model as one component of the interaction term. Defaults to NULL.
`min.nonmissing`	minimum threshold for observations required to fit a model. Defaults to 1.
`return.models`	logical: whether to return fitted model objects. Defaults to FALSE.
`...`	additional arguments to be passed to the model-fitting function.

Assume we wish to discover the first-order linear model containing three predictors, chosen from a pool of eight predictors, with maximal R^2. To increase our probability of discovering the global optimum with respect to our chosen criterion, we choose to execute ten random starts. To decrease the time until completion, we run the algorithm on five cores. Let `dat.all` denote the data frame containing the response variable, denoted Y, and all eight predictors. Then the code required to execute this search is provided below, acting on a simulated response variable and eight continuous predictors.

```
set.seed(40508)

# simulate explanatory variables, response, and uninformative variables
x1 <- rnorm(100)
x2 <- rnorm(100)
x3 <- rnorm(100)
dat.all <- data.frame(Y = 2 + 3*x1 - 4*x2 + 0.5*x3 + rnorm(100),
                      X1 = x1, X2 = x2, X3 = x3,
                      X4 = rnorm(100), X5 = rnorm(100), X6 = rnorm(100),
                      X7 = rnorm(100), X8 = rnorm(100))

# load package and execute rFSA
library(rFSA)
fsa.fit <- FSA(formula = 'Y~1', data = dat.all, fitfunc = lm, quad = FALSE, m = 3,
               numrs = 10, cores = 5, interactions = FALSE, criterion = r.squared,
               minmax = "max", return.models = FALSE)
print(fsa.fit)
```

rFSA first generates the ten random starts, each of which consists of three indices: one corresponding to each predictor in the initial model. For each random start, the algorithm generates every unique exchange of variables that can be achieved from the initial model. Corresponding models are fitted by calling the fitfunc, in this case lm, and the criterion function calculated for each of the resulting models. If the user specifies multiple cores, then the mcapply function from R package **parallel** is employed to fit multiple models simultaneously. Criterion values, here R^2 values, are stored in a hash table to prevent having to fit the same model multiple times across different random starts, thereby achieving an additional gain in computational efficiency as well as reducing the memory requirement. (If return.models is TRUE, then the model objects themselves are stored in a separate list.) **rFSA** makes the exchange that achieves the greatest improvement in model criterion, and then repeats the process of generating and making exchanges. The algorithm ceases iterating on a single random start when the best exchange yields no change in the model, or when the working model converges to a feasible solution discovered previously.

The FSA function is written as an S3 object. Returned results assume class definition FSA and can be used in conjunction with standard S3 functions print, summary, predict, fitted, and plot, described in greater detail below.

Implementation

The core of the package is the FSA optimization procedure, which is depicted as a flowchart in Figure (1). As this figure illustrates, the procedure can be divided into several sub-procedures as follows:

- The algorithm first generates a number (specified by user, here denoted M) of random starts $P_1, P_2 \cdots P_M$. Each member P_i is a random combination of variables from the predictor set.
- For each random start P_i, **rFSA** then generates all possible variable exchanges $Q_{i1}, Q_{i2} \cdots Q_{iN}$. Similar to a random start, each Q_{ij} is also a combination of variables, but differs from its corresponding start position P_i by precisely one variable. As a result, we obtain $M \times N$ combinations.
- Next, for each combination Q_{ij}, **rFSA** fits the model using the function specified by the input argument fitfunc and calculates the corresponding criterion value using the function specified by criterion. Thus we obtain a criterion value C_{ij} for each combination Q_{ij}.
- The criterion values $C_{i1}, C_{i2} \cdots C_{iN}$ are used to identify the optimal exchange among $Q_{i1}, Q_{i2}, \cdots, Q_{iN}$ from the starting configuration P_i. The best swap combination, denoted P_i', is that with the smallest or largest criterion depending on the user-specified argument minmax.
- Finally, the algorithm decides whether to continue iterating by comparing the best swap P_i' and initial combination P_i. If they are equal, then the algorithm stops processing and stores P_i as a feasible solution. Otherwise, it updates P_i to P_i' and continues iterating until a feasible solution is found.

Note that the computationally most-intensive stage of the algorithm occurs when obtaining the model fit and criterion values for combinations $Q_{11} \cdots Q_{1N}, \cdots, Q_{M1} \cdots Q_{MN}$. Moreover, some of this work is redundant, as the same model fit may be required as many as M times. We therefore elect

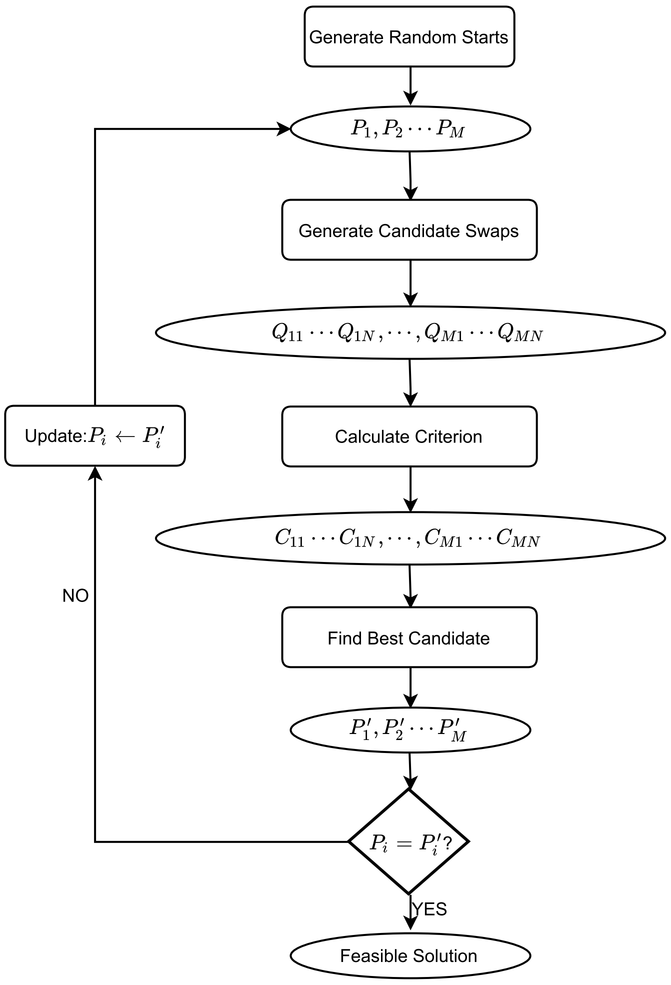


Figure 1: Overview of our implementation of the FSA optimization algorithm. A number of random starts are generated, followed by candidate variable exchanges and the criterion value associated with each new model. The best swap candidate is chosen according to the criterion values and then compared with the starting configuration. If they are equal, then they constitute a feasible solution. Otherwise, the starting configuration is updated by assuming the value of the best swap, and the procedure iterates until a feasible solution is found.

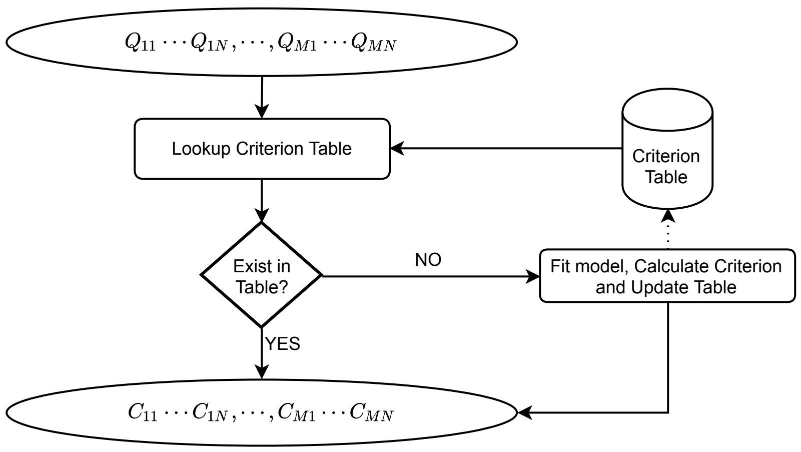

$Q_{11} \cdots Q_{1N}, \cdots, Q_{M1} \cdots Q_{MN}$
Lookup Criterion Table
Criterion Table
Exist in Table?
NO
Fit model, Calculate Criterion and Update Table
YES
$C_{11} \cdots C_{1N}, \cdots, C_{M1} \cdots C_{MN}$

Figure 2: Illustration of storing and referencing criterion values using a global table. For each combination of variables, we first check for the model criterion in the table and return it if found. Otherwise, we fit the model, calculate the criterion, and add the calculated value to the table. To further speed execution, model fitting for different combinations of variables is processed in parallel using multiple cores.

to store criterion values upon calculation, thereby reducing the computational burden and improving execution efficiency. A detailed schematic of this implementation is provided in Figure (2).

The `FSA` function implements the procedure depicted in Figure (2) to maintain a global table of criterion values for easy reference. Prior to calculating a model criterion, **rFSA** first checks whether the model has been fitted previously and, if so, uses the stored criterion value rather than refitting the model. If the model's criterion is not represented in the existing table, then **rFSA** fits the model, calculates the criterion value, and stores that value in the table. Specifically, we adopt following techniques to optimize our implementation of the algorithm:

- By maintaining a look-up table, we avoid redundant computations to fit the same model multiple times.
- We implement the look-up table as a hash table, which achieves constant-time insertion and querying.
- We use the R package **hashmap** (Russell, 2017) to implement the criterion table. **hashmap** is written in C++, yielding further improvement to the execution speed of **rFSA**.
- We fit models in parallel on multiple computation cores using the `mclapply` function of the **parallel** package. Because model fitting is independent across different sets of variables, the task is appropriate for multi-thread or multi-process parallelization.

Usage

In this section, we consider some nuances related to the arguments of the function `FSA`, and end with details of the results returned by the same function.

The `formula` parameter allows users to specify an initial fit. Because initial models are randomly generated, it is sufficient to specify a null model containing solely the desired response variable and an intercept term, *e.g.*, `Y ~ 1`. Variables to be fixed in the model should be provided in the form of a vector of character strings representing the variable names, provided to `FSA` as the argument `fixvar`.

The `data` structure should contain only the target response variable and predictor variables of interest, with each variable occupying a distinct column of the data structure. Variables known to be superfluous should be omitted from the data structure prior to executing `FSA`. Furthermore, categorical variables should be denoted in quotes or stored as factors to distinguish from quantitative variables, and should assume at least two levels to avoid complications due to invariance.

Models will be fitted using the user-specified `fitfunc`, to which the `formula`, `data`, and any additional arguments will be passed. The objective function `criterion` may be any user-specified R function that accepts as its argument the model object returned by `fitfunc`, provided the criterion function returns a single numeric value (or `NA`) that can be minimized or maximized. If users intend to write their own criterion functions, we recommend that they protect against errors by enshrouding the functions in `tryCatch` statements, available in base R. As an alternative, users may specify one of the criterion functions built into **rFSA**, which are detailed below. In either case, the specified criterion function will be maximized or minimized in accordance with the value of `minmax`.

The `mclapply` function used to support parallel processing in **rFSA** accommodates the use of multiple cores for Unix environments only. For this reason, **rFSA** automatically instantiates the `cores` parameter to 1 for Windows users, regardless of the user-specified value. For Unix machines with multiple cores, we recommend a maximum threshold for `cores` of one fewer than the number of cores available on the machine. Users can execute `parallel::detectCores()` to determine the number of cores on their computers.

To request subset selection without interactions, `interaction` must be set to `FALSE`. (In this case, the `quad` parameter will be ignored.) In this scenario, `m` denotes the number of predictors to compose each resulting subset. In the case of `interactions = TRUE`, the meaning of `m` changes to represent the desired interaction order. (During model fitting, all lower-order terms associated with the interactions are also included). In such an instance, the parameter `var4int` can be used to fix a variable as one member of the interaction term, thereby restricting the `FSA` function to identify m-way interactions containing the specified predictor along with any other $m - 1$ variables. The `quad` parameter specifies whether higher-order interactions should be permitted to contain the same variable twice. Regardless of these parameters, the `FSA` function yields feasible solutions equal in number to `numrs`. However, these models may not be unique if a feasible solution is accessible from multiple random starting positions. In order to provide cleaner results, **rFSA** produces a table summarizing these results (see Function Outputs, below).

Finally, the FSA function accommodates testing for the feasibility of a model, where the desired predictors may be provided as the argument checkfeas. If the model is feasible, then the resulting feasible solution will contain the same variables as checkfeas; otherwise, the FSA function returns the feasible model accessible from the initial model specified in checkfeas. If the user specifies multiple random starts, then the test of feasibility occupies the final random start.

Criterion Functions

Criterion functions built into **rFSA** include R^2 (r.squared) and adjusted R^2 (adj.r.squared), Allen's PRESS statistic (apress), interaction p-values (int.p.val), Akaike's Information Criterion (AIC) and the Bayesian Information Criterion (BIC), penalized Quasi-likelihood under the Independence model Criterion (QICu.geeglm), and Bhattacharyya Distance (bdist). These functions may not be appropriate for all functions provided to fitfunc as, for example, r.squared does not accommodate generalized linear models.

Most of the above metrics are well known and thus do not require introduction, but we summarize briefly two of the less-common statistics. The Bhattacharyya Distance (Bhattacharyya, 1946) is a distance measure that quantifies the disparity between two distributions. The Bhattacharyya Distance is particularly useful for a binary response when the user's interest lies in exploring two-way interactions between continuous explanatory variables (Janse, 2017). This is a common problem in large genetic datasets, in which context bdist is doubly useful because it converges faster than the other criterion functions. The Quasi-likelihood under the Independence model Criterion (QIC) (Pan, 2001) is a goodness-of-fit statistic for generalized estimating equations, analogous to AIC. The function built into **rFSA** accepts an object of class geeglm, fitted with **geepack** (Højsgaard et al., 2006; Yan and Fine, 2004; Yan, 2002), and returns the penalized QIC, denoted QICu, which is preferred for subset selection.

Function Outputs

A successful run of the FSA function returns a list with seven elements, described below:

Returned Values:

$originalfit	model object representing the fit of the user-specified original model.
$call	list of FSA function parameters and their actualized values.
$solutions	list of initial and final (*i.e.*, feasible) predictors contained in each model; criterion function values for feasible models; and number of exchanges made. If return.models=TRUE then solutions will contain an object of all fitted models called checked.model.
$table	data frame of unique feasible solutions; corresponding criterion values; and the number of random starts that converged to each solution.
$efficiency	character string contrasting the total number of fitted models and criterion values calculated using **rFSA** against those required for exhaustive search on the same dataset.
$nfits	integer representing the total number of fitted models.

Given an FSA object as its argument, the print command will display a table containing the original user-specified model and all feasible solutions that the algorithm found over numrs random starts. The table also contains criterion values for each model, and the number of random starts that converged to each feasible solution. Generally speaking, the original fit is not a feasible solution, and thus the number of random starts for that model will be listed as NA. The summary command acting on an object of class FSA will display a list containing as its elements the summary output of each fitted model; the format therefore depends on the chosen fitfunc. The predict and fitted functions operate in a similar fashion to summary, returning a list of predicted or fitted values, respectively, for each model. Finally, the plot function generates a Q-Q plot and a plot of residuals against fitted values for each model, displayed in a compact manner.

Availability

Currently, **rFSA version 0.9.1** is available to download from the Comprehensive R Archive Network at https://cran.r-project.org/web/packages/rFSA. To install the newest beta version of **rFSA**, first install the **devtools** package in R, and then run devtools::install_github("joshuawlambert/rFSA").

Shiny App

An easy-to-use Shiny Application is also available to facilitate the basic functions of the package. We believe this application will serve an important role for researchers unfamiliar with R, who would nonetheless benefit from the ability to describe and/or make predictions from large datasets. Our Shiny Application allows users to upload their own data and specify parameter values *via* radio buttons and drop-down boxes, as well as to visualize fundamental elements of the resulting models. The application currently subsists on a server hosted by the University of Kentucky, accessible from `https://shiny.as.uky.edu/mcfsa/`. **Important:** Users *should not* upload sensitive data to our FSA Shiny Application. To accommodate protected data, a future version of **rFSA** will permit users to run local instances of the Shiny Application.

Comparison to other packages

Several R packages exist already for finding best subsets and exploring pairwise interactions. Two of the most-popular packages in current use are the **leaps** (Lumley and Miller, 2009) package and **glmulti** (Calcagno, 2013). These packages use criterion functions such as R^2, adjusted R^2, Mallow's C_p, residual sum of squares, and AIC/BIC to identify the best subset of predictor variables to describe a given response variable. Although these packages are useful, they offer a limited selection of statistical models and criterion functions. In this section, we describe these and other limitations on existing packages, and explain how **rFSA** seeks to overcome them.

The **leaps** package leverages exhaustive search, forward or backward selection, or a sequential-replacement algorithm to find the best subset of predictors to compose a model. This sequential-replacement algorithm is a variation on the FSA introduced by Miller (1984) and described above, and therefore serves as a relatively efficient option for analyzing large datasets. Although the **leaps** package is flexible and robust, at present it does not accommodate interaction terms, model forms other than first-order linear, or the use of alternative criterion functions to the aforementioned five. The **bestglm** (McLeod and Xu, 2017) package seeks to extend best-subset model selection to generalized linear models but is not natively capable of looking for higher-order interactions, using external criterion functions, or accommodating other statistical methods. It also makes no special consideration for large problems, rendering it unsuitable for datasets with more than 100 predictors. **glmulti** improves on the aforementioned packages in that it is capable of incorporating pairwise interaction terms into exhaustive search or a genetic algorithm, but it does not support other statistical methods, higher-order interactions, or flexible criterion functions.

Timing comparisons

We now present the results of a simulation conducted to compare the performance of **rFSA** against that of of exhaustive search with **leaps**. We include neither **glmulti** nor **bestglm** in the comparison because neither package was able to accommodate the high-dimensional datasets of interest ($p > 100$).

Simulations were conducted on twenty-five datasets of $N = 250$ observations for various numbers of predictors, again denoted by p. In each dataset, a continuous response was generated randomly from a standard normal distribution. Half of the predictors were generated randomly from a standard normal distribution, and the other half, from a Bernoulli distribution with $P(X = 1) = 0.50$.

The execution speed of each package was measured individually for each dataset. Simulations were conducted in R version 3.4.1 on a Linux machine with Intel(R) Xeon E5-2698 v4 @ 2.20GHz with 128.00 GB of memory. A single core was allocated for each of `regsubsets` and the `FSA` function. Code follows for calling the relevant function in each of packages **leaps** (version 3.0) and **rFSA** (version 0.9.1) to search for best subsets of size three:

```
leaps::regsubsets(x = ..., y = ..., nbest = 1, nvmax = 3, really.big = T,
                method = "exhaustive")
rFSA::FSA(X1 $\sim$ 1, data = ..., interactions = F, m = 3, numrs = 1, criterion = AIC,
        minmax = "min")
```

Figure 3 compares the run-time in $\log(seconds)$ for these commands over twenty-five simulations and $p = (150, 250, 350, 450, 650, 750, 850)$ when searching for best subsets of size three. Exhaustive search with **leaps** exhibits a superior run-time on average for $p \leq 350$, while **rFSA** is more efficient when $p > 350$. **rFSA** was approximately 13 minutes faster for $p = 750$. The growth in execution time for **rFSA** slows relative to that of **leaps** as the number of variables increases: Although the overhead is higher for `FSA`, the costs associated with analyzing incrementally larger volumes of data are lower.

Figure 4 depicts run-times for twenty-five simulations of $p = (50, 100, 150, 200, 250, 300, 350)$ when searching for best subsets of size five. In this scenario, exhaustive search with **leaps** evinces a faster

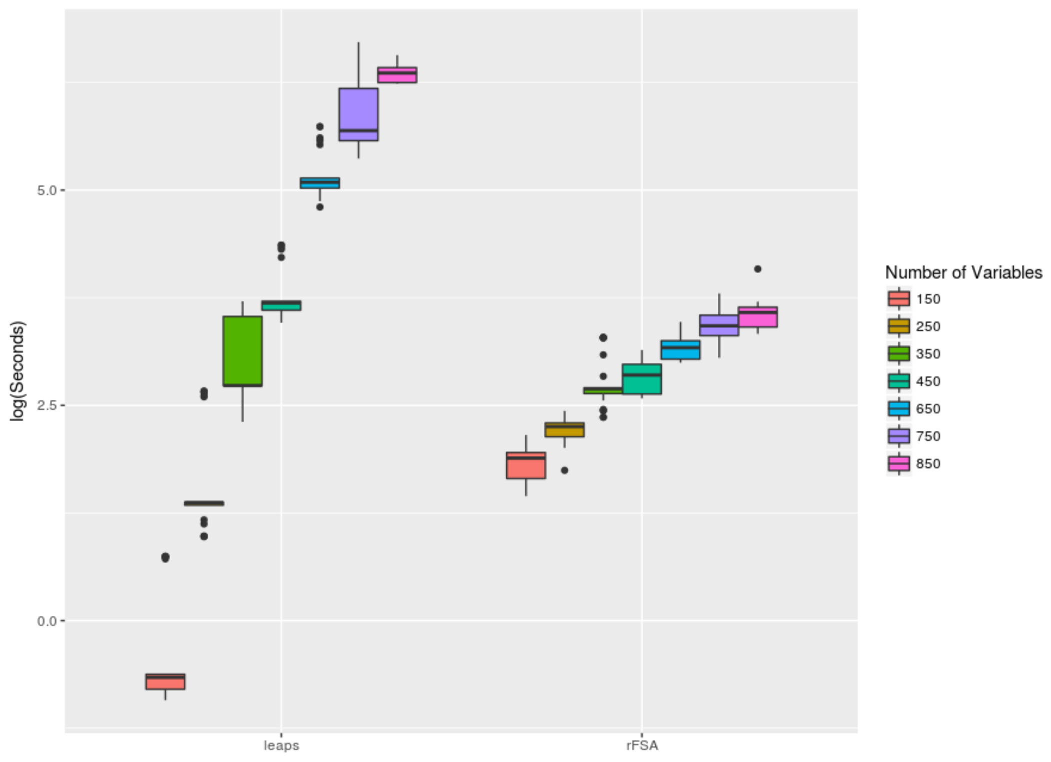


Figure 3: Timing comparison of exhaustive search with **leaps** *versus* **rFSA** with one random start. Each package was assigned to identify best subsets of size three from datasets of size $n = 250$ and various p.

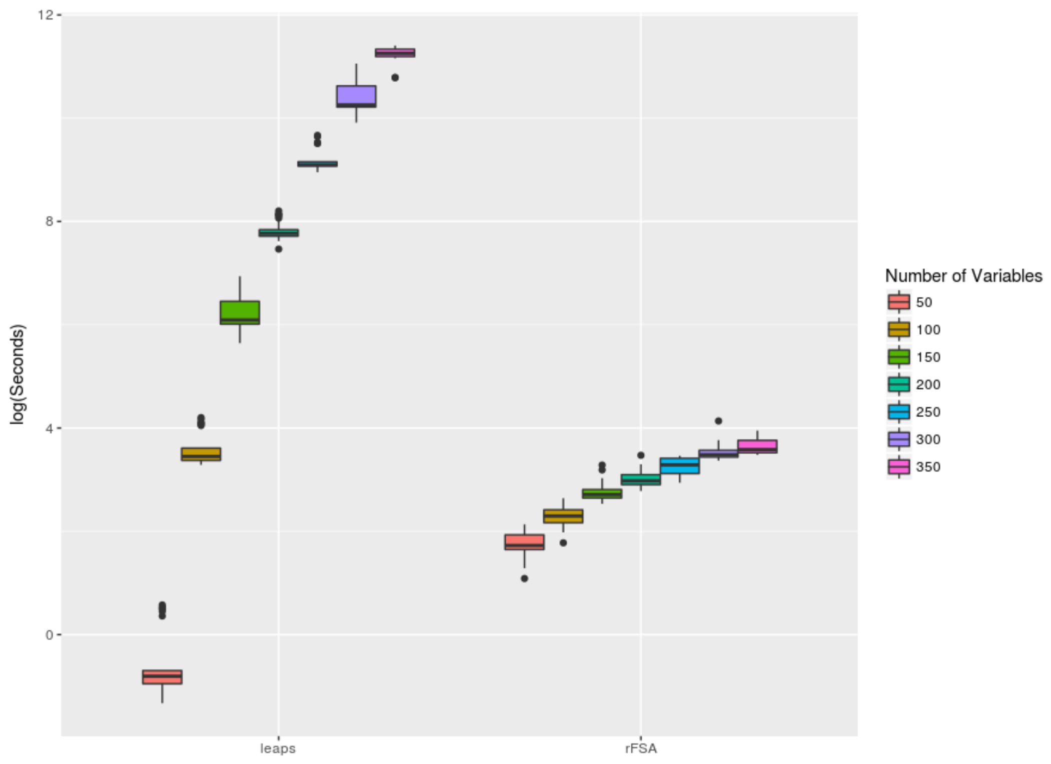


Figure 4: Timing comparison of exhaustive search with **leaps** *versus* **rFSA** with one random start. Each package was assigned to identify best subsets of size five from datasets of size $N = 250$ and various p.

time for $p = 50$, while **rFSA** exhibits a superior run-time for all other datasets ($p > 50$). The largest difference in time between `leaps::regsubsets` and `rFSA::FSA` for best subsets of size five and a fixed p was 89, 815 seconds (or 24.9 hours) for $p = 350$. Figures 3 and 4 both illustrate that the growth in run-time of our implemented FSA algorithm is much slower than **leaps**, which confirms the better time complexity of **rFSA**. As a result, our implemented algorithm is able to analyze datasets with large p at a much faster rate than the conventional alternative package.

Across the 175 simulations for best subsets of size three, **rFSA** identified the optimal solution in 158 of the simulations. When searching for best subsets of size five, **rFSA** discovered the optimal solution in 144 simulations and afforded gains in run-time of up to 24 hours. Although **rFSA** does not implement an exhaustive search, execution with many random starts often requires fewer computations while yet producing the optimal solution with arbitrarily high probability (Hawkins, 1994). Thus for large p, we argue that **rFSA** is a practical solution for researchers who wish to consider high-dimensional data, higher-order terms, generalized linear or mixed models, or other non-traditional statistical methods and criterion functions.

Example

Two- and three-way interactions in Census Data

The 2014 Planning Database Block Group Data (PDB) from the Census Bureau is publicly available at `http://www.census.gov/research/data/planning_database/2014/`. Descriptions of the variables can be found from PDB documentation on the aforementioned website. Herein we examine only the census blocks associated with Kentucky, with several variables removed because they were transformations of other variables. The final dataset contained 3285 observations and 67 quantitative explanatory variables in addition to the quantitative response variable, mail response rate. The revised dataset can be downloaded from `https://github.com/joshuawlambert/data/raw/master/census_data_nopct.csv`.

In this example, we search for the best linear model containing (a) two main effects and their interaction, or (b) three main effects, as well as their pairwise and three-way interactions. To fit linear models using the `FSA` function, we set `fitfunc` equal to `lm`. For simplicity, we choose not to fix any variables in the models, which we achieve by setting `fixvar` equal to `NULL`. We provide a null model regressing the response variable y (Mail Response Rate) solely on an intercept term, thereby restricting **rFSA** to use y as the response variable throughout the procedure. The criterion function is taken to be `int.p.val` (interaction p-value) and minimized on each iteration. To specify a desire for two-way interactions, we set parameters `interactions = TRUE` and `m = 2`; for three-way interactions, `interactions = TRUE` and `m = 3`. We request 50 random starts, which **rFSA** completed in approximately one minute for `m = 2` or five minutes for `m = 3` on a Windows 7 machine with Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz and 24.00 GB of memory.

R Code follows for reproduction of these results:

```
install.packages("rFSA")
library(rFSA)
download.file("http://raw.githubusercontent.com/joshuawlambert/data/master/
               census_data_nopct.csv",destfile = "tmp.csv")
census_data_nopct <- read.csv(file = "tmp.csv")

# find two-way interactions
set.seed(123)
fit_2_way <- FSA(formula = "y~1", data = census_data_nopct, fitfunc = lm,
                 fixvar = NULL, quad = F, m = 2, numrs = 50, cores = 1,
                 interactions = T, criterion = int.p.val, minmax = "min")

print(fit_2_way)    # summary of solutions found
summary(fit_2_way)  # list of summaries from each lm fit
plot(fit_2_way)     # diagnostic plots


# find three-way interactions
set.seed(1234)
fit_3_way <- FSA(formula = "y~1", data = census_data_nopct, fitfunc = lm,
                 fixvar = NULL, quad = F, m = 3, numrs = 50, cores = 1,
                  interactions = T, criterion = int.p.val, minmax = "min")

print(fit_3_way)    # summary of solutions found
summary(fit_3_way)  # list of summaries from each lm fit
plot(fit_3_way)     # diagnostic plots
```

Two-way interactions

Across 50 random starts seeking two-way interactions, **rFSA** discovered three unique feasible solutions. The solution with the smallest two-way interaction p-value occurred 24 times, ranking second in prevalence. The three interactions suggest relationships between (a) number of households with both spouses as members, and number of persons aged 18 to 24 ($p = 6.19 \times 10^{-38}$); (b) average household income and average aggregate house value in dollars ($p = 1.68 \times 10^{-43}$); and (c) number of mobile homes and total vacant housing units ($p = 7.26 \times 10^{-19}$). All three interaction p-values are significant using a Bonferroni-adjusted cutoff of $\frac{0.05}{\binom{67}{2}} \approx 0.00045$.

Having identified a set of feasible solutions, an investigator generally wishes to examine a summary of each model fit. Given an object, `fit`, returned by `FSA`, the function call `summary(fit)` returns a list of summaries for each model fit (including the original). Assessing the fit of each feasible solution can be facilitated with `plot(fit)` to display diagnostic plots for the original and feasible models. Each solution should be considered in a practical manner, with reasonable interpretations of the interaction term being considered prior to its inclusion in a final model.

The three interactions listed above are informative for understanding the types of results returned by the algorithm. For example, one might expect that a larger average household income would be correlated positively with the average value of houses in a census block. Thus, as in any linear regression setting, multicollinearity may account for apparent relationships in the data. The second interaction, between number of households with both spouses as members and the number of persons aged 18 to 24, may be more meaningful in the description of mail response rate (y). Because this interaction is both justifiable and statistically significant, an investigator would have considerable evidence to warrant its inclusion in a final model.

Three-way interactions

Across 50 random starts seeking three-way interactions, **rFSA** identified 14 feasible solutions. The most-frequently observed feasible solution occurred 13 times and boasted the smallest three-way interaction p-value. This interaction contains two variables identified previously in our search for two-way interactions. The three-way interaction among the number of households in which both spouses are members, the number of persons aged 18 to 24, and the number of people who indicate no Hispanic origin and their sole race as "White," affords an interaction p-value of 7.62×10^{-24} and remains significant under a Bonferroni correction. This interaction suggests that the combined effect of married couples and young adults is further modified by the number of people who identify as "White." That is, in locations with a larger population of whites, the effect on mail response rate of having more married couples changes depending on the prevalence of young adults aged 18 and 24. The interaction is reasonably easy both to interpret and to justify, as well as being statistically significant, and thus would warrant additional investigation by an interested researcher.

In sum, leveraging **rFSA** can highlight associations and interactions underlying a dataset that may not be immediately apparent to the researcher. The package is easy to manipulate (as demonstrated by the sparseness of the code provided in this example) as well as highly efficient, and generally returns multiple subsets of variables to permit flexible exploration and validation.

Conclusion

In this paper, we discuss the implementation in R of a complex algorithm originally proposed by Miller (1984) and Hawkins (1994). We further demonstrate its versatility and computational efficiency by comparison with existing subset-selection packages and by execution on a real-life census dataset. Our package, **rFSA**, is available on CRAN and GitHub. Additionally, a light version of the algorithm is available as a Shiny Application for the convenience of users with limited familiarity with R.

rFSA boasts several improvements over existing R packages for finding best subsets, some of the most popular of which include **leaps**, **bestglm**, and **glmulti**. Although these packages all perform well in implementing exhaustive searches without interactions, they offer a limited choice of statistical technique, and often struggle when confronted with high-dimensional datasets. By contrast, **rFSA** accommodates large datasets, higher-order interaction terms, and a variety of model forms and criterion functions, and can be adapted to work with less-traditional statistical methods such as survey-weighted, penalized, hierarchical, zero-inflated, survival, and many other regression techniques. The results permit facile interpretation and remain flexible to conventional modes of statistical inference. Finally, the algorithm exhibits a considerable speedup on single-core operations for large subsets, and large p, relative to variable-selection packages in common use.

To improve the accessibility and convenience of the FSA for use by the general research community, we plan to incorporate more features of **rFSA** into the existing Shiny Application. Moreover, we intend to improve the data-visualization features of both App and package, and to build an off-line version of the Shiny App for users with secure data. We will seek further improvements to execution speed and resource management to encourage analysis of yet-larger datasets, and incorporate additional model forms and criterion functions to permit efficient analysis by non-traditional methods.

Through its versatility and flexibility, **rFSA** provides an alternative algorithm for model selection that allows users to find statistically optimal subsets and interaction effects in a variety of datasets. Improved model selection may afford better predictive power, which can in turn illuminate relationships underlying large datasets in a variety of research fields.

Summary

This paper outlines an R package, named **rFSA**, for subset selection and discovery of higher-order interactions. **rFSA** is flexible to a variety of statistical models and criterion functions, including those implemented by the user, and boasts execution speeds superior to existing subset-selection packages in context of large datasets. The release version of **rFSA** is hosted on CRAN, and the development version can be accessed from GitHub by calling `devtools::install_github("joshuawlambert/rFSA")`.

Acknowledgements

This research and package creation were supported in part by the Kentucky Biomedical Research Infrastructure Network and INBRE Grant (P20 RR16481) and a National Multiple Sclerosis Society Pilot Grant (PP-1609-25975).

Bibliography

A. Bhattacharyya. On a measure of divergence between two multinomial populations. *Indian Statistical Institute*, 7, 1946. [p301]

J. Bien and R. Tibshirani. *hierNet: A Lasso for Hierarchical Interactions*, 2014. URL https://CRAN.R-project.org/package=hierNet. R package version 1.6. [p297]

J. Bien, J. Taylor, and R. Tibshirani. A lasso for hierarchical interactions. *The Annals of Statistics*, 41, 2013. URL https://doi.org/10.1214/13-aos1096. [p297]

V. Calcagno. **glmulti***: Model Selection and Multimodel Inference Made Easy*, 2013. URL https://cran.r-project.org/web/packages/glmulti/. R package version 1.0.7. [p296, 302]

D. Foster and R. Stine. Variable selection in data mining: Building a predictive model for bankruptcy. *Journal of the American Statistical Association*, 99:303–313, 2004. URL https://doi.org/10.1198/016214504000000287. [p295]

J. Friedman. **glmnet***:Lasso and Elastic-Net Regularized Generalized Linear Models*, 2008. URL https://cran.r-project.org/web/packages/glmnet/. R package version 2.0-5. [p296]

B. Goudey. High performance computing enabling exhaustive analysis of higher order single nucleotide polymorphism interaction in genome wide association studies. *Health Information Science and Systems*, 3, 2015. URL https://doi.org/10.1186/2047-2501-3-s1-s3. [p295, 296]

D. Hawkins. The feasible solution algorithm for least trimmed squares regression. *Computational Statistics & Data Analysis*, 17:185–196, 1994. URL https://doi.org/10.1016/0167-9473(92)00070-8. [p295, 296, 304, 305]

S. Højsgaard, U. Halekoh, and J. Yan. The r package geepack for generalized estimating equations. *Journal of Statistical Software*, 15/2:1–11, 2006. URL https://doi.org/10.18637/jss.v015.i02. [p301]

S. Janse. Inference using bhattacharyya distance to model interaction effects when the number of predictors far exceeds the sample size. *Theses and Dissertations–University of Kentucky Statistics*, 2017. URL https://doi.org/10.13023/etd.2017.455. [p301]

R. Jiang. A random forest approach to the detection of epistatic interactions in case-control studies. *BMC Bioinformatics*, 10, 2009. URL https://doi.org/10.1186/1471-2105-10-s1-s65. [p297]

E. Lampa. The identification of complex interactions in epidemiology and toxicology: a simulation study of boosted regression trees. *Environmental Health*, 13, 2014. URL https://doi.org/10.1186/1476-069x-13-57. [p297]

T. Lumley and A. Miller. **leaps**: *Regression Subset Selection*, 2009. URL https://cran.r-project.org/web/packages/leaps/. R package version 2.9. [p296, 302]

A. I. McLeod and C. Xu. *Bestglm: Best Subset GLM*, 2017. URL https://CRAN.R-project.org/package=bestglm. R package version 0.36. [p302]

A. Miller. Selection of subsets of regression variables. *Journal of the Royal Statistical Society*, 147:389–425, 1984. URL https://doi.org/10.2307/2981576. [p295, 296, 302, 305]

W. Pan. Akaike's information criterion in generalized estimating equations. *Biometrics*, 57:120–125, 2001. URL https://doi.org/10.1111/j.0006-341x.2001.00120.x. [p301]

N. Russell. *Hashmap: The Faster Hash Map*, 2017. URL https://CRAN.R-project.org/package=hashmap. R package version 0.2.0. [p300]

J. Yan. Geepack: Yet another package for generalized estimating equations. *R-News*, 2/3:12–14, 2002. [p301]

J. Yan and J. P. Fine. Estimating equations for association structures. *Statistics in Medicine*, 23:859–880, 2004. URL https://doi.org/10.1002/sim.1650. [p301]

Y. Zhang and J. Liu. Bayesian inference of epistatic interactions in case-control studies. *Nature Genetics*, 39, 2007. URL https://doi.org/10.1038/ng2110. [p297]

Dr. Joshua Lambert
College of Nursing
Assistant Professor & Biostatistician
214 Procter Hall
University of Cincinnati
Cincinnati, OH, USA
Joshua.Lambert@uc.edu

Dr. Liyu Gong
Post-Doctoral Scholar
Institute for Biomedical Informatics
University of Kentucky
Lexington, KY, USA
liyu.gong@uky.edu

Corrine F. Elliott, M.S.
Department of Statistics
University of Kentucky
Lexington, KY, USA
ORCiD 0000-0001-7935-9945
corrine.elliott@uky.edu

Dr. Katherine Thompson
Department of Statistics
Assistant Professor
317 Multidisciplinary Science Building
University of Kentucky

Lexington, KY, USA
katherine.thompson@uky.edu

Dr. Arnold Stromberg
Department of Statistics
Department Chair and Professor
313 Multidisciplinary Science Building
University of Kentucky
Lexington, KY, USA
stromberg@uky.edu

Dot-Pipe: an S3 Extensible Pipe for R

by John Mount and Nina Zumel

Abstract Pipe notation is popular with a large league of R users, with **magrittr** being the dominant realization. However, this should not be enough to consider piping in R as a settled topic that is not subject to further discussion, experimentation, or possibility for improvement. To promote innovation opportunities, we describe the **wrapr** R package and "dot-pipe" notation, a well behaved sequencing operator with S3 extensibility. We include a number of examples of using this pipe to interact with and extend other R packages.

Introduction

Using pipes to sequence operations has a number of advantages. Piping is analogous to representing function composition as a left to right flow of values, which is a natural direction for Western readers, and is much more legible than composition represented as nesting.

Pipe notation is a popular topic in the R community. Related work includes:

- **data.table**: Dowle and Srinivasan (2017) use the open and closed square bracket in **data.table** `][`, which is essentially as an example of *piping* or *method chaining*.
- **magrittr**: Bache and Wickham (2014) popularized pipe used in **dplyr** (Wickham et al., 2017).
- **future**: Bengtsson (2017) offers a powerful distributed processing package with pipe notation.
- **rmonad**: Arendsee (2017) created a monadic operator package, capturing exceptions in addition to managing composition and values.
- **pipeR**: Ren (2016) contains a collection of sequencing methods including pipes and method chaining.
- **backpipe**: Brown (2016) introduces a right to left pipe operator.
- **drake**: Landau (2018) contains a work-flow/graph toolkit for reproducible code and high-performance computing.

This article will discuss using the operator `%.>%` from the package **wrapr** (Mount and Zumel, 2018) (colloquially called "dot-pipe" or "dot-arrow"). Dot-pipe is compatible with many other meta-programming paradigms, and is directly S3 extensible.

Pipe notations

There are a number of important pipe notations in and out of R:

- In mathematical function composition or application, one can write `a ∘ b` to denote `a(b)`.
- In Unix, `process1 | process2` streams results from `process1` as input to `process2`.
- In APL's reduce/apply slash notations.
- In F#'s forward pipe operator, `a |> b` means `b a`, using F#'s partial application feature.
- In **magrittr** pipes, `a %>% b(...)` is most commonly used to denote `{. <- a; b(., ...)}` (with dot side effects hidden).
- With the dot-pipe `%.>%` (the topic of this article), where `a %.>% b` is intended to approximately mean `{. <- a; b}`.

Using `%.>%` to sequence operations

In this section, we demonstrate the use of "dot-pipe" `%.>%` from **wrapr** and some of its merits. The intended semantics of `%.>%` are:

`a %.>% b` is nearly equivalent to `{. <- a; b}`

where `a` and `b` are taken to be R expressions, presumably with the dot occurring as a unbound (or free) symbol in `b`. For example:

```
>   library("wrapr")
>   5 %.>% sin(.)
[1] -0.9589243

>   print(.)
[1] 5
```

Notice the **wrapr** dot-pipe leaves the most recent left-hand side value in the variable named dot. While this is a visible side-effect of the pipe which can conflict with other uses of dot, we feel these explicit semantics are sensible, easy to teach, and easy to work with.

We can also write `5 %.>% sin`, as the dot-pipe looks up functions by name (even for qualified names such as `base::sin`) as a user convenience. This function lookup is a non referentially transparent special case, as names are deliberately treated differently than values. However, it is an important capability that we will discuss later and greatly expand using S3 object oriented dispatch. Dot-pipe's default service does not work with the expression `5 %.>% sin()` and throws an informative error message. Maintaining an explicit distinction between `sin` (a name), `sin()` (an expression with no free-use of dot), and `sin(.)` (an expression with free-use of dot) has benefits, some of which we will demonstrate in the Extending the sequencer section. For dot-pipe usage in general, the explicit expression `sin(.)` is preferred to `sin` under the rubric "dot-pipe has lots of dots."

Additional dot-pipe examples include:

```
>   5 %.>% {1 + .}
[1] 6

>   5 %.>% (1 + .)
[1] 6
```

Notice dot-pipe treated the last two statements similarly. We warn the reader that in R the expression `5 %.>% 1 + .` is read as `(5 %.>% 1) + .`, since special operators (those using `%`) have higher operator precedence than binary arithmetic operators.

The dot-pipe works well with many packages, including **dplyr**:[1]

```
>   library("dplyr")
>   disp <- 4
>   mtcars %.>%
+       filter(., .data$cyl == .env$disp) %.>%
+       nrow(.)
[1] 11
```

Extending the sequencer

Dot-pipe's primary dispatch is user extensible; by default, it treats `a %.>% b` as `{. <- a; b}`. However, it does this via S3 dispatch through a method of signature `apply_left(a,b,--more--)`. User or package code can override this method to add custom effects. For example, one can extend dot-pipe to be a **ggplot2** Wickham et al. (2018) layer compositor as we show below.

```
>  library("ggplot2")
>  apply_left.gg <- function(pipe_left_arg,
+                            pipe_right_arg,
+                            pipe_environment,
+                            left_arg_name,
+                            pipe_string,
+                            right_arg_name) {
+   pipe_right_arg <- eval(pipe_right_arg,
+                          envir = pipe_environment,
+                          enclos = pipe_environment)
+   pipe_left_arg + pipe_right_arg
+  }
```

We have defined an implementation of `apply_left.gg`, as this is the class used by **ggplot2** to recognize its own objects (i.e., **ggplot2** works by defining `` `+`.gg``). Essentially `apply_left.gg(a,b)` is implemented as `a + b`, the only detail being `b` is passed as a un-evaluated language argument,

[1]Example adapted from https://github.com/tidyverse/dplyr/issues/3286.

so it must be evaluated before being used as a regular value, a detail discussed in the package documentation.

We can now easily write a pipeline that combines sequencing **dplyr** transformation steps and combining **ggplot2** geom objects, producing figure 1.

```
>  data.frame(x = 1:20) %.>%
+     mutate(., y = cos(3*x)) %.>%
+     ggplot(., aes(x = x,  y = y)) %.>%
+     geom_point() %.>%
+     geom_line() %.>%
+     ggtitle("piped ggplot2",
+             subtitle = "wrapr")
```

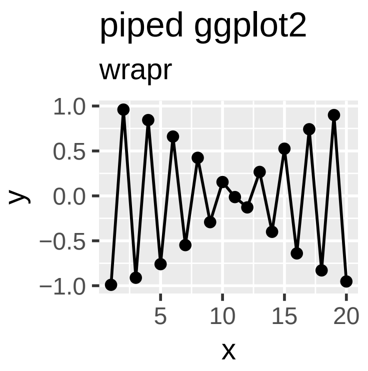


Figure 1: Example plot produced by a pipeline.

Notice how we can use the same pipe notation for both the initial **dplyr** data processing steps and for the later **ggplot2** layer aggregation steps. As before, the data processing steps (e.g., `mutate`) require dot as a free symbol to specify where the piped values go. However, the **ggplot2** steps do not use such a dot argument, as these functions do not expect previous steps as arguments.

Dot-pipe was able to add capabilities to the **ggplot2** package without requiring any changes to the **ggplot2** package. This extension capability is important.

Treating names as functions

If an object on the right hand side of a dot-pipe stage is an R language name (or a qualified name such as `base::sin`), then that object is retrieved. If the result is a function, the function is applied. If the result is a more general object, then S3 dispatch is used on the class of this *second* or right hand side argument. That is, `a %.>% b` is treated as `b(a)` or $\texttt{f}_{\texttt{class(b)}}$`(a,b)`.

A good example using this capability is extending the **rquery** package (Mount, 2018) to allow relational operator trees to be used both as inspectable objects and as functions that can be applied directly to data. In the following example, we create an operator tree that adds the column `y` to a data frame, `d`.

```
>   library("rquery")
>   optree <- mk_td(table_name = "d", columns = "x") %.>%
+      extend_nse(., y = cos(2*x))
```

We can treat `optree` as an object as we show below.

```
>   class(optree)
[1] "relop_extend" "relop"

>   print(optree)
[1] "table(d; x) %.>% extend(., y := cos(2 * x))"

>   column_names(optree)
[1] "x" "y"
```

```
>   columns_used(optree)
$d
[1] "x"
```

Or we can pipe into it, as we now demonstrate.

```
>   # get a database connection
>   db = DBI::dbConnect(RSQLite::SQLite(),
+                       ":memory:")
>   # make our db connection available to rquery package
>   options(list("rquery.rquery_db_executor" = list(db = db)))
>   data.frame(x = 1:3) %.>% optree # apply optree to d
  x          y
1 1 -0.4161468
2 2 -0.6536436
3 3  0.9601703
```

In this example, the **rquery** package defined a surrogate S3 method for the right hand side pipe argument: `apply_right.relop`. Any user or package can extend the dot-pipe to suit their needs, just as we have shown here. The **rquery** package defines `apply_right.relop` allowing new data to be applied to existing pipelines as we saw above.

We provide `wrapr::apply_right_S4()` as an S4 dispatch interface. This flexibility can be used to define special effects such as "same class to same class" ideas. For example, we can arrange for data frames to automatically call `rbind` when piped into each other. Note that it usually does not make sense to pipe into a non-expression or non-function object.

```
>  d1 <- data.frame(x = 1)
>  d2 <- data.frame(x = 2)
>  tryCatch(
+     d1 %.>% d2,
+     error = function(e) { invisible(cat(format(e))) })
wrapr::apply_right_S4 default called with classes:
 d1 data.frame
 d2 data.frame
  must have a more specific S4 method defined to dispatch
 NULL
```

If one sets a generic signature for `apply_right_S4` this can be made a sensible and useful operation.

```
>  setMethod(
+   "apply_right_S4",
+   signature = c("data.frame", "data.frame"),
+   definition = function(pipe_left_arg,
+                         pipe_right_arg,
+                         pipe_environment,
+                         left_arg_name,
+                         pipe_string,
+                         right_arg_name) {
+     rbind(pipe_left_arg, pipe_right_arg)
+   })
>  d1 %.>% d2
  x
1 1
2 2
```

However, the `apply_right` execution path is only active when the right pipe argument is a name. The `rbind` effect would not work if piped directly into a value. The default `apply_right` implementation is an S3 dispatch on the *right* pipe argument.

```
>  d1 %.>% data.frame(x = 2)
  x
1 2
```

In this case `data.frame(x = 2)` was evaluated an an expression where the dot had the value `data.frame(x = 1)`, which was in turn ignored.

Dot-pipe semantics

We have been describing dot-pipe semantics by introducing transformed code that we consider equivalent to the dot-pipe pipeline. Think of that as the specification. Dot-pipe's implementation is not by code substitution but through execution rules we outline here.

In R, special operators (those written with `%`) are left to right associative (meaning `a %.>% b %.>% c` is taken to mean `(a %.>% b) %.>% c`) with fairly high operator precedence (meaning they are applied earlier than some other operators).

The dot-pipe semantics are realized by the following processing rules. `a %.>% b` is processed as follows:

Def. We choose the default "control on the left case" (or "L case") if the second or right hand side argument `b` is not a R language name or other dereferencable entity, otherwise we take the "control on right case" (or "R case"). We then continue with one of these two cases.

L case. S3 dispatch is performed on `apply_left(a,b,env,nm)`, with `class(a)` being the method-determining argument, and `b` an un-evaluated R language object. The default implementation of `apply_left(a,b,env,nm)` is `. <-a ; eval(b)` (performed in the calling environment).

R case. We look-up the second or right hand side argument `b` and then branch as follows.

i. If `b` is now a function, the value `b(a)` is returned.
ii. Otherwise, S3 dispatch is performed on `apply_right(a,b,env,nm)` with `class(b)` as the method determining argument. If `b` is an S4 object, `apply_right.default(a,b,env,nm)` in turn dispatches to `apply_right_S4(a,b,env,nm)`.

This may seem involved, but it is in fact quite regular with only one exception: a dereference triggers right-dispatch. Roughly, the rule is: "treat the second or right hand side argument as an expression, unless it is a name." The intent is for dot-pipe to have simple semantics that are capable of being combined many ways to allow rich emergent behavior.

Comparison with magrittr

The **magrittr** package supplies a very popular R pipe operators, so it is worth a bit of discussion.

The **magrittr** package works by capturing entire (possibly more than one step) pipelines un-evaluated and then inspecting the captured code for its own piping symbols. This can be confirmed by looking at the implementation and also by attempting to re-name the **magrittr** pipe.

```
>  library("magrittr")
>  5 %>% sin
[1] -0.9589243

>  `%userpipe%` <- magrittr::`%>%`
>  tryCatch(
+     5 %userpipe% sin,
+     error = function(e) {e})
<simpleError in pipes[[i]]: subscript out of bounds>
```

The **wrapr** pipe executes by looking only at the arguments it is given, holding the right argument un-evaluated until the left value is available. Multiple stage **wrapr** pipes are just an effect of running stages one after the other.

```
>  `%userpipe%` <- wrapr::`%.>%`
>   5 %userpipe% sin
[1] -0.9589243
```

There are also differences in how **magittr** and **wrapr** handle functions and function arguments. With **magittr**, one can not reliably pipe into `substitute` with `%>%`. Note that the word `value` below is the result, not an input.

```
>  library("magrittr")
>  5 %>% substitute
value
```

Also, `%>%` does not work with qualified names unless one uses the more general expression notation `base::sin(.)`.

```
> tryCatch(
+    5 %>% base::sin,
+    error = function(e) {e})
<simpleError in .::base: unused argument (sin)>
```

In contrast, `%.>%` behaves closer to common user expectations.

```
> library("wrapr")
> 5 %.>% substitute
[1] 5

> 5 %.>% base::sin
[1] -0.9589243
```

Note that the **magrittr** package can in fact trigger S3 dispatch through its exposition pipe operator, `%$%` pipe, but this is only because this operator calls the method `with()`, which is itself S3 overridable. This should not be treated as an intended feature of the **magrittr**.

Example applications

Both R users and package developers can achieve a great number of useful effects by adding S3 implementations for `apply_left()` or for `apply_right()`. Some possibilities include:

- Enabling `%.>%` as a layering function for **ggplot2** (as a replacement for +, as we demonstrated).
- Enabling auto-application of **rquery** operation trees to data frames (as we demonstrated).
- Enabling auto-application of models by mapping `apply_right.model_class` to the appropriate `predict` method.

  ```
  > d <- data.frame(x = 1:5, y = c(1, 1, 0, 1, 0))
  > model <- glm(y~x, family = binomial, data = d)
  > apply_right.glm <-
  +    function(pipe_left_arg,
  +             pipe_right_arg,
  +             pipe_environment,
  +             left_arg_name,
  +             pipe_string,
  +             right_arg_name) {
  +    predict(pipe_right_arg,
  +            newdata = pipe_left_arg,
  +            type = 'response')
  + }
  > data.frame(x = c(1, 3)) %.>% model
          1         2
  0.9428669 0.6508301
  ```

  Notice we can pipe new data directly into the model for prediction. The S3 `apply_right` extensions give us a good opportunity to regularize model predictions functions to take the same arguments and have the desired default behaviors.

- Enabling pipe notation for SQL.

  ```
  > # get a database connection
  > db = DBI::dbConnect(RSQLite::SQLite(),
  +                     ":memory:")
  > apply_right.SQLiteConnection <-
  +    function(pipe_left_arg,
  +             pipe_right_arg,
  +             pipe_environment,
  +             left_arg_name,
  +             pipe_string,
  +             right_arg_name) {
  +   DBI::dbGetQuery(pipe_right_arg, pipe_left_arg)
  + }
  > "SELECT * FROM sqlite_temp_master" %.>% db
  [1] type     name     tbl_name rootpage sql
  <0 rows> (or 0-length row.names)
  ```

Here we piped SQL code directly into the database connection.

- A string concatenation operator.

```
>  apply_left.character <- function(pipe_left_arg,
+                                  pipe_right_arg,
+                                  pipe_environment,
+                                  left_arg_name,
+                                  pipe_string,
+                                  right_arg_name) {
+     pipe_right_arg <- eval(pipe_right_arg,
+                           envir = pipe_environment,
+                           enclos = pipe_environment)
+     paste0(pipe_left_arg, pipe_right_arg)
+   }
>  "a" %.>% "b" %.>% "c"
[1] "abc"
```

  One can, of course, define a string concatenation operator directly, but this is a good example of the use of the dot-pipe as a sort of compound constructor.

- A formula term collector.

```
>  apply_left.formula <- function(pipe_left_arg,
+                                pipe_right_arg,
+                                pipe_environment,
+                                left_arg_name,
+                                pipe_string,
+                                right_arg_name) {
+     pipe_right_arg <- eval(pipe_right_arg,
+                           envir = pipe_environment,
+                           enclos = pipe_environment)
+     pipe_right_arg <- paste(pipe_right_arg, collapse = " + ")
+     update(pipe_left_arg, paste(" ~ . +", pipe_right_arg))
+  }
>  (y~a) %.>% c("b", "c", "d") %.>% "e"
y ~ a + b + c + d + e
```

We anticipate motivated package authors can find many special cases that the dot-pipe can streamline for their users. The value will be when many packages add effects on the same pipe, so users know by using that pipe they will simultaneously have many powerful features made available.

We have found it profitable to roughly think of `apply_left()` as a "programmable comma" and `apply_right()` as "automatic execution" (usually achieved by overriding `print()`).

Limitations

There are limitations to the class-driven pipe dispatch approach. The class of the left item (driving `apply_left()`) is often uninformative, as in R it will very often be a data frame. The class of the right item (used by `apply_right()`) is not available until the right item has been evaluated, which is too late for the most common pipe effect (evaluating the right item with the left available as a dot). However, the authors feel this system is more R like as it leaves more of the execution to the R interpreter and tries to minimize the pipe operator itself being a type of replacement interpreter implementation.

Conclusion

We have demonstrated a predictable, well-behaved, S3-extensible tool for sequencing or pipe-lining operations in R. The left-dispatch of `apply_left()` method is useful in assembling composite structures such as building a **ggplot2** plot up from pieces. The right-dispatch `apply_right()` is unusual, but a natural extension of the "pipes write functions on the right" idea. The goal of dot-pipe is to supply simple semantics that can be composed into powerful specific applications. The dot-pipe can be used to extend packages, or to add user desired effects. We would like the **wrapr** dot-pipe to be a testing ground both for pipe-aware package extensions and for experimenting with the nature of piping in R itself.

Acknowledgments

The authors would like to thank the **wrapr** users for their feedback. In particular we would like to thank the R Journal editors and reviewers who contributed a number of important points, including the idea that S4 evaluation was a good possibility and the natural way to discuss dispatching on a right argument.

Bibliography

Z. Arendsee. *rmonad: A Monadic Pipeline System*, 2017. URL `https://CRAN.R-project.org/package=rmonad`. R package version 0.4.0. [p309]

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2014. URL `https://CRAN.R-project.org/package=magrittr`. R package version 1.5. [p309]

H. Bengtsson. *future: Unified Parallel and Distributed Processing in R for Everyone*, 2017. URL `https://CRAN.R-project.org/package=future`. R package version 1.6.2. [p309]

C. Brown. *backpipe: Backward Pipe Operator*, 2016. URL `https://CRAN.R-project.org/package=backpipe`. R package version 0.1.8.1. [p309]

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2017. URL `https://CRAN.R-project.org/package=data.table`. R package version 1.10.4-3. [p309]

W. M. Landau. *drake: Data Frames in R for Make*, 2018. URL `https://CRAN.R-project.org/package=drake`. R package version 5.0.0. [p309]

J. Mount. *rquery: Relational Query Generator for Data Manipulation*, 2018. URL `https://CRAN.R-project.org/package=rquery`. R package version 0.3.0. [p311]

J. Mount and N. Zumel. *wrapr: Wrap R Functions for Debugging and Parametric Programming*, 2018. URL `https://CRAN.R-project.org/package=wrapr`. [p309]

K. Ren. *pipeR: Multi-Paradigm Pipeline Implementation*, 2016. URL `https://CRAN.R-project.org/package=pipeR`. R package version 0.6.1.3. [p309]

H. Wickham, R. Francois, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2017. URL `https://CRAN.R-project.org/package=dplyr`. R package version 0.7.4. [p309]

H. Wickham, W. Chang, L. Henry, T. L. Pedersen, K. Takahashi, C. Wilke, and K. Woo. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2018. URL `https://CRAN.R-project.org/package=ggplot2`. R package version 3.0.0. [p310]

John Mount
Win-Vector LLC
552 Melrose Ave., San Francisco CA, 94127
USA
`jmount@win-vector.com`

Nina Zumel
Win-Vector LLC
552 Melrose Ave., San Francisco CA, 94127
USA
`nzumel@win-vector.com`

idmTPreg: Regression Model for Progressive Illness Death Data

by Leyla Azarang and Manuel Oviedo de la Fuente

Abstract The progressive illness-death model is frequently used in medical applications. For example, the model may be used to describe the disease process in cancer studies. We have developed a new R package called **idmTPreg** to estimate regression coefficients in datasets that can be described by the progressive illness-death model. The motivation for the development of the package is a recent contribution that enables the estimation of possibly time-varying covariate effects on the transition probabilities for a progressive illness-death data. The main feature of the package is that it befits both non-Markov and Markov progressive illness-death data. The package implements the introduced estimators obtained using a direct binomial regression approach. Also, variance estimates and confidence bands are implemented in the package. This article presents guidelines for the use of the package.

Introduction

In a classical survival study, patients start from an initial state "alive" and are followed up until the end of the study. They make a transition to the absorbing state "dead", unless they drop out of the study. In many medical studies, state "alive" includes two or more transient states. For example, in the illness trajectory of a cancer, a particular stage of the illness as a transient state is usually observed over time (e.g. cancer recurrence). Multi-state models are particularly useful for modeling the overall process of survival Hougaard (2000), Andersen and Keiding (2002). A multi-state model is a model for a continuous-time stochastic process allowing individuals to move among a finite number of states. A transition from one state to another one is the occurrence of one event of interest. The progressive illness-death model depicted in Figure 1 is a three-state multi-state model used in the medical literature to describe disease progression (Meira-Machado et al., 2008). It consists of three states: "Healthy" (state 1), "Diseased" (state 2) and "Dead" (state 3). All patients start in healthy state, then each patient can either have a transition directly to the dead state or they can be diseased before moving to the dead state. That means, the trajectory for a patient will be $1 \longrightarrow 3$ or $1 \longrightarrow 2 \longrightarrow 3$, but the entire trajectory might not be observed due to censoring.

Often, in the presence of censoring, estimating the effect of a set of prognostic factors on the course of the disease is an important target for the progressive illness-death model. The effect of prognostic factors for the Markov progressive illness-death model is generally modeled by Aalen´s additive model, Aalen et al. (2001). According to their model, the effects on transition intensities are estimated and, from them, the effects on transition probabilities (therefore, the conditional transition probabilities) can be calculated by solving the so-called forward Kolmogorov differential equation. But the violation of Markov condition does not allow such a calculation. To this end, Meira-Machado´s approach (Meira-Machado et al., 2014) based on kernel smoothing is an alternative to the Aalen model. The method considers the nonparametric estimation of conditional transition probabilities in a non-Markov illness-death Model that allows only conditioning on continuous covariates in low dimension. Azarang et al. (2017) proposed the direct binomial regression method for the transition probabilities in the right-censored progressive illness-death model. By applying this method, one can estimate the possibly time-varying regression coefficients as covariate effects on the transition probabilities. The method does not require the Markov assumption and has no restriction on the dimension of covariates. In addition, it can be applied for both continuous and categorical covariates. Based on this method, we have developed a software package in R, called **idmTPreg** (available from the Comprehensive R Archive Network at `https://cran.r-project.org/web/packages/idmTPreg/`). The progressive illness-death model illustrated in Figure 1 is the only progressive disease model supported by the **idmTPreg** package.

Furthermore, several packages to analyse multi-state survival data like the progressive illness-death data are available on the Comprehensive R Archive Network (CRAN). For example, the **mstate** de Wreede et al. (2010), provides estimation of the transition probabilities possibly depending on covariates, the implemented technique assumes proportional transition intensities. The package can be applied to right censored and left truncated multi-state data. The **msm** package Jackson (2011), in terms of covariates, can be used to obtain conditional transition probabilities in continuous-time Markov and hidden-Markov multistate models for longitudinal data. In the **p3state.msm** package Meira-Machado and Roca-Pardiñas (2011), estimates of regression parameters can be obtained by assuming that each transition may be specified by a Cox-type model. To this end, one of the three possible options for the model can be chosen; TDCM, CMM or CSMM. TDCM (time dependent

Cox model) associates a time dependent covariate with the occurrence of disease, in CMM (Cox Markov model) it is assumed that the future is independent of the past given the present state, and the CSMM model (Cox semi-Markov model) emphasizes the importance of time spent in the current state, *Survival*.

This paper describes **idmTPreg** package and its capabilities through the following sections. In the next section, we outline the methodology of the direct modeling approach in the progressive illness-death model, the detailed mathematics underlying the package have been discussed in our previous paper. Then, we describe the package in full detail, and demonstrate how to apply the functions provided by the package through the analysis of a real dataset. Ultimately, the last section gives a summary of the work.

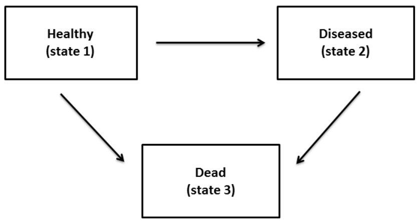


Figure 1: Progressive illness-death model.

An overview of the methodology

In this section we briefly review the methodological background behind the **idmTPreg** package. As mentioned before, the progressive illness-death model presents two transient states (state 1 and state 2) and an absorbing state (state 3). The three possible transitions are shown by forward arrows in Figure 1. We assume here that recovery (transition from 2 to 1) is not possible. Five different transition probabilities of the model are: $p_{11}(s,t)$, $p_{12}(s,t)$, $p_{13}(s,t)$, $p_{22}(s,t)$ and $p_{23}(s,t)$; s and t $(s < t)$ are times. Among the transition probabilities the following relations hold: $p_{11}(s,t) + p_{12}(s,t) + p_{13}(s,t) = 1$ and $p_{22}(s,t) + p_{23}(s,t) = 1$. The transition probabilities $p_{kl}(s,t)$´s, where $k = 1,2$; $l = 1,2,3$ and $k \le l$, are defined as:

$p_{kl}(s,t)$ = Pr(patient is at state l at time t | patient was at state k at time s.)

Likewise, in the presence of covariates ($\boldsymbol{X}$) the conditional transition probabilities denoted by $p_{kl}(s,t|X)$ are defined. Let Z be the sojourn time in state 1, T be the total survival time, and C be the censoring time of the model. The observed information is $(\tilde{Z}, \Delta_1, \tilde{T}, \Delta, \mathbf{X})$, where Δ_1 and Δ are the censoring indicator of Z and T respectively; $\tilde{Z} = min(Z,C)$ and $\tilde{T} = min(T,C)$; and $\mathbf{X}$ is the vector of time-independent covariates. In order to correct for estimation bias due to censoring, we update Δ_1 and Δ with time t and define Δ_1^t and Δ^t as follows:

$$\Delta^t = 1_{\{\min(T,t) \le C\}} = \begin{cases} \Delta & \text{if } \tilde{T} \le t \\ 1 & \text{if } \tilde{T} > t \end{cases}$$

and

$$\Delta_1^t = 1_{\{\min(Z,t) \le C\}} = \begin{cases} \Delta & \text{if } \tilde{Z} \le t \\ 1 & \text{if } \tilde{Z} > t \end{cases}$$

Azarang et al. (2017) introduced regression modeling to estimate the possibly time-varying coefficients for the conditional transition probabilities ($p_{kl}(s,t|\mathbf{X})$, $k = 1,2$; $l = 1,2,3$; $k \le l$) in the progressive illness-death model via defining binomial time-varying response variables ($Y_{kl}(t)$'s), and time-varying random weights $W_{kl}^s(t)$ (depending on transition from k to l the weights are either a function of Δ_1^t or a function of Δ^t and are estimated by $\hat{W}_{kl}^s(t)$, see Azarang et al. (2017) for more details). The response variables and weights for each transition probability are well defined so that:

$$p_{kl}(s,t|\mathbf{X}) = E_s[W_{kl}^s(t)Y_{kl}(t)|\mathbf{X}] \quad k = 1,2; \quad l = 1,2,3; \quad k \le l,$$

where E_s for $k = 1$ is the expectation value conditioned on the event of being observed at the initial state by a given time s $(Z > s)$ and for $k = 2$ it is the expectation value conditioned on being observed at the intermediate state by time s $(Z \leq s < T)$. Then for a fixed time s, the linear predictor $\boldsymbol{X}\boldsymbol{\beta}_{kl}^{(s)}(t)$ is linked to the transition probability $p_{kl}(s, t|X)$ via an allowable link function for the binomial family. Without the loss of generality, we consider logit link function:

$$p_{kl}(s, t|\boldsymbol{X}) = \frac{\exp(\boldsymbol{X}\boldsymbol{\beta}_{kl}^{(s)}(t))}{1 + \exp(\boldsymbol{X}\boldsymbol{\beta}_{kl}^{(s)}(t))}.$$

As mentioned before, we consider s to be fixed and $t \in [a, \tau]$, where t is the last event time point, and $p_{kl}(s, a|\boldsymbol{x}) > 0$. Then, $\boldsymbol{\beta}_{kl}^{(s)}(t)$, which is a vector of possibly time-varying coefficients, can be estimated by solving the following score equations:

$$\sum_{I_k} \frac{\partial p_{kl}(s, t|\mathbf{X}_i)}{\partial \boldsymbol{\beta}_{kl}^{(s)}(t)} \hat{W}_{kl}^{s}(t)[Y_{kl}(t) - p_{kl}(s, t|\mathbf{X}_i)] \equiv 0,$$

where $I_1 = \{i : \tilde{Z}_i > s\}$ and $I_2 = \{i : \tilde{Z}_i \leq s < \tilde{T}_i\}$. Note that the expected value of the above score functions equals zero. Since the estimates of the coefficients, $\hat{\boldsymbol{\beta}}_{kl}^{(s)}(t)$, are piecewise constant between the jump times of Y_{kl} (that are the jump times of W_{kl}^{s} too) we can fit the standard approach for generalised linear models at each jump time of the corresponding response variable. Therefore, for each t between a and τ, $\hat{\boldsymbol{\beta}}_{kl}^{(s)}(t)$ are given.

Package description

The **idmTPreg** package provides estimates of the coefficients on transition probabilities for a progressive illness-death dataset. The package consists of 6 user-visible functions described in Table 1. In addition, there is an invisible function called `mod.glm.fit` that is contained inside the main function to fit generalized linear type models. This function is a modified version of `glm.fit` available in the R Stats Package. The modification was done to give special weights to the binary responses discussed in the previous section. The modified function `mod.glm.fit` gives the estimated vector of coefficients. Also, the call to `mod.glm.fit` has been programmed in parallel to obtain 95% pointwise bootstrap confidence bands. The number of cores for parallel execution is set to the number of CPU cores on the current host by default unless it is specified by the user. Then, `registerDoParallel` of the **doParallel** package is used to register the parallel backend. The parallel computation is performed by the `foreach` function of **foreach** package.

The main function, intended to be called by the user, is `TPreg()`. The data frame to be passed into the main function of the package, other than covariates, must contain `Zt`, `Tt`, `delta1`, and `delta` variables (denoted by $\tilde{Z}$, $\tilde{T}$, Δ_1 and Δ in the previous section). Assessing these variables for a non-statistician might be challenging, so `iddata` function is used to convert simple records of progressive illness-death data to the proper format. To this end, one may ask clinicians to give them the total survival time (`Stime`), the indicator of uncensored total survival time (`Sind`), the arrival time to the diseased state (`Iltime`), the indicator of visiting diseased state (`Ilind`) and a vector of covariates (`cov`). By convention for patients who have not been diseased (`Ilind=0`), their arrival time to the diseased state is recorded equal to their total survival time.

Example of application

To illustrate our method with the capability of our package, we consider the colon cancer dataset which is freely available as a part of the **survival** package. In this study, from 929 patients who had curative-intent resections of stage III colon cancer, 315 were randomly assigned to observation only, 310 to levamisole alone (Lev), and 304 to levamisole plus fluorouracil (Lev+5FU). See Moertel et al. (1990) for details. By the end of the study, 477 patients remained alive, 468 developed a recurrence, and 452 died and among these, 38 died without recurrence. The possible events for a patient may be described by the progressive illness-death model with states 1, 2 and 3 corresponding to "Alive and disease-free", "Alive with recurrence", and "Dead" respectively. A subset of colon cancer dataset called `colonTPreg` is available in the **idmTPreg** package, but just three risk factors were included in the dataset : Age (in years) and Nodes (number of lymph nodes with detectable cancer), and treatment. We use the `colonTPreg` dataset to demonstrate the functionality of the package.

```
> library(idmTPreg)
> data(colonTPreg)
> head(colonTPreg)
```

Function	Description
`TPreg`	Fits the semi-parametric regression model to estimate the effects on transition probabilities for a sequence of time.
`summary`	Gives details about the estimated effects of pre-specified transition probabilities for a sequence from a given `s` to a given `t`.
`plot`	Makes a plot for the estimated effect of pre-specified transition probabilities along time, from time `s` to time `t`.
`print`	Provides the details about the estimated effects of pre-specified transition probabilities for given `s` and `t`.
`iddata`	Converts a raw illness-death data to a data frame which can be passed into `TPreg` function (described before) .

Table 1: Functions and summary of their descriptions in the **idmTPreg** package.

```
  id   Zt   Tt delta1 delta Nodes Age treatment
1  1  968 1521      1     1     5  43   Lev+5FU
2  2 3087 3087      0     0     1  63   Lev+5FU
3  3  542  963      1     1     7  71       Obs
4  4  245  293      1     1     6  66   Lev+5FU
5  5  523  659      1     1    22  69       Obs
6  6  904 1767      1     1     9  57   Lev+5FU
```

Each row in the data corresponds to a single individual. The columns `Zt`, and `Tt` are time variables, measured in days. For instance, patient 1 experienced a recurrence after 968 days (transition from state 1 to state 2), and died after 1521 days. Patient 2 was censored after 3087 days without having the recurrence. These data have a suitable format for the analysis.

Estimates for the effect of covariates on transition probabilities are obtained using function `TPreg()`. The first argument of this function is an object of class `"formula"` which specifies the covariates on the right-hand side of the $\sim$ operator, that are separated by + operators. The left side of the $\sim$ operator is left empty because the time-dependent binary responses corresponding to each transition are defined in the main function. The `data` argument must be a data frame of `"iddata"` class or a data frame similar to `colonTPreg` format described previously. The `link` argument is a suitable link function for binomial family (logit, probit and cauchit). Argument `s` is the current time for the transition probabilities; default is zero which reports the occupation probabilities. Argument `t` is the Future time for the transition probabilities; default is NULL which is the largest uncensored sojourn time in the initial state. The `R` argument specifies the number of bootstrap replicates, default is 199. The argument `trans` indicates the possible transition(s) for a progressive illness-death model. The output is an object of class `"TPreg"`. This object has its own `print()`, `summary()` and `plot()` methods.

We apply the method described in the previous section to estimate the effect of the risk factors on transition "Alive and disease-free"$\longrightarrow$"Dead". To see the result for each time of the sequence from time s = 0 to 7 years with the default increment, we use the following input command:

```
> co13 <- TPreg( ~ Age + Nodes + treatment, colonTPreg,
+               link = "logit", s = 0, R = 99,
+               t = 365.24*7, trans = "13")
> co13
```

```
Call:
TPreg(formula = ~Age + Nodes + treatment, data = colonTPreg,
link = "logit", s = 0, t = 365.24 * 7, R = 99, trans = "13")

Transition:
[1] "13"

(s,t):
[1]    0.00 2556.68

Coefficients:
                   Estimate      St.Err         LW.L        UP.L      P.value
X.Intercept.     -1.5899472 0.588179192 -2.742778368 -0.43711594 6.868203e-03
Age               0.0162786 0.008990345 -0.001342478  0.03389967 7.019108e-02
Nodes             0.2773286 0.066204572  0.147567677  0.40708960 2.802296e-05
treatmentLev     -0.2360843 0.209801808 -0.647295861  0.17512722 2.604733e-01
treatmentLev.5FU -0.6222765 0.281455569 -1.173929431 -0.07062360 2.704119e-02


[1] "18 observations deleted due to missingness from 'data'"
```

The `print()` method returns the results for `s=0` and `t= 2556.68` days; and provides 95% pointwise bootstrap confidence bands based on nonparametric resampling and normal method (normal approximation of two-sided nonparametric confidence interval). Then, using the `summary()` function one can obtain estimated values at each especified time between `s=0` and `t=2556.68`. The estimates between the jump times of Y_{13} in the time interval [s,t] are piecewise constant, and one can choose a vector of times from the jump times via argument `by`. Then, at the selected times the values are estimated. This argument gives the increment of the sequence from time `s` to time `t`. The default is $\lfloor \frac{\max(\tilde{Z})-\min(\tilde{Z})}{q_{0.01}(\tilde{Z})} \rfloor$, where $q_{0.01}(.)$ is the sample quantile corresponding to 0.01 probability and $\lfloor x \rfloor$ gives the largest integer less than or equal to x.

```
> summary(co13)

Call:
TPreg(formula = ~Age + Nodes + treatment, data = colonTPreg,
        link = "logit", s = 0, t = 365.24 * 7, R = 99, trans = "13")
(s,t):
[1]    0.00 2556.68

 Transition 13  :

   Coefficients:
    time   X.Intercept.        Age       Nodes  treatmentLev  treatmentLev.5FU
1     23.00   -28.325534 -0.024555212 -0.03918837  3.388159555      24.27533991
2    402.00    -4.092865  0.025096305  0.08050421  0.156140164      -0.07222873
3    659.00    -2.220785  0.003369437  0.16278647  0.001584098      -0.22781061
4    938.00    -2.003750  0.007666566  0.19530069 -0.009332136      -0.35912373
5   1313.00    -1.599525  0.008508928  0.20271435  0.068328191      -0.54960487
6   1891.00    -1.438688  0.009063383  0.24205416 -0.035327744      -0.48638357
7   2122.00    -1.623667  0.013412336  0.25594531 -0.124328431      -0.52617865
8   2221.00    -1.296856  0.010038380  0.25308608 -0.202232077      -0.59947602
9   2385.00    -1.488674  0.014490552  0.26120609 -0.295496733      -0.67206843
10 2556.68    -1.589947  0.016278598  0.27732864 -0.236084318      -0.62227652

  Standard Errors:
    time   X.Intercept.        Age       Nodes  treatmentLev  treatmentLev.5FU
1     23.00 6951.8911865 0.012738568 0.02106333 6951.7974079     7332.1281285
2    402.00    0.7536143 0.012812950 0.02905815    0.3390078        0.3389652
3    659.00    0.5991375 0.009148785 0.03054517    0.2120707        0.2345916
4    938.00    0.4703039 0.006894272 0.03200788    0.1997101        0.2180372
5   1313.00    0.4797216 0.007066202 0.03454175    0.1730820        0.1684948
6   1891.00    0.4183546 0.006680265 0.03836143    0.1509584        0.1735455
7   2122.00    0.5117859 0.007847209 0.03600325    0.1715612        0.2003901
8   2221.00    0.4941523 0.007429709 0.03879966    0.2166193        0.1933320
```

```
9  2385.00    0.4466957 0.007199118 0.04701630    0.1918290        0.2106014
10 2556.68    0.5881792 0.008990345 0.06620457    0.2098018        0.2814556

 Lower limit:
    time   X.Intercept.         Age       Nodes  treatmentLev  treatmentLev.5FU
1     23.00 -13654.032259 -4.952281e-02 -0.08047250 -1.362213e+04 -1.434670e+04
2    402.00     -5.569949 -1.707772e-05  0.02355023 -5.083151e-01 -7.366005e-01
3    659.00     -3.395094 -1.456218e-02  0.10291793 -4.140744e-01 -6.876100e-01
4    938.00     -2.925546 -5.846207e-03  0.13256524 -4.007639e-01 -7.864766e-01
5  1313.00     -2.539779 -5.340829e-03  0.13501253 -2.709125e-01 -8.798548e-01
6  1891.00     -2.258663 -4.029937e-03  0.16686576 -3.312062e-01 -8.265328e-01
7  2122.00     -2.626767 -1.968194e-03  0.18537895 -4.605884e-01 -9.189432e-01
8  2221.00     -2.265394 -4.523850e-03  0.17703874 -6.268059e-01 -9.784067e-01
9  2385.00     -2.364197  3.802815e-04  0.16905414 -6.714816e-01 -1.084847e+00
10 2556.68     -2.742778 -1.342478e-03  0.14756768 -6.472959e-01 -1.173929e+00


 Upper limit:
    time   X.Intercept.         Age       Nodes  treatmentLev  treatmentLev.5FU
1     23.00 13597.3811919 0.0004123815 0.002095746 1.362891e+04     1.439525e+04
2    402.00    -2.6157811 0.0502096878 0.137458182 8.205954e-01     5.921430e-01
3    659.00    -1.0464752 0.0213010553 0.222655003 4.172426e-01     2.319888e-01
4    938.00    -1.0819546 0.0211793383 0.258036135 3.820996e-01     6.822917e-02
5  1313.00    -0.6592702 0.0223586842 0.270416172 4.075688e-01    -2.193550e-01
6  1891.00    -0.6187126 0.0221567029 0.317242567 2.605507e-01    -1.462343e-01
7  2122.00    -0.6205667 0.0287928650 0.326511677 2.119315e-01    -1.334141e-01
8  2221.00    -0.3283172 0.0246006100 0.329133417 2.223417e-01    -2.205453e-01
9  2385.00    -0.6131503 0.0286008224 0.353358041 8.048813e-02    -2.592898e-01
10 2556.68    -0.4371159 0.0338996729 0.407089599 1.751272e-01    -7.062360e-02

  p.value:
    time   X.Intercept.         Age       Nodes  treatmentLev  treatmentLev.5FU
1     23.00 9.967490e-01 0.05390150 6.281445e-02    0.9996111      0.997358354
2    402.00 5.604497e-08 0.05015178 5.597856e-03    0.6451000      0.831259885
3    659.00 2.100303e-04 0.71265448 9.855485e-08    0.9940401      0.331501133
4    938.00 2.039302e-05 0.26612969 1.049831e-09    0.9627297      0.099542518
5  1313.00 8.552157e-04 0.22852284 4.392863e-09    0.6930100      0.001106878
6  1891.00 5.840419e-04 0.17486385 2.793398e-10    0.8149671      0.005068814
7  2122.00 1.511058e-03 0.08741659 1.169284e-12    0.4686432      0.008645272
8  2221.00 8.680053e-03 0.17665908 6.896373e-11    0.3505190      0.001930250
9  2385.00 8.602758e-04 0.04413322 2.765814e-08    0.1234587      0.001416893
10 2556.68 6.868203e-03 0.07019108 2.802296e-05    0.2604733      0.027041188


[1] "18 observation(s) deleted due to missingness from 'data'"
```

The `plot()` method is used to plot estimated regression coefficients with 95% confidence bands to visualize possible time-varying effects of covariates along time. Argument covar of `plot.Tpreg()` function indicates the covariates for which their effects are to be plotted. The argument `rug` ((TRUE) by default) adds a rug representation of times between time `s` and time `t`. And argument `Ylim` gives the list of limits for the y axes.

```
> plot(co13, covar = c("Age", "Nodes", "treatmentLev", "treatmentLev.5FU"),
        Ylim = list(c(-0.1,0.1), c(-0.5,0.5), c(-2,2), c(-2,2)))
```

Figure 2 shows the plot corresponding the adjusted effects of Age, Nodes, Lev and Lev+5FU on transition probability p_{13}. The covariate age, shows no significant effect of age on p_{13} along time. The increasing number of nodes with detectable cancer significantly increases the probability of dying steadily over time. As for the effect of treatment, after 1000 days the Lev+5FU decreases the transition probability to the death state (in the long run), while no effect of Lev on p_{13} is appreciated. By setting `trans = all` inside `TPreg()` function, `plot()` simultaneously displays the effect of prespesified covariate(s) on all transition probabilities.

We set `by=1` and `trans=11` to calculate all regression results on transition probability p_{11} for all times between 0 and 7 years. Compared with a larger `by`, `by=1` results in a longer time for R to run the code.

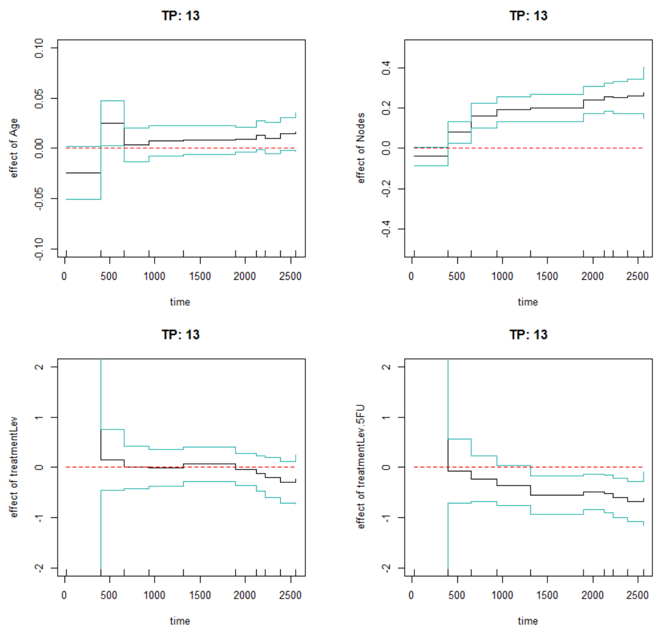


Figure 2: Estimated effects of the Age (upper left corner), Nodes (upper right corner), Lev (lower left corner), and Lev+5FU (lower right corner) on the probability of transition from "Alive and disease-free" to "Dead" along time.

```
> co11 <- TPreg( ~ Age + Nodes + treatment, colonTPreg,  link = "logit", s = 0,
             R = 199, by = 1, t = 365.24*7, trans = "11")
> co11

Call:
TPreg(formula = ~Age + Nodes + treatment, data = colonTPreg,
link = "logit", s = 0, t = 365.24 * 7, R = 199, by = 1, trans = "11")

Transition:
[1] "11"

(s,t):
[1]    0.00 2556.68

Coefficients:
                  Estimate      St.Err        LW.L         UP.L      P.value
X.Intercept.      0.909156911 0.585357182 -0.23814317  2.056456987 0.1203834661
Age              -0.009283428 0.009090987 -0.02710176  0.008534907 0.3071746946
Nodes            -0.260231227 0.068103658 -0.39371440 -0.126748057 0.0001328551
treatmentLev      0.099705824 0.279628624 -0.44836628  0.647777927 0.7214173649
treatmentLev.5FU  0.743733904 0.253849957  0.24618799  1.241279820 0.0033916175


[1] "18 observations deleted due to missingness from 'data'"


> plot(co11, covar = c("Age", "Nodes", "treatmentLev", "treatmentLev.5FU"),
        Ylim = list(c(-0.1,0.1), c(-0.5,0.5), c(-2,2), c(-2,2)))
```

The rugs on x axis are all jump times of the response variable corresponding to transition (i.e. $Y_{11}(t)$). As mentioned in the second section, the estimates of the coefficients are piecewise constant between the jump times. Therefore, Figure 3 depicts all the estimates, for every time between 0 and 7 years. From the figure, we see that the number of nodes significantly reduces the probability of consistently staying healthy and treatment Lev+5FU significantly increases this probability at each time point.

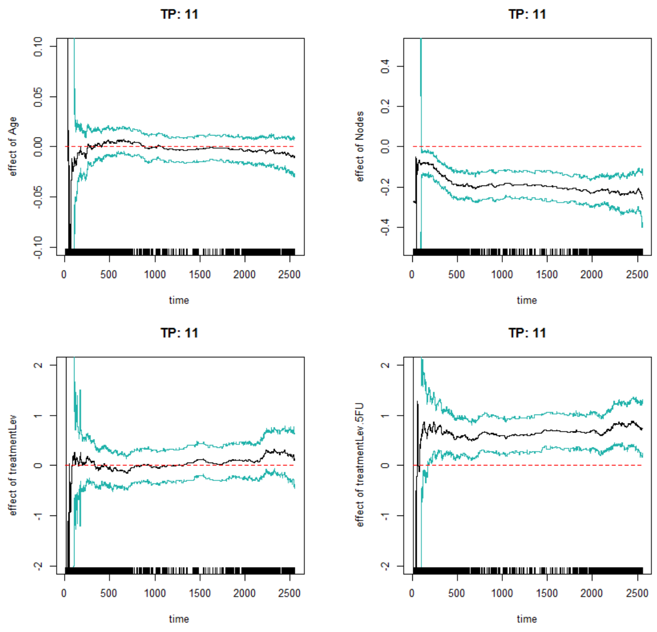


Figure 3: Estimated effects of the Age (upper left corner), Nodes (upper right corner), Lev (lower left -corner), and Lev+5FU (lower right corner) on disease-free survival along time.

Summary

This paper describes the implementation of a flexible method in R for fitting a regression model to possibly non-Markov progressive illness-death data. The **idmTPreg** package offers the user the opportunity to estimate possibly time-varying effect of covariates on the transition probabilities for the progressive illness-death model. We have explained the use of the **idmTPreg** package by applying the method to a colon cancer dataset. The results in this paper were obtained using R 3.4.2. In a future version of the package, we plan to implement a similar method to estimate coefficients on net survivals for a progressive illness-death in a relative survival setting.

Acknowledgment

This work was supported by funding from the European Community's Seventh Framework Programme FP7/2011: Marie Curie Initial Training Network MEDIASRES ("Novel Statistical Methodology for Diagnostic/Prognostic and Therapeutic Studies and Systematic Reviews"; www.mediasres-itn.eu) with the Grant Agreement Number 290025, by the Basque Government through the BERC 2014-2017 program, by Spanish Ministry of Economy and Competitiveness MINECO: BCAM Severo Ochoa excellence accreditation SEV-2013-0323, Grant MTM2016-76969-P and European Regional Development Fund (ERDF). The second author acknowledges financial support from Ministerio de Economía y Competitividad Grant MTM2016-76969-P and European Regional Development Fund (ERDF).

Bibliography

O. O. Aalen, Ø. Borgan, and H. Fekjaer. Covariate adjustment of event histories estimated from markov chains: The additive approach. *Biometrics*, 57(4):993–1001, dec 2001. URL `https://doi.org/10.1111/j.0006-341X.2001.00993.x`. [p317]

P. K. Andersen and N. Keiding. Multi-state models for event history analysis. *Statistical Methods in Medical Research*, 11(2):91–115, apr 2002. URL `https://doi.org/10.1191/0962280202SM276ra`. [p317]

L. Azarang, T. Scheike, and J. de Uña-Álvarez. Direct modeling of regression effects for transition probabilities in the progressive illness-death model. *Statistics in Medicine*, 2017. URL `https://doi.org/10.1002/sim.7245`. [p317, 318]

L. C. de Wreede, M. Fiocco, and H. Putter. The mstate package for estimation and prediction in non- and semi-parametric multi-state and competing risks models. *Computer Methods and Programs in Biomedicine*, 99(3):261–274, sep 2010. URL `https://doi.org/10.1016/j.cmpb.2010.01.001`. [p317]

P. Hougaard. *Analysis of Multivariate Survival Data*, 2000. ISBN 978-1-4612-1304-8. [p317]

C. H. Jackson. Multi-state models for panel data: The msmPackage forR. *Journal of Statistical Software*, 38(8), 2011. [p317]

L. Meira-Machado and J. Roca-Pardiñas. p3state.msm: Analyzing survival data from an illness-death model. *Journal of Statistical Software*, 38(3), 2011. [p317]

L. Meira-Machado, J. de Uña-Álvarez, C. Cadarso-Suárez, and P. K. Andersen. Multi-state models for the analysis of time-to-event data. *Statistical Methods in Medical Research*, 18(2):195–222, apr 2008. URL `https://doi.org/10.1177/0962280208092301`. [p317]

L. Meira-Machado, J. de Uña-Álvarez, and S. Datta. Nonparametric estimation of conditional transition probabilities in a non-markov illness-death model. *Computational Statistics*, 30(2):377–397, oct 2014. URL `https://doi.org/10.1007/s00180-014-0538-6`. [p317]

C. G. Moertel, T. R. Fleming, J. S. Macdonald, D. G. Haller, J. A. Laurie, P. J. Goodman, J. S. Ungerleider, W. A. Emerson, D. C. Tormey, J. H. Glick, M. H. Veeder, and J. A. Mailliard. Levamisole and fluorouracil for adjuvant therapy of resected colon carcinoma. *New England Journal of Medicine*, 322 (6):352–358, feb 1990. URL `https://doi.org/10.1056/NEJM199002083220602`. [p319]

Leyla Azarang
BCAM - Basque Center for Applied Mathematics
Bilbao, Vizcaya, Spain
ORCiD: 0000-0002-1785-3857
`lazarang@bcamath.org`

Manuel Oviedo de la Fuente
Technological Institute for Industrial Mathematics
Universidade de Santiago de Compostela
Santiago de Compostela, A Coruña, Spain
ORCiD: 0000-0001-7360-3249
`manuel.oviedo@usc.es`

lmridge: A Comprehensive R Package for Ridge Regression

by Muhammad Imdad Ullah, Muhammad Aslam, and Saima Altaf

Abstract The ridge regression estimator, one of the commonly used alternatives to the conventional ordinary least squares estimator, avoids the adverse effects in the situations when there exists some considerable degree of multicollinearity among the regressors. There are many software packages available for estimation of ridge regression coefficients. However, most of them display limited methods to estimate the ridge biasing parameters without testing procedures. Our developed package, **lmridge** can be used to estimate ridge coefficients considering a range of different existing biasing parameters, to test these coefficients with more than 25 ridge related statistics, and to present different graphical displays of these statistics.

Introduction

For data collected either from a designed experiment or from an observational study, the ordinary least squares (OLS) method does not provide precise estimates of the effect of any explanatory variable (regressor) when regressors are interdependent (collinear with each other). Consider a multiple linear regression (MLR) model,

$$y = X\beta + \varepsilon, \tag{1}$$

where y is an $n \times 1$ vector of observation on dependent variable, X is known design matrix of order $n \times p$, β is a $p \times 1$ vector of unknown parameters and ε is an $n \times 1$ vector of random errors with mean zero and variance $\sigma^2 I_n$, where I_n is an identity matrix of order n.

The OLS estimator (OLSE) of β is given by

$$\hat{\beta} = (X'X)^{-1}X'y, \tag{2}$$

which depends on characteristics of the matrix $X'X$. If $X'X$ is ill-conditioned (near dependencies among various columns (regressors) of $X'X$ exist) or $det(X'X) \approx 0$, then the OLS estimates are sensitive to a number of errors, such as non-significant or imprecise regression coefficients (Kmenta, 1980) with wrong sign and non-uniform eigenvalues spectrum. Moreover, the OLS method, can yield high variances of estimates, large standard errors, and wide confidence intervals. Quality and stability of the fitted model may be questionable due to erratic behaviour of the OLSE in case when regressors are collinear.

Researchers may tempt to eliminate regressor(s) causing the problem by consciously removing regressors from the model. However, this method may destroy the usefulness of the model by removing relevant regressor(s) from the model. To control variance and instability of the OLS estimates, one may regularize the coefficients, with some regularization methods such as ridge regression (RR), Liu regression, and Lasso regression methods etc., as alternative to OLS. Computationally, RR suppresses the effects of collinearity and reduces the apparent magnitude of the correlation among regressors in order to obtain more stable estimates of the coefficients than the OLS estimates and it also improves accuracy of prediction (see Hoerl and Kennard, 1970a; Montgomery and Peck, 1982; Myers, 1986; Rawlings et al., 1998; Seber and Lee, 2003; Tripp, 1983, etc.).

There are only a few software programs and R packages capable of estimating and/ or testing of ridge coefficients. The design goal of our **lmridge** (Imdad and Aslam, 2018b) is primarily to provide functionality of all possible ridge related computations. The output of our developed package (**lmridge**) is consistent with output of existing software/ R packages. The package, **lmridge** also provides the most complete suite of tools for ordinary RR, comparable to those listed in Table 1. For package development and R documentation, we followed Hadley (2015), Leisch (2008) and R Core Team (2015). The **ridge** package by Moritz and Cule (2017) and `lm.ridge()` from the **MASS** (Venables and Ripley, 2002) also provided guidance in coding.

All available software and R packages mentioned in Table 1 are compared with our **lmridge** package. For multicollinearity detection, NCSS statistical software (NCSS 11 Statistical Software, 2016) computes VIF/TOL, R^2, eigenvalue, eigenvector, incremental and cumulative percentage of eigenvalues and CN. For RR, ANOVA table, coefficient of variation, plot of residuals vs predicted, histogram and density trace of residuals are also available in NCSS. In SAS (Inc., 2011), `collin` option in the `model` statement is used to perform collinearity diagnostics while for remedy of multicollinearity, RR can be performed using a `ridge` option in `proc reg` statement. The `outVIF` option results in

	NCSS	SAS	Stata	StatGraphics	**lrmest**	**ltsbase**	**penalized**	**glmnet**	**ridge**	**lmridge**
Standardization of regressors										
	✓	✓	✓	✓		✓	✓	✓	✓	✓
Estimation and testing of ridge coefficient										
Estimation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Testing			✓		✓				✓	✓
SE of coef	✓		✓		✓				✓	✓
Ridge related statistics										
R^2	✓		✓	✓						✓
adj-R^2			✓	✓						✓
m-scale & ISRM										✓
Variance										✓
Bias^2										✓
MSE					✓	✓				✓
F-test			✓							✓
Shrinkage factor										✓
CN										✓
σ^2										✓
C$_k$										✓
DF										✓
EDF										✓
Eft										✓
Hat matrix										✓
Var-Cov matrix										✓
VIF	✓			✓					✓	✓
Residuals	✓		✓	✓		✓	✓			✓
Ridge fitted						✓	✓			✓
Predict	✓		✓	✓			✓	✓	✓	✓
Ridge model selection										
CV & GCV			✓				✓	✓		✓
AIC & BIC										✓
PRESS										✓
Ridge related graphs										
Ridge trace	✓	✓		✓					✓	✓
VIF trace	✓	✓		✓						✓
Bias, var, MSE										✓
CV, GCV										✓
AIC & BIC										✓
m-scale, ISRM										✓
DF, RSS, PRESS										✓

Table 1: Comparison of ridge related software and R packages.

VIF values. For RR, Stata (StataCorp, 2014) has no built-in command, however `ridgereg` add-on is available that performs calculation on scalar `k`. The **lrmest** package (Dissanayake et al., 2016) computes estimators such as OLS, ordinary RR (ORR), Liu estimator (LE), LE type-1,2,3, Adjusted Liu Estimator (ALTE), and their type-1,2,3 etc. Moreover, **lrmest** provides scalar mean square error (MSE), prediction residual error sum of squares (PRESS) values of some of the estimators. The testing of ridge coefficient is performed only on scalar `k`, however, for vector of `k`, function `rid()` of **lrmest** package returns only MSE along with value of biasing parameter used. The function `optimum()` of **lrmest** package can be used to get the optimal scalar MSE and PRESS values (Arumairajan and Wijekoon, 2015). Statgraphics standardizes the dependent variable and computes some statistics for detection of collinearity such as R^2, adj-R^2, and VIF. Statgraphics also facilitates to perform RR and computes different RR related statistics such as VIF and ridge trace for different biasing parameter used, R^2, adj-R^2 and standard error of estimates etc. The **ltsbase** package (Kan-Kilinc and Alpu, 2013, 2015) computes ridge and Liu estimates based on the least trimmed squares (LTS) method. The MSE value from four regression models can be compared graphically if the argument `plot=TRUE` is passed to the `ltsbase()` function. There are three main functions (i) `ltsbase()` computes the minimum MSE values for six models: OLS, ridge, ridge based on LTS, LTS, Liu, and Liu based on LTS method for sequences of biasing parameters ranging from 0 to 1. If `print=TRUE`, `ltsbase()` prints all the MSEs (along with minimum MSE) for ridge, Liu, and ridge & Liu based on LTS method for the sequence of biasing parameters given by the user, (ii) the `ltsbaseDefault()` function returns the fitted values and residual of the six models (OLS, ridge, Liu, LTS, and ridge & Liu based LTS methods) having minimum MSE, and (iii) the `ltsbaseSummary()` function returns the coefficients and the biasing parameter for the best MSE among the four regression models. The **penalized** package (Goeman et al., 2017) is designed for penalized estimation in generalized linear models. The supported models are linear regression, logistic regression, Poisson regression and the Cox proportional hazard models. The **penalized** package allows an L1 absolute value ("LASSO") penalty, and L2 quadratic ("ridge") penalty or a combination of the two. It is also possible to have a fused LASSO penalty with L1 absolute value penalty on the coefficients and their differences. The **penalized** package also includes facilities for likelihood, cross-validation and for optimization of the tuning parameter. The **glmnet** package (Friedman et al., 2010) has some efficient procedures for fitting the entire LASSO or elastic-net regularization path for linear regression, logistic and multinomial regression model, Poisson regression and Cox model. The **glmnet** can also be used to fit the RR model by setting `alpha` argument to zero. The **ridge** package fits linear and also logistic RR models, including functions for fitting linear and logistic RR models for genome-wide SNP data supplied as files names when the data are too big to read into R. The RR biasing parameter is chosen automatically using the method proposed by Cule and De Iorio (2012), however value of biasing parameter can also be specified for estimation and testing of ridge coefficients. The function, `lm.ridge()` from **MASS** only fits linear RR model and returns ridge biasing parameters given by Hoerl and Kennard (1970a) and Venables and Ripley (2002) and vector GCV criterion, given by Golub et al. (1979).

There are other software and R packages that can be used to perform RR analysis such as S-PLUS (S-PLUS, 2008), Shazam (Shazam, 2011) and R packages such as **RXshrink** (Obenchain, 2014), **rrBLUP** (Endelman, 2011), **RidgeFusion** (Price, 2014), **bigRR** (Shen et al., 2013), **lpridge** (Seifert, 2013), **genridge** (Friendly, 2017) and **CoxRidge** (Perperoglou, 2015) etc.

This paper outlines the collinearity detection methods available in the existing literature and uses the **mctest** (Imdad and Aslam, 2018a) package through an illustrative example. To overcome the issues of the collinearity effect on regressors a thorough introduction to ridge regression, properties of the ridge estimator, different methods for selecting values of k, and testing of the ridge coefficients are presented. Finally, estimation of the ridge coefficients, methods of selecting a ridge biasing parameter, testing of the ridge coefficients, and different ridge related statistics are implemented in R within the **lmridge**.

Collinearity detection

Diagnosing collinearity is important to many researchers. It consists of two related but separate elements: (1) detecting the existence of collinear relationship among regressors and (2) assessing the extent to which this relationship has degraded the parameter estimates. There are many diagnostic measures used for detection of collinearity in the existing literature provided by various authors (Belsley et al., 1980; Curto and Pinto, 2011; Farrar and Glauber, 1967; Fox and Weisberg, 2011; Gunst and Mason, 1977; Imdadullah et al., 2016; Klein, 1962; Koutsoyiannis, 1977; Kovács et al., 2005; Marquardt, 1970; Theil, 1971). These diagnostics methods assist in determining whether and where some corrective action is necessary (Belsley et al., 1980). Widely used, and the most suggested diagnostics, are value of pair-wise correlations, variance inflation factor (VIF)/ tolerance (TOL) (Marquardt, 1970), eigenvalues and eigenvectors (Kendall, 1957), CN & CI (Belsley et al., 1980; Chatterjee and Hadi, 2006; Maddala, 1988), Leamer's method (Greene, 2002), Klein's rule (Klein, 1962), the tests proposed by Farrar and Glauber (Farrar and Glauber, 1967), Red indicator (Kovács et al., 2005), corrected VIF (Curto and Pinto, 2011) and Theil's measures (Theil, 1971), (see also Imdadullah et al. (2016)). All of these diagnostic measures are implemented in the R package, **mctest**. Below, we use the Hald dataset (Hald, 1952), for testing collinearity among regressors. We then use the **lmridge** package to compute the ridge coefficients for different ridge related statistics and methods of selection of ridge biasing parameter is also performed. For optimal choice of ridge biasing parameter, graphical representations of the ridge coefficients, vif values, cross validation criteria (CV & GCV), ridge DF, RSS, PRESS, ISRM and m-scale versus used ridge biasing parameter are considered. In addition graphical representation of model selection criteria (AIC & BIC) of ridge regression versus ridge DF is also performed. The Hald data are about heat generated during setting of 13 cement mixtures of 4 basic ingredients and used by Hoerl et al. (1975). Each ingredient percentage appears to be rounded down to a full integer. The data set is already bundled in **mctest** and **lmridge** packages.

Collinearity detection: Illustrative example

```
> library("mctest")
> x <- Hald[, -1]
> y <- Hald[,  1]
> mctest (x, y)
Call:
omcdiag(x = x, y = y, Inter = TRUE, detr = detr, red = red, conf = conf,
   theil = theil, cn = cn)
```

```
Overall Multicollinearity Diagnostics

                       MC Results detection
Determinant |X'X|:         0.0011         1
Farrar Chi-Square:        59.8700         1
Red Indicator:             0.5414         1
Sum of Lambda Inverse:   622.3006         1
Theil's Method:            0.9981         1
Condition Number:        249.5783         1

1 --> COLLINEARITY is detected
0 --> COLLINEARITY is not detected by the test
```

The results from all overall collinearity diagnostic measures indicate the existence of collinearity among regressor(s). These results do not tell which regressor(s) are reasons of collinearity. The individual collinearity diagnostic measures can be obtained through:

```
> imcdiag(x = x, y, all = TRUE)
Call:
imcdiag(x = x, y = y, method = method, corr = FALSE, vif = vif,
       tol = tol, conf = conf, cvif = cvif, leamer = leamer, all = all)

Individual Multicollinearity Diagnostics

    VIF  TOL  Wi  Fi  Leamer  CVIF  Klein
X1    1    1   1   1       0     0      0
X2    1    1   1   1       1     0      1
X3    1    1   1   1       1     0      0
X4    1    1   1   1       0     0      1

1 --> COLLINEARITY is detected
0 --> COLLINEARITY is not detected by the test

X1, X2, X3, X4, coefficient(s) are non-significant may be due to multicollinearity

R-square of y on all x: 0.9824

* use method argument to check which regressors may be the reason of collinearity
```

Results from the most of individual collinearity diagnostics suggest that all of the regressors are the reason for collinearity among regressors. The last line of `imcdiag()` function's output suggests that `method` argument should be used to check which regressors may be the reason of collinearity among different regressors. For further information about `method` argument, see the help file of `imcdiag()` function.

Ridge regression analysis

In the seminal work by Hoerl (1959, 1962, 1964) and Hoerl and Kennard (1970b,a) have developed ridge analysis technique that purports the departure of the data from orthogonality. Hoerl (1962) introduced the RR, based on the James-Stein estimator by stating that existence of correlation among regressors can cause errors in estimating regression coefficients when applying the OLS method. The RR is similar to the OLS method however, it shrinks the coefficients towards zero by minimizing the MSE of the estimates, making the RR technique better than the OLSE with respect to MSE, when regressors are collinear with each other. A penalty (degree of bias) is imposed on the size of coefficients in the RR to reduce their variances. However, the expected values of these estimates are not equal to the true values and tend to under estimate the true parameter. Though the ridge estimators are biased but have lower MSE (more precision) than the OLSEs have, less sensitive to sampling fluctuations or model misspecification if number of regressors is more than the number of observations in a data set (i.e., $p > n$), and omitted variables specification bias (Theil, 1957). In summary, the RR procedure is intended to overcome the ill-conditioned situation, and is used to improve the estimation of regression coefficients when regressors are correlated and it also improves the accuracy of prediction (Seber and Lee, 2003). Obtaining the ridge model coefficients ($\hat{\beta}_R$) is relatively straight forward, because the ridge coefficients are obtained by solving a slightly modified form of the OLS method.

The design matrix X in Eq. (1) can be standardized, scaled or centered. Usually, standardization of X matrix is done as described by Belsley et al. (1980) and Draper and Smith (1998), that is, $X_j = \frac{x_{ij}-\bar{x}_j}{\sqrt{\sum(x_{ij}-\bar{x}_j)^2}}$; where $j = 1, 2, \cdots, p$ such that $\overline{X}_j = 0$ and $X_j'X_j = 1$, where X_j is the jth column of the matrix X. In this way, the new design matrix (say $\tilde{X}$) that contains the standardized p columns and the matrix $\tilde{X}'\tilde{X}$ will be correlation matrix of regressors. To avoid complexity of different notations and terms, the centered and scaled design matrix $\tilde{X}$ will be represented by X and centered response variable as y.

The ridge model coefficients are estimated as,

$$\hat{\beta}_{R_k} = (X'X + kI_p)^{-1}X'y, \tag{3}$$

where $\hat{\beta}_{R_k}$ is the vector of standardized RR coefficients of order $p \times 1$ and kI_p is a positive semi-definite matrix added to the $X'X$ matrix. Note that for $k = 0$, $\hat{\beta}_{R_k} = \hat{\beta}_{ols}$.

The addition of constant term k to diagonal element of $X'X$ (in other words addition of kI_p to $X'X$) in Eq. (3) is known as penalty and k is called the biasing or shrinkage parameter. Addition of this biasing parameter guarantees the invertibility of $X'X$ matrix, such that there is always a unique solution $\hat{\beta}_{R_k}$ exists (Draper and Smith, 1998; Hoerl and Kennard, 1970a; McCallum, 1970) and the condition number (CN) of $X'X + kI$ ($CN_k = \sqrt{\frac{\lambda_1+kI}{\lambda_p+kI}}$) also becomes smaller as compared to that of $X'X$, where λ_1 is the largest and λ_p is the smallest eigenvalues of the correlation matrix $X'X$. Therefore, the ridge estimator (RE) is an improvement over the OLSE for collinear data.

It is desirable to select the smallest value of k for which stabilized regression coefficients occur and there always exists a particular value of k for which the total MSE of the REs is less than the MSE of the OLSE, however, the optimum value of k (which produces minimum MSE as compared to other values of ks) varies from one application to another and hence optimal value of k is unknown. Any estimator that has a small amount of bias, less variance and substantially more precise than an unbiased estimator may be preferred since it will have larger probability of being close to the true parameter being estimated. Therefore, criterion of goodness of estimation considered in the RR is the minimum total MSE.

Properties of the ridge estimator

Let X_j denotes the jth column of X $(1, 2, \cdots, p)$, where $X_j = (x_{1j}, x_{2j}, \cdots, x_{nj})'$. As already discussed, assume that the regressors are centered such that $\sum\limits_{i=1}^{n} x_{ij} = 0$ and $\sum\limits_{i=1}^{n} x_{ij}^2 = 1$ and the response variable y is centered.

The RR is the most popular among biased methods, because of its relationship to the OLS method and statistical properties of the RE are also well defined. Most of the RR properties have been discussed, proved and extended by many researchers such as Allen (1974); Hemmerle (1975); Hoerl and Kennard (1970b,a); Marquardt (1970); McDonald and Galarneau (1975); Newhouse and Oman (1971). Table 2 lists the RR properties.

Theoretically and practically, the RR is used to propose some new methods for the choice of the biasing parameter k to investigate the properties of RE, since biasing parameter plays a key role while the optimal choice of k is the main issue in this context. In the literature, there are many methods for estimating the biasing parameter k (see Allen, 1974; Guilkey and Murphy, 1975; Hemmerle, 1975; Hoerl and Kennard, 1970b,a; McDonald and Galarneau, 1975; Obenchain, 1977; Hocking et al., 1976; Lawless and Wang, 1976; Vinod, 1976; Kasarda and Shih, 1977; Hemmerle and Brantle, 1978; Wichern and Churchill, 1978; Nordberg, 1982; Saleh and Kibria, 1993; Singh and Tracy, 1999; Wencheko, 2000; Kibria, 2003; Khalaf and Shukur, 2005; Alkhamisi et al., 2006; Alkhamisi and Shukur, 2007; Khalaf, 2013, among many more), however, there is no consensus about which method is preferable (Chatterjee and Hadi, 2006). Similarly, each of the estimation method of biasing parameter cannot guarantee to give a better k or even cannot give a smaller MSE as compared to that for the OLS.

Methods of selecting values of k

The optimal value of k is one which gives minimum MSE. There is one optimal k for any problem, while a wide range of k ($0 < k < k_{opt}$) give smaller MSE as compared to that of the OLS. For collinear data, a small change in k varies the RR coefficients rapidly. At some values of k, the ridge coefficients get stabilized and the rate of change slow down gradually to almost zero. Therefore, a disciplined way of selecting the shrinkage parameter is required that minimizes the MSE. The biasing parameter k depends on the true regression coefficients (β) and the variance of the residuals σ^2, unfortunately

sr.#	Property	Formula
1)	Mean	$E(\hat{\beta}_R) = (X'X + kI_p)^{-1}X'X\beta$
2)	Shorter regression coeffs.	$\hat{\beta}'_R\hat{\beta}_R \leq \hat{\beta}'\hat{\beta}$
3)	Linear transformation	$\hat{\beta}_R = Z\hat{\beta}$,where $Z = (X'X + kI)^{-1}X'X$
4)	Variance	$Var(\hat{\beta}_R) = \sigma^2 \sum_{j=1}^{p} \frac{\lambda_j}{(\lambda_j+k)^2}$
5)	Var-Cov matrix	$Cov(\hat{\beta}_R) = Cov(Z\hat{\beta}) = \sigma^2(X'X + kI)^{-1}X'X(X'X + kI)^{-1} = \sigma^2[VIF]$
6)	Bias	$Bias(\hat{\beta}_R) = -k(X'X + kI)^{-1}\beta = -k\,P\,diag\left(\frac{1}{\lambda_j + k}\right)P'\beta$
7)	MSE	$MSE = \sigma^2 \sum_{j=1}^{p} \frac{\lambda_j}{(\lambda_j+k)^2} + \sum_{j=1}^{p} \frac{k^2\alpha_j^2}{(\lambda_j+k)^2}$
8)	Distance between $\hat{\beta}_R$ and β	$\hat{\beta}_R$ and the true vector of β have minimum distance
9)	Inflated RSS	$\phi_0 = k^2\hat{\beta}'_R(X'X)^{-1}\hat{\beta}_R$
10)	R_R^2	$R_R^2 = \frac{\hat{\beta}'_R X'y - k\hat{\beta}'_R\hat{\beta}_R}{y'y}$
11)	Sampling fluctuations	The $\hat{\beta}_R$ is less sensitive to the sampling fluctuation
12)	Accurate prediction	$\sigma^2_{f_R} = \sigma^2\left[1 + x'\,P\,diag\left(\frac{\lambda_j}{(\lambda+k)^2}\right)P'x\right] + (Bias(\hat{\beta}_R))^2$
13)	Wide range of k	$0 < k < k_{max}$, have smaller set of MSE than OLSE
14)	Optimal k	An optimal k always exists that gives minimum MSE
15)	DF Ridge	$df_{R_k} = EDF = \sum_{j=1}^{p} \frac{\lambda_j}{\lambda_j+k} = trace\left[H_{R_k}\right]$, where $H_{Rk} = X\left(X'X + kI\right)^{-1}X'$
16	Effective no. of parameters	$EP = trace[2H_{R_k} - H_{R_k}H'_{R_k}]$
17	Residual EDF	$REDF = n - trace[2H_{R_k} - H_{R_k}H'_{R_k}] = n - EP$

Table 2: Properties of the ridge estimator.

these are unknown, but they can be estimated from the sample data.

We classified these estimation method as (i) Subjective or (ii) Objective

Subjective methods

In all these methods, the selection of k is subjective or of judgmental nature and provides graphical evidence of the effect of collinearity on the regression coefficient estimates and also accounts for variation by the RE as compared to the OLSE. In these methods, the reasonable choice of k is done using the ridge trace, df trace, VIF trace and plotting of bias, variance, and MSE. The ridge trace is a graphical representation of regression coefficients $\hat{\beta}_R$, as a function of k over the interval $[0, 1]$. The df trace and VIF trace are like the ridge trace plot in which EDF and VIF values are plotted against k. Similarly, plotting of bias, variance, and MSE from the RE may also be helpful in selecting an appropriate value of k. All these graphs can be used for selection of optimal (but judgmental) value of k from horizontal axis to assess the effect of collinearity on each of the coefficients. The effect of collinearity is depressed when value of k increases and all the values of the ridge coefficients, EDF and VIF values decrease and/ or may stabilize after certain value of k. These graphical representations do not provide a unique solution, rather they render a vaguely defined class of acceptable solutions. However, these traces are still useful graphical representations to check for some optimal k.

Objective methods

Suppose, we have set of observations $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$ and the RR model as given in Eq. (3). Objective methods, to some extent, are similar to judgmental methods for selection of biasing parameter k, but they require some calculations to obtain these biasing parameters. Table 3 lists widely used methods to estimate the biasing parameter k already available in the existing literature. Table 3 also lists other statistics that can be used for the selection of the biasing parameter k. There are other

method	formula	reference
C_k	$C_k = \frac{SSR_k}{s^2} - n + 2 + 2\,trace(H_{R_k}) = \frac{SSR_k}{s^2} + 2(1 + trace(H_{R_k})) - n$	Kennard (1971); Mallows (1973)
$PRESS_k$	$PRESS_k = \sum_{i=1}^{n}(y_i - \hat{y}_{(i,-i)_k})^2 = \sum_{i=1}^{n} e^2_{(i,-i)_k}$	Allen (1971, 1974)
CV	$CV_k = n^{-1}\sum_{i=1}^{n}(y_i - X_j\hat{\beta}_{jR_k})^2$	Delaney and Chatterjee (1986)
GCV	$GCV_k = \frac{SRR_k}{n-(1+trace(H_{R_k}))^2}$	Golub et al. (1979)
ISRM	$ISRM_k = \sum_{j=1}^{p}\left(\frac{p\left(\frac{\lambda_j}{\lambda_j+k}\right)^2}{\sum_{j=1}^{p}\frac{\lambda_j}{(\lambda_j+k)^2}\lambda_j} - 1\right)^2$	Vinod (1976)
m-scale	$m = p - \sum_{j=1}^{p}\frac{\lambda_j}{\lambda_j+k}$	Vinod (1976)
Information criteria	$AIC = n\cdot\log(RSS/n) + 2\cdot df_{Rk}$ $BIC = n\cdot\log(RSS) + 2\cdot df_{Rk}$	Akaike (1973); Schwarz (1978)
Effectiveness index (Eft)	$EF = \frac{\sigma^2\,trace(X'X)^{-1} - \sigma^2\,trace(VIF)}{(Bias(\hat{\beta}_R))^2}$	Lee (1979)

Table 3: Objective methods for selection of biasing parameter k.

methods to estimate biasing parameter k. Table 4 lists various methods for the selection of biasing parameter k, proposed by different researchers.

Testing of the ridge coefficients

Investigating of the individual coefficients in a linear but biased regression models such as ridge based, exact and non-exact t type and F test can be used. Exact t-statistics derived by Obenchain (1977) based on the RR for matrix G whose columns are the normalized eigenvectors of $X'X$, is,

$$t^* = \frac{\hat{\beta}_{R_j} - \beta_j}{\sqrt{\hat{var}(\hat{\beta}_{R_j} - \beta_j)}}, \tag{4}$$

where $j = 1, 2, \cdots, p$, $\hat{var}(\hat{\beta}_{R_j} - \beta_j)$ is an unbiased estimator of the variance of the numerator in Eq. (4), and

$$\beta_j = g_i'\Delta G'[I - (X'X)^{-1}e_i'(e_i(X'X)^{-1}e_i')^{-1}]\hat{\beta}(0),$$

where g_i' is the ith row of G, Δ is the $(p\times p)$ diagonal matrix with ith diagonal element given by $\delta_i = \frac{\lambda_i}{\lambda_i+k}$ and e_i is the ith row of the identity matrix.

It has been established that $\beta_R \sim N(ZX\beta, \phi = Z\Omega Z')$, where $Z = (X'X + kI_p)^{-1}X'$. Therefore, for jth ridge coefficient $\beta_R \sim N(Z_jX\beta, \phi_{jj} = Z_j\Omega Z_j')$ (see Aslam, 2014; Halawa and El-Bassiouni, 2000). Halawa and El-Bassiouni (2000) presented to tackle the problem of testing $H_0 : \beta_j = 0$ by considering a non-exact t type test of the form,

$$t_{R_j} = \frac{\hat{\beta}_{R_j}}{\sqrt{S^2(\hat{\beta}_{R_j})}},$$

where $\hat{\beta}_{R_j}$ is the jth element of RE and $S^2(\hat{\beta}_{R_j})$ is an estimate of the variance of $\hat{\beta}_{R_j}$ given by the ith diagonal element of the matrix $\sigma^2(X'X + kI_p)^{-1}X'X(X'X + kI_p)^{-1}$.

Sr. #	Formula	Reference
1)	$K_{HKB} = \frac{p\hat{\sigma}^2}{\hat{\beta}'\hat{\beta}}$	Hoerl and Kennard (1970a)
2)	$K_{TH} = \frac{(p-2)\hat{\sigma}2}{\hat{\beta}'\hat{\beta}}$	Thisted (1976)
3)	$K_{LW} = \frac{p\hat{\sigma}^2}{\sum_{j=1}^{p}\lambda_j\hat{\alpha}_j^2}$	Lawless and Wang (1976)
4)	$K_{DS} = \frac{\hat{\sigma}^2}{\hat{\beta}'\hat{\beta}}$	Dwividi and Shrivastava (1978)
5)	$K_{LW} = \frac{(p-2)\hat{\sigma}^2 \times n}{\hat{\beta}'X'X\hat{\beta}}$	Venables and Ripley (2002)
6)	$K_{AM} = \frac{1}{p}\sum\limits_{j=1}^{p}\frac{\hat{\sigma}^2}{\hat{\alpha}\ j}$	Kibria (2003)
7)	$\hat{K}_{GM} = \frac{\hat{\sigma}^2}{\left(\prod\limits_{j=1}^{p}\hat{\alpha}_j^2\right)^{\frac{1}{p}}}$	Kibria (2003)
8)	$\hat{K}_{MED} = Median\{\frac{\hat{\sigma}^2}{\hat{\alpha}_j^2}\}$	Kibria (2003)
9)	$K_{KM2} = max\left(\frac{1}{\sqrt{\frac{\hat{\sigma}^2}{\hat{\alpha}_j^2}}}\right)$	Muniz and Kibria (2009)
10)	$K_{KM3} = max\left(\sqrt{\frac{\hat{\sigma}_j^2}{\hat{\alpha}_j^2}}\right)$	Muniz and Kibria (2009)
11)	$K_{KM4} = \left(\prod\limits_{j=1}^{p}\frac{1}{\sqrt{\frac{\hat{\sigma}_j^2}{\hat{\alpha}_j^2}}}\right)^{\frac{1}{p}}$	Muniz and Kibria (2009)
12)	$K_{KM5} = \left(\prod\limits_{j=1}^{p}\sqrt{\frac{\hat{\sigma}_j^2}{\hat{\alpha}_j^2}}\right)^{\frac{1}{p}}$	Muniz and Kibria (2009)
13)	$K_{KM6} = Median\left(\frac{1}{\sqrt{\frac{\hat{\sigma}_j^2}{\hat{\alpha}_j^2}}}\right)$	Muniz and Kibria (2009)
14)	$K_{KM8} = max\left(\frac{1}{\sqrt{\frac{\lambda_{max}\hat{\sigma}^2}{(n-p)\hat{\sigma}^2+\lambda_{max}\hat{\alpha}_j^2}}}\right)$	Muniz et al. (2012)
15)	$K_{KM9} = max\left(\sqrt{\frac{\lambda_{max}\hat{\sigma}^2}{(n-p)\hat{\sigma}^2+\lambda_{max}\hat{\alpha}_j^2}}\right)$	Muniz et al. (2012)
16)	$K_{KM10} = \left(\prod\limits_{j=1}^{p}\frac{1}{\sqrt{\frac{\lambda_{max}\hat{\sigma}^2}{(n-p)\sigma^2+\lambda_{max}\hat{\alpha}_j^2}}}\right)^{\frac{1}{p}}$	Muniz et al. (2012)
17)	$K_{KM11} = \left(\prod\limits_{j=1}^{p}\sqrt{\frac{\lambda_{max}\hat{\sigma}^2}{(n-p)\hat{\sigma}^2+\lambda_{max}\hat{\alpha}_j^2}}\right)^{\frac{1}{p}}$	Muniz et al. (2012)
18)	$K_{KM12} = Median\left(\frac{1}{\sqrt{\frac{\lambda_{max}\hat{\sigma}^2}{(n-p)\hat{\sigma}^2+\lambda_{max}\hat{\alpha}_j^2}}}\right)$	Muniz et al. (2012)
19)	$K_{KD} = max\left(0, \frac{p\hat{\sigma}^2}{\hat{\alpha}'\hat{\alpha}} - \frac{1}{n(VIF_j)_{max}}\right)$	Dorugade and Kashid (2010)
20)	$K_{4(AD)} = Harmonic\ Mean[K_i(AD)]$ $= \frac{2p}{\lambda_{max}}\sum\limits_{j=1}^{p}\frac{\hat{\sigma}^2}{\hat{\alpha}_j^2}$	Dorugade (2014)

Table 4: Different available methods to estimate *k*.

The statistic t_{R_j} is assumed to follow a Student's t distribution with $(n-p)$ d.f. (Halawa and El-Bassiouni, 2000). Hastie and Tibshirani (1990); Cule and De Iorio (2012) suggested to use $[n - trace(H_{Rk})]$ d.f. For large sample size, the asymptotic distribution of this statistic is normal (Halawa and El-Bassiouni, 2000). Thus, H_0 is rejected when $|T| > Z_{1-\frac{\alpha}{2}}$.

Similarly, for testing the hypothesis $H_0 : \beta \neq \beta_0$, where β_0 is vector of fixed values. The F statistic for significance testing of the ORR estimator β_R with $E(\hat{\beta}_R) = ZX\beta$ and estimate of $Cov(\beta_R)$ distributed as $F(DF_{ridge}, REDF)$ is

$$F = \frac{1}{p}(\hat{\beta}_R - ZX\beta)' \left(Cov(\hat{\beta}_R)\right)^{-1} (\hat{\beta}_R - ZX\beta)$$

The R package lmridge

Our R package **lmridge** contains functions related to fitting of the RR model and provides a simple way of obtaining the estimates of RR coefficients, testing of the ridge coefficients, and computation of different ridge related statistics, which prove helpful for selection of optimal biasing parameter k. The package computes different ridge related measures available for the selection of biasing parameter k, and also computes value of different biasing parameters proposed by some researchers in the literature.

The **lmridge** objects contain a set of standard methods such as `print()`,`summary()`,`plot()` and `predict()`. Therefore, inferences can be made easily using `summary()` method for assessing the estimates of regression coefficients, their standard errors, t values and their respective p values. The default function `lmridge` which calls `lmridgeEst()` to perform required computations and estimation for given values of non-stochastic biasing parameter k. The syntax of default function is,

```
lmridge (formula,data,scaling = ("sc","scaled","centered"),K,...)
```

The four arguments of **lmridge()** are described in Table 5:

Argument	Description
`formula`	Symbolic representation for RR model of the form, response $\sim$ predictors.
`data`	Contains the variables that have to be used in RR model.
`K`	The biasing parameter, may be a scalar or vector. If a K value is not provided, $K = 0$ will be used as the default value, i.e., the OLS results will be produced.
`scaling`	The methods for scaling the predictors. The `sc` option uses the default scaling of the predictors in correlation form as described in (Belsley, 1991; Draper and Smith, 1998); the `scaled` option standardizes the predictors having zero mean and unit variance; and the `centered` option centers the predictors.

Table 5: Description of `lmridge()` function arguments.

The `lmridge()` function returns an object of class "lmridge". The function `summary()`,`kest()`, and `kstats1()` etc., are used to compute and print a summary of the RR results, list of biasing parameter given in Table 4, and ridge related statistics such as estimated squared bias, R^2 and variance etc., after addition of k to diagonal of $X'X$ matrix. An object of class "lmridge" is a list, the components of which are described in Table 6:

Table 7 lists the functions and methods available in **lmridge** package:

The lmridge package implementation in R

The use of **lmridge** is explained through examples by using the Hald dataset.

```
> library("lmridge")
> mod <- lmridge(y ~ X1 + X2 + X3 + X4, data = as.data.frame(Hald),
+  scaling = "sc", K = seq(0, 1, 0.001))
```

The output of linear RR from **lmridge()** function is assigned to an object `mod`. The first argument of the function is `formula`, which is used to specify the required linear RR model for the data provided as second argument. The `print` method for `mod`, an object of class "lmridge", will display the de-scaled coefficients. The output (de-scaled coefficients) from the above command is only for a few selected biasing parameter values.

Object	Description
coef	A named vector of fitted ridge coefficients.
xscale	The scales used to standardize the predictors.
xs	The scaled matrix of predictors.
y	The centered response variable.
Inter	Whether an intercept is included in the model or not.
K	The RR biasing parameter(s).
xm	A vector of means of design matrix X.
rfit	Matrix of ridge fitted values for each biasing parameter k.
d	Singular values of the SVD of the scaled predictors.
div	Eigenvalues of scaled regressors for each biasing parameter k.
scaling	The method of scaling used to standardized the predictors.
call	The matched call.
terms	The `terms` object used.
Z	A matrix $(X'X + kI_p)^{-1}X'$ for each biasing parameter.

Table 6: Objects from "lmridge" class.

```
Call:
lmridge.default(formula = y ~ ., data = as.data.frame(Hald),
K = seq(0, 1, 0.001))
       Intercept      X1      X2       X3       X4
K=0.01  82.67556 1.31521 0.30612 -0.12902 -0.34294
K=0.05  85.83062 1.19172 0.28850 -0.21796 -0.35423
K=0.5   89.19604 0.78822 0.27096 -0.36391 -0.28064
K=0.9   90.22732 0.65351 0.24208 -0.34769 -0.24152
K=1     90.42083 0.62855 0.23540 -0.34119 -0.23358
```

To get the ridge scaled coefficients `mod$coef` can be used,

```
> mod$coef
       K=0.01    K=0.05      K=0.5      K=0.9        K=1
X1  26.800306  24.28399  16.061814  13.316802  12.808065
X2  16.500987  15.55166  14.606166  13.049400  12.689060
X3  -2.862655  -4.83610  -8.074509  -7.714626  -7.570415
X4 -19.884534 -20.53939 -16.272482 -14.004088 -13.543744
```

Objects of class "lmridge" contain components such as `rfit`, `K` and `coef` etc. For fitted ridge model, the generic method `summary()` is used to investigate the ridge coefficients. The parameter estimates of ridge model are summarized using a matrix of 5 columns namely *estimates*, *estimates (Sc)*, *StdErr (Sc)*, *t values (Sc)* and *P(>|t|)* for ridge coefficients. The following results are shown only for $K = 0.012$ which produces minimum MSE as compared to others values specified in the argument.

```
> summary(mod)
Call:
lmridge.default(formula = y ~ ., data = as.data.frame(Hald), K = 0.012)

Coefficients: for Ridge parameter K= 0.012
          Estimate Estimate (Sc) StdErr (Sc) t-value (Sc)  Pr(>|t|)
Intercept   83.1906     -246.5951    269.2195       -0.916  0.3837
X1           1.3046       26.5843      3.8162        6.966  0.0001 ***
X2           0.3017       16.2649      4.6337        3.510  0.0067 ***
X3          -0.1378       -3.0585      3.7655       -0.812  0.4377
X4          -0.3470      -20.1188      4.7023       -4.279  0.0021 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Ridge Summary
     R2    adj-R2  DF ridge         F       AIC       BIC
0.96990   0.95980   3.04587 134.14893  23.24068  58.30578
Ridge minimum MSE= 390.5195 at K= 0.012
P-value for F-test ( 3.04587 , 9.779581 ) = 2.914733e-08
```

Functions	Description
Ridge coefficient estimation and testing	
`lmridgeEst()`	The main model fitting function for implementation of RR models in R.
`coef()`	Display de-scaled ridge coefficients.
`lmridge()`	Generic function and default method that calls `lmridgeEst()` and returns an object of S3 class "lmridge" with different set of methods to standard generics. It has a print method for display of ridge de-scaled coefficients.
`summary()`	Standard RR output (coefficient estimates, scaled coefficients estimates, standard errors, t values and p values); returns an object of class "summaryridge" containing the relative summary statistics and has a print method.
Residuals, fitted values and prediction	
`predict()`	Produces predicted value(s) by evaluating `lmridgeEst()` in the frame `newdata`.
`fitted()`	Displays ridge fitted values for observed data.
`residuals()`	Displays ridge residuals values.
`press()`	Generic function that computes prediction residual error sum of squares (PRESS) for ridge coefficients.
Methods to estimate k	
`kest()`	Displays various k (biasing parameter) values from different authors available in literature and have a print method.
Ridge statistics	
`vcov()`	Displays associated Var-Cov matrix with matching ridge parameter k values
`hatr()`	Generic function that displays hat matrix from RR.
`infocr()`	Generic function that compute information criteria AIC and BIC.
`vif()`	Generic function that computes VIF values.
`rstats1()`	Generic function that displays different statistics of RR such as MSE, squared bias and R^2 etc., and have print method.
`rstats2()`	Generic function that displays different statistics of RR such as df, m-scale and LSRM etc., and have print method.
Ridge plots	
`plot()`	Ridge and VIF trace plot against biasing parameter k.
`bias.plot()`	Bias-Variance tradeoff plot. Plot of ridge MSE, bias and variance against k
`cv.plot()`	Cross validation plots of CV and GCV against biasing parameter k.
`info.plot()`	Plot of AIC and BIC against k.
`isrm.plot()`	Plots ISRM and m-scale measure.
`rplots.plot()`	Miscellaneous ridge related plots such as df-trace, RSS and PRESS plots.

Table 7: Functions and methods in **lmridge** package.

The `summary()` function also displays ridge related R^2, adjusted-R^2, df, F statistics, AIC, BIC and minimum MSE at certain k given in `lmridge()`.

The `kest()` function, which works with ridge fitted model, computes different biasing parameters developed by researchers, see Table 4. The list of different `k` values (22 in numbers) may help in deciding the amount of bias needs to be introduced in RR.

```
> kest(mod)

Ridge k from different Authors
                              k values
Thisted (1976):                0.00581
Dwividi & Srivastava (1978):   0.00291
LW (lm.ridge)                  0.05183
LW (1976)                      0.00797
HKB (1975)                     0.01162
Kibria (2003) (AM)             0.28218
```

```
Minimum GCV at                0.01320
Minimum CV at                 0.01320
Kibria 2003 (GM):             0.07733
Kibria 2003 (MED):            0.01718
Muniz et al. 2009 (KM2):     14.84574
Muniz et al. 2009 (KM3):      5.32606
Muniz et al. 2009 (KM4):      3.59606
Muniz et al. 2009 (KM5):      0.27808
Muniz et al. 2009 (KM6):      7.80532
Mansson et al. 2012 (KMN8):  14.98071
Mansson et al. 2012 (KMN9):   0.49624
Mansson et al. 2012 (KMN10):  6.63342
Mansson et al. 2012 (KMN11):  0.15075
Mansson et al. 2012 (KMN12):  8.06268
Dorugade et al. 2010:         0.00000
Dorugade et al. 2014:         0.00000
```

The `rstats1()` and `rstats2()` functions can be used to compute different statistics for a given ridge biasing parameter specified in a call to `lmridge`. The ridge statistics are MSE, squared bias, F statistics, ridge variance, degrees of freedom by Hastie and Tibshirani (1990), condition numbers, PRESS, R^2, and ISRM etc. Following are the results using `rstats1()` and `rstats2()` functions, for some ($K = 0$, 0.012, 0.1, 0.2).

```
> rstats1(mod)
Ridge Regression Statistics 1:

         Variance   Bias^2       MSE rsigma2        F     R2 adj-R2        CN
K=0     3309.5049   0.0000 3309.5049  5.3182 125.4142 0.9824 0.9765 1376.8806
K=0.012   72.3245 318.1951  390.5195  4.9719 134.1489 0.9699 0.9598  164.9843
K=0.1     19.8579 428.4112  448.2692  5.8409 114.1900 0.8914 0.8552   22.9838
K=0.2     16.5720 476.8887  493.4606  7.6547  87.1322 0.8170 0.7560   12.0804

> rstats2(mod)
Ridge Regression Statistics 2:

             CK DF ridge     EP    REDF      EF   ISRM m scale    PRESS
K= 0     6.0000    4.0000 4.0000  9.0000  0.0000 3.9872  0.0000 110.3470
K= 0.012 4.8713    3.0459 3.2204  9.7796 10.1578 3.6181  0.9541  92.8977
K= 0.1   4.2246    2.5646 2.9046 10.0954  7.6829 2.8471  1.4354 121.2892
K= 0.2   3.8630    2.2960 2.7290 10.2710  6.9156 2.5742  1.7040 162.2832
```

The residuals, fitted values from the RR and predicted values of the response variable y can be computed using functions `residual()`, `fitted()` and `predict()`, respectively. To obtain the *Var-Cov* matrix, VIF and Hat matrix, the function `vcov()`, `vif()` and `hatr()` can be used. The df are computed by following Hastie and Tibshirani (1990). The results for VIF, *Var-Cov* and diagonal elements of the hat matrix from `vif()`, `vcov()` and `hatr()` functions are given below for $K = 0.012$.

```
> hatr(mod)
> hatr(mod)[[2]]
> diag(hatr(mod)[[2]])
> diag(hatr(lmridge(y ~ ., as.data.frame(Hald), K = c(0, 0.012)))[[2]])
      1       2       3       4       5       6       7       8       9      10      11
0.39680 0.21288 0.10286 0.16679 0.24914 0.04015 0.28424 0.30163 0.12502 0.58426 0.29625
     12      13
0.12291 0.16294

> vif(mod)
              X1        X2       X3        X4
k=0     38.49621 254.42317 46.86839 282.51286
k=0.012  2.92917   4.31848  2.85177   4.44723
k=0.1    1.28390   0.51576  1.20410   0.39603
k=0.2    0.78682   0.34530  0.75196   0.28085

R> vcov(mod)
$`K=0.012`
```

```
          X1        X2        X3        X4
X1 14.563539  1.668783 11.577483  4.130232
X2  1.668783 21.471027  3.066958 19.075274
X3 11.577483  3.066958 14.178720  4.598000
X4  4.130232 19.075274  4.598000 22.111196
```

Following are possible uses of some functions to compute different ridge related statistics. For detail description of these functions/ commands, see the **lmridge** package documentation.

```
> mod$rfit
> resid(mod)
> fitted(mod)
> infocr(mod)
> press(mod)
```

For given values of X, such as for first five rows of X matrix, the predicted values for some $K = 0, 0.012, 0.1,$ and 0.2 will be computed by `predict()`:

```
> predict(mod, newdata = as.data.frame(Hald[1 : 5, -1]))
        K=0   K=0.012     K=0.1     K=0.2
1  78.49535  78.52225  79.75110  80.73843
2  72.78893  73.13500  74.32678  75.38191
3 105.97107 106.39639 106.04958 105.62451
4  89.32720  89.48443  89.52343  89.65432
5  95.64939  95.73595  96.56710  96.99781
```

The model selection criteria's of AIC and BIC can be computed using `infocr()` function for each value of K used in argument of `ridge()`. For some $K = 0, 0.012, 0.1,$ and 0.2, the AIC and BIC values are:

```
> infocr(mod)
               AIC      BIC
K=0       24.94429 60.54843
K=0.012   23.24068 58.30578
K=0.1     24.78545 59.57865
K=0.2     27.98813 62.62961
```

The effect of multicollinearity on the coefficient estimates can be identified by using different graphical displays such as ridge, VIF and df traces, plotting of RSS against df, PRESS vs k, and the plotting of bias, variance, and MSE against K etc. Therefore, for selection of optimal k using subjective (judgmental) methods, different plot functions are also available in **lmridge** package. For example, the ridge (Figure 1) or vif trace (Figure 2) can be plotted using `plot()` function. The argument to plot functions are `abline = TRUE`, and `type = c("ridge","vif")`. By default, ridge trace will be plotted having horizontal line parallel to horizontal axis at $y = 0$ and vertical line on x-axis at k having minimum GCV.

```
> mod <- lmridge(y ~ ., data = as.data.frame(Hald), K = seq(0, 0.5, 0.001))
> plot(mod)
> plot(mod, type = "vif", abline = FALSE)
> plot(mod, type = "ridge", abline = TRUE)


> bias.plot(mod, abline = TRUE)
> info.plot(mod, abline = TRUE)


> cv.plot(mod, abline = TRUE)
```

The vertical lines in ridge trace and VIF trace suggest the optimal value of biasing parameter k selected at which GCV is minimum. The horizontal line in ridge trace is reference line at $y = 0$ for ridge coefficient against vertical axis .

The bias-variance tradeoff plot (Figure 3) may be used to select optimal k using `bias.plot()` function. The vertical line in bias-variance tradeoff plot shows the value of biasing parameter k and horizontal line shows minimum MSE for ridge.

The plot of model selection criteria AIC and BIC for choosing optimal k (Figure 4), `info.plot()` function may be used,

Function `cv.plot()` plots the CV and GCV cross validation against biasing parameter k for the optimal selection of k (see Figure 5), that is,

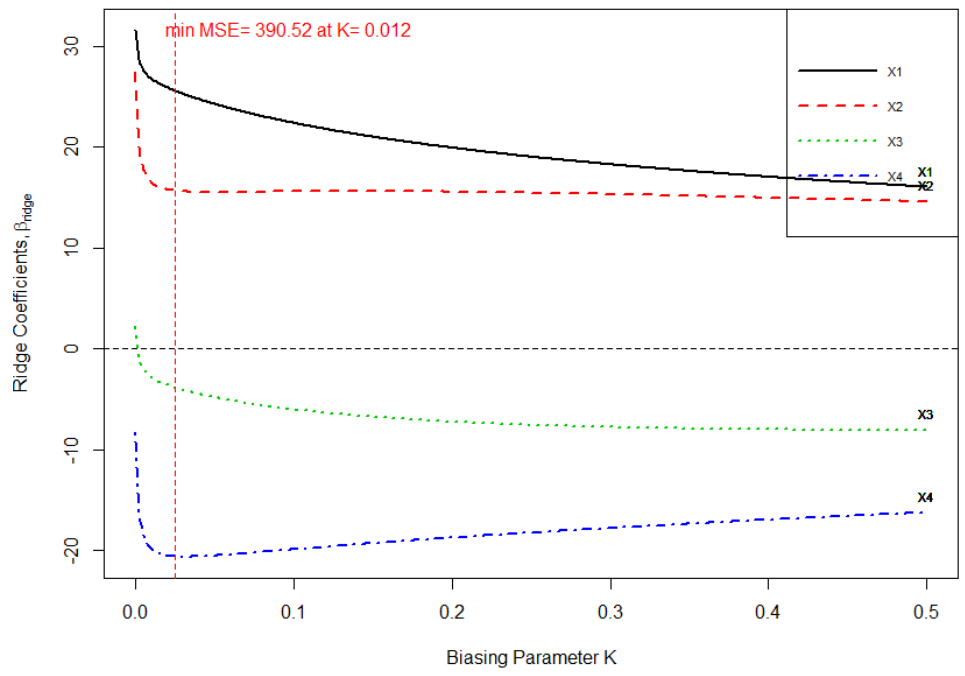


Figure 1: Ridge trace plot.

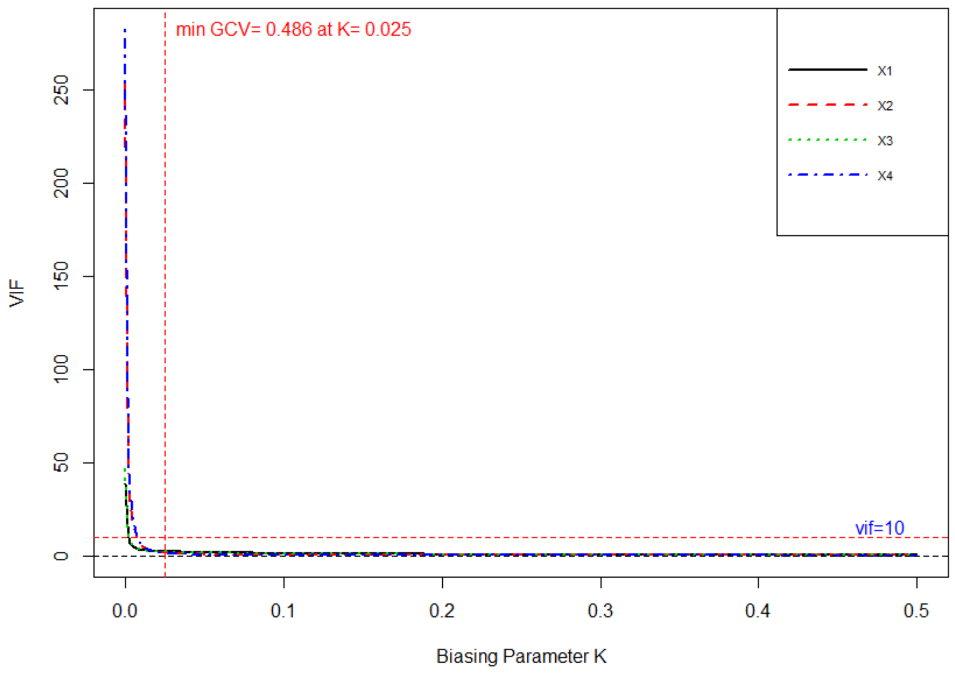


Figure 2: VIF trace.

```
> isrm.plot(mod)
```

The *m*-scale and ISRM (Figure 6) measures by Vinod (1976) can also be plotted from function of `isrm.plot()` and can be used to judge the optimal value of *k*.

Function `rplots.plot()` plots the panel of three plots namely (i) df trace, (ii) RSS vs *k* and (iii) PRESS vs *k* and may be used to judge the optimal value of *k*, see Figure 7.

```
> rplots.plot(mod)
```

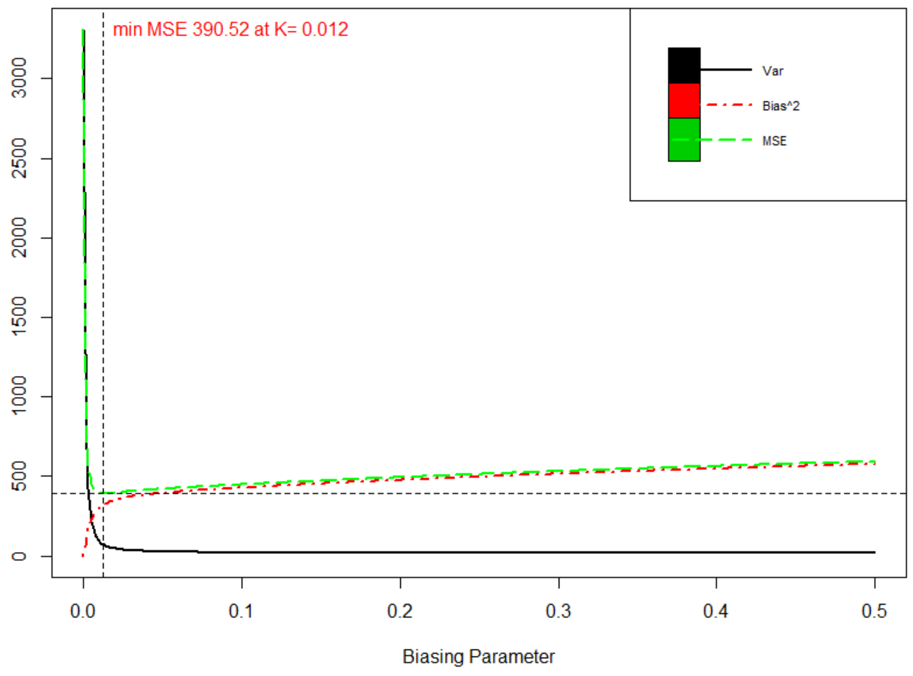


Figure 3: Bias-variance trade-off.

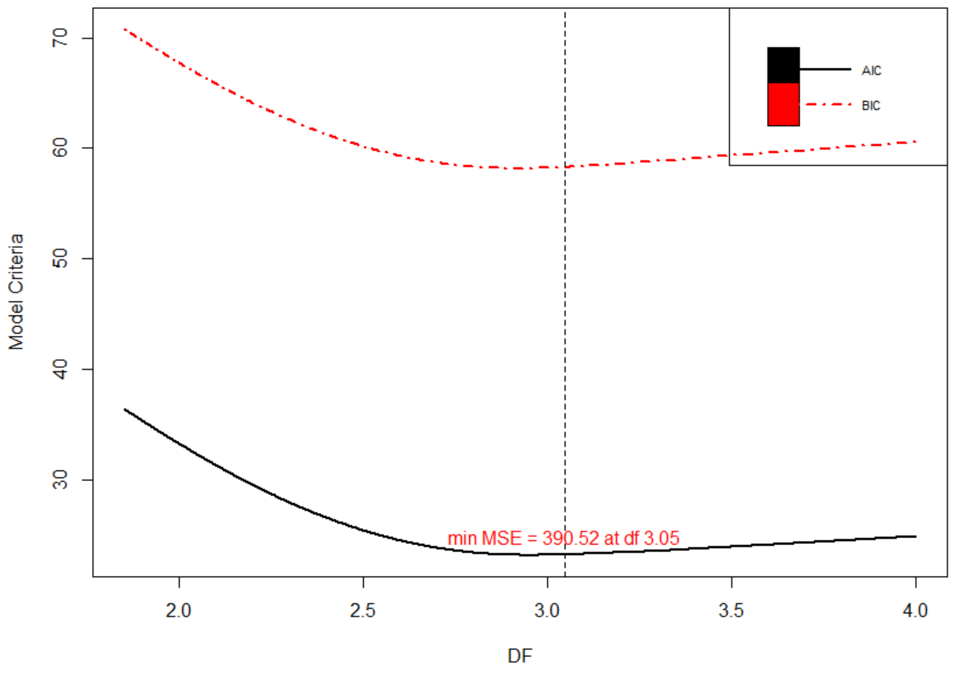


Figure 4: Information criteria plot (AIC and BIC).

Summary

Our developed **lmridge** package provides the most complete suite of tools for RR available in R, comparable to those available as listed in Table 1. We have implemented functions to compute the ridge coefficients, testing of these coefficients, computation of different ridge related statistics and computation of the biasing parameter for different existing methods by various authors (see Table 4).

We have greatly increased the ridge related statistics and different graphical methods for the selection of biasing parameter k through **lmridge** package in R.

Up to now, a complete suite of tools for RR was not available for an open source or paid version of statistical software packages, resulting in reduced awareness and use of developed ridge related statistics. The package **lmridge** provides a complete open source suite of tools for the computation of

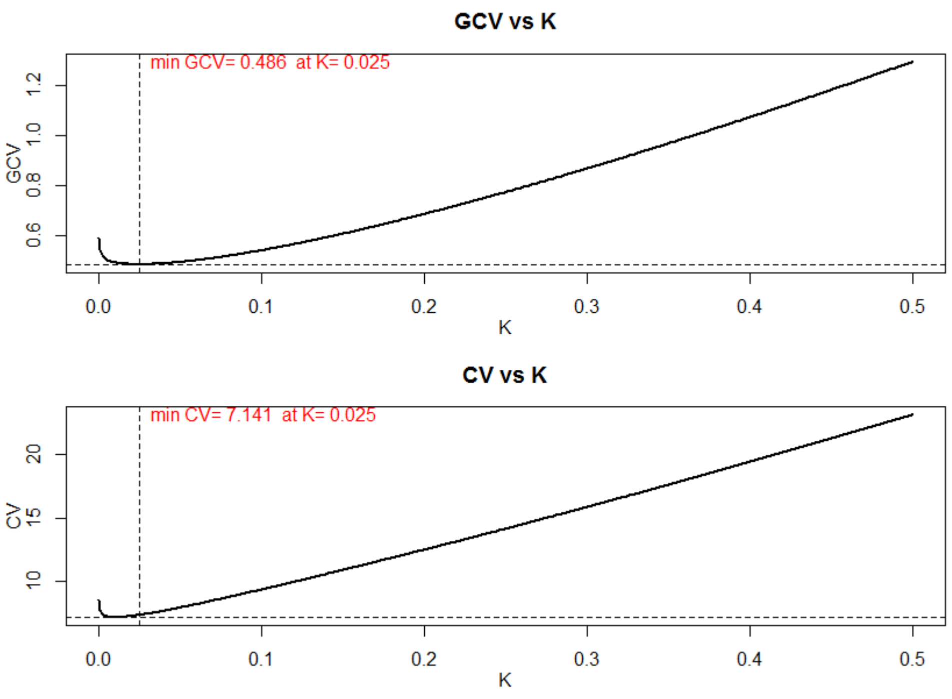


Figure 5: Cross-validation plots (CV and GCV).

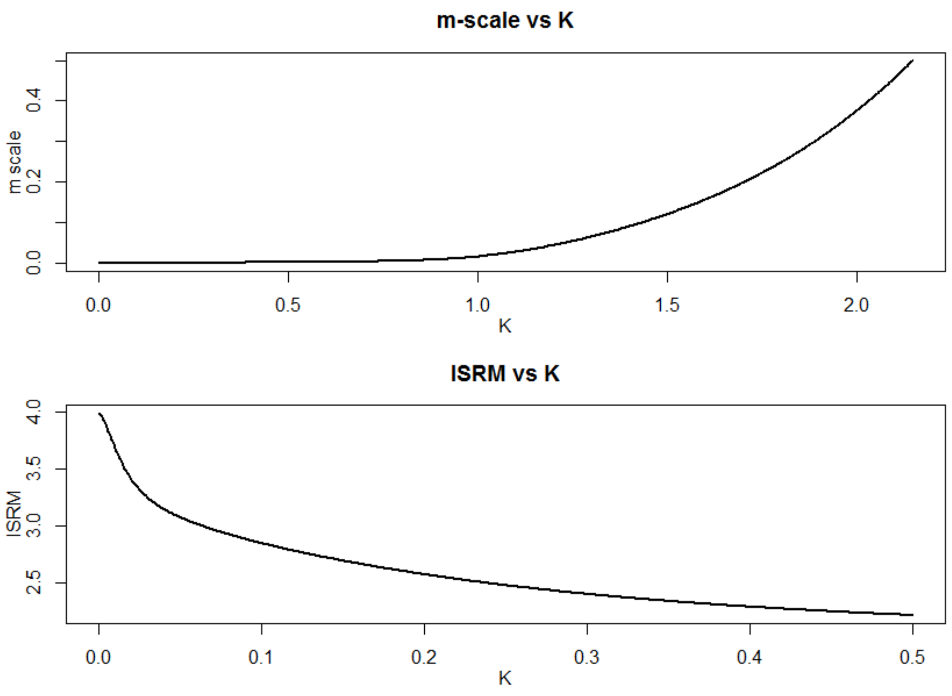


Figure 6: m-scale and ISRM plot.

ridge coefficients estimation, testing and computation of different statistics. We believe the availability of these tools will lead to increase utilization and better ridge related practices.

Bibliography

H. Akaike. A New Look at the Statistical Model Identification. *IEEE Transaction on Automatic Control*, 9 (6):716–723, 1973. URL `https://doi.org/10.1109/TAC.1974.1100705`. [p332]

M. A. Alkhamisi and G. Shukur. A Monte Carlo Study of Recent Ridge Parameters. *Communica-*

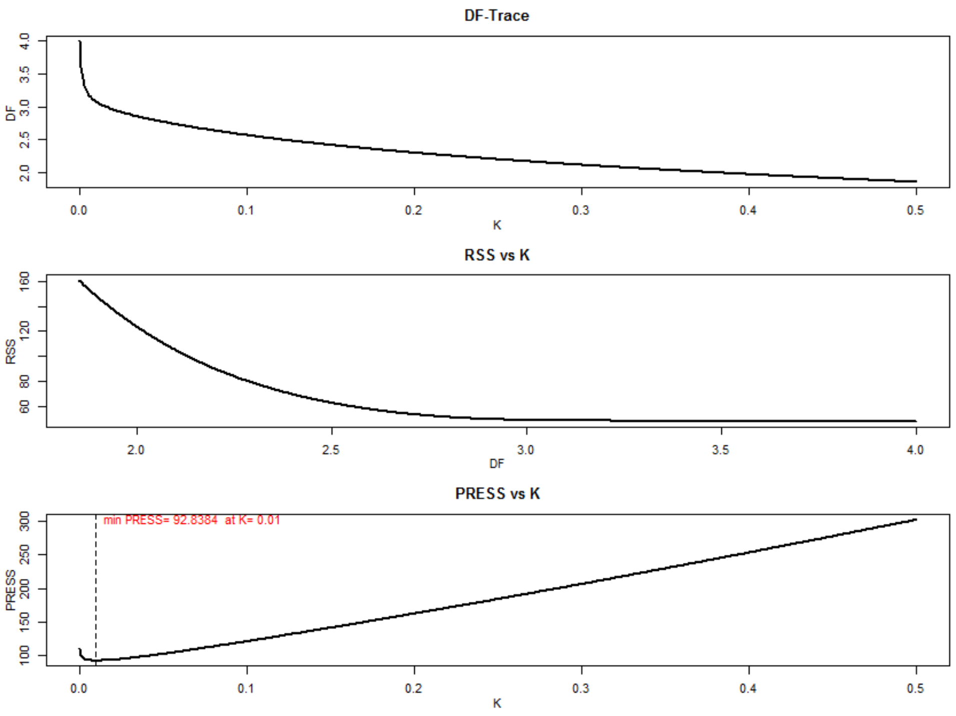


Figure 7: Miscellaneous ridge plots.

tions in Statistics-Simulation and Computation, 36(3):535–547, 2007. URL https://doi.org/10.1080/03610910701208619. [p330]

M. A. Alkhamisi, G. Khalaf, and G. Shukur. Some Modifications for Choosing Ridge Parameter. *Communications in Statistics-Theory and Methods*, 35(11):2005–2020, 2006. URL https://doi.org/10.1080/03610920600762905. [p330]

D. M. Allen. Mean Square Error of Prediction as a Criterion for Selecting Variables. *Technometrics*, 13(3):469–475, 1971. URL http://doi.org/10.2307/1267161. [p332]

D. M. Allen. The Relationship Between Variable Selection and Data Augmentation and Method for Prediction. *Technometrics*, 16(1):125–127, 1974. URL http://doi.org/10.2307/1267500. [p330, 332]

S. Arumairajan and P. Wijekoon. Optimal generalized biased estimator in linear regression model. *Open Journal of Statistics*, 5:403–411, 2015. URL https://doi.org/10.4236/ojs.2015.55042. [p327]

M. Aslam. Using Heteroscedasticity-Consistent Standard Errors for the Linear Regression Model with Correlated Regressors. *Communications in Statistics-Simulation and Computation*, 43(10):2353–2373, 2014. URL https://doi.org/10.1080/03610918.2012.750354. [p332]

D. A. Belsley. A Guide to Using the Collinearity Diagnostics. *Computer Science in Economics and Management*, 4(1):33–50, 1991. URL https://doi.org/10.1007/BF00426854. [p334]

D. A. Belsley, E. Kuh, and R. E. Welsch. *Diagnostics: Identifying Influential Data and Sources of Collinearity*. John Wiley & Sons, New York, 1980. chap. 3. [p328, 330]

S. Chatterjee and A. S. Hadi. *Regression Analysis by Example*. John Wiley & Sons, 4th edition, 2006. [p328, 330]

E. Cule and M. De Iorio. A Semi-Automatic Method to Guide the Choice of Ridge Regression. *Annals of Applied Statistics*, arxiv:1205.0686v1, 2012. URL https://arxiv.org/abs/1205.0686v1. [p328, 334]

J. D. Curto and J. C. Pinto. The Corrected VIF (CVIF). *Journal of Applied Statistics*, 38(7):1499–1507, 2011. URL https://doi.org/10.1080/02664763.2010.505956. [p328]

N. J. Delaney and S. Chatterjee. Use of the Bootstrap and Cross-Validation in Ridge Regression. *Journal of Business & Economic Statistics*, 4(2):255–262, 1986. URL http://doi.org/10.2307/1391324. [p332]

A. Dissanayake, P. Wijekoon, and R-Core. lmrest: Different Types of Estimators to Deal with Multicollinearity, 2016. URL https://cran.r-project.org/package=lrmest.Rpackageversion3.0. [p327]

A. V. Dorugade. New Ridge Parameters for Ridge Regression. *Journal of the Association of Arab Universities for Basic and Applied Sciences*, 15:94–99, 2014. URL https://doi.org/10.1016/j.jaubas.2013.03.005. [p333]

A. V. Dorugade and D. N. Kashid. Alternative Method for Choosing Ridge Parameter for Regression. *Applied Mathematical Sciences*, 4(9):447–456, 2010. [p333]

N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley & Sons, New York, 2nd edition, 1998. [p330, 334]

T. D. Dwividi and V. K. Shrivastava. On the Mimimum Mean Square Error Estimators in a Regression Model. *Communications in Statistics-Theory and Methods*, 7(5):487–494, 1978. URL https://doi.org/10.1080/03610927808827642. [p333]

J. B. Endelman. Ridge Regression and Other Kernels for Genomic Selection with R Package rrBLUP. *Plant Genome*, 4(3):250–255, 2011. URL http://doi.org/10.3835/plantgenome2011.08.0024. [p328]

D. E. Farrar and R. R. Glauber. Multicollinearity in Regression Analysis: The Problem Revisted. *The Review of Economics and Statistics*, 49(1):92–107, 1967. URL http://doi.org/10.2307/1937887. [p328]

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks, CA, 2nd edition, 2011. URL https://socialsciences.mcmaster.ca/jfox/Books/Companion. [p328]

J. Friedman, T. Hastie, and R. Tibshirani. Regulaziation Paths for Generalized Linear Models via Coordinate Decent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL http://doi.org/10.18637/jss.v033.i01. [p328]

M. Friendly. genridge: Generalized Ridge Trace Plots for Ridge Regression, 2017. URL https://cran.r-project.org/web/packages/genridge. R package version 0.6-6. [p328]

J. J. Goeman, R. J. Meijer, and N. Chaturvedi. Penalized: L1 (Lasso and Fused Lasso) and L2 (Ridge) Penalized Estimation in GLMs and in the Cox Model, 2017. URL https://cran.r-project.org/web/packages/penalized. R package version 0.9-50. [p327]

G. H. Golub, M. Heath, and G. Wahba. Generalized Cross Validation as a Method for Choosing a Good Ridge Parameter. *Technometrics*, 21(2):215–223, 1979. URL http://doi.org/10.2307/1268518. [p328, 332]

W. H. Greene. *Econometric Analysis*. Prentic Hall, New Jersey, 5th edition, 2002. [p328]

D. K. Guilkey and J. L. Murphy. Direct Ridge Regression Techniques in Cases of Multicollinearity. *Journal of American Statistical Association*, 70(352):769–775, 1975. URL http://doi.org/10.1080/01621459.1975.10480301. [p330]

R. F. Gunst and R. L. Mason. Advantages of Examining Multicollinearities in Regression Analysis. *Biometrics*, 33(1):249–260, 1977. [p328]

W. Hadley. *R Packages: Organize, Test, Document, and Share Your Code*. O'Reilly Media, 2015. [p326]

A. M. Halawa and M. Y. El-Bassiouni. Tests of Regression Coefficients Under Ridge Regression Models. *Journal of Statistical-Computation and Simulation*, 65(1–4):341–356, 2000. URL https://doi.org/10.1080/00949650008812006. [p332, 334]

A. Hald. *Statistical Theory with Engineering Applications*. John Wiley & Sons, 1952. [p328]

T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman & Hall, 1990. [p334, 337]

W. J. Hemmerle. An Explicit Solution for Generalized Ridge Regression. *Technometrics*, 17(3):309–314, 1975. URL http://doi.org/10.2307/1268066. [p330]

W. J. Hemmerle and T. F. Brantle. Explicit and Constrained Generalized Ridge Estimation. *Technometrics*, 20(2):109–120, 1978. URL http://doi.org/10.2307/1268701. [p330]

R. R. Hocking, F. M. Speed, and M. J. Lynn. A Class of Biased Estimators in Linear Regression. *Technometrics*, 18(4):425–437, 1976. URL http://doi.org/10.2307/1268658. [p330]

A. E. Hoerl. Optimum Solution of Many Variables Equations. *Chemical Engineering Progress*, 55:67–78, 1959. [p329]

A. E. Hoerl. Application of Ridge Analysis to Regression Problems. *Chemical Engineering Progress*, 58: 54–59, 1962. URL http://doi.org/10.1002/sim.4780030311. [p329]

A. E. Hoerl. Ridge Analysis. *Chemical Engineering Progress Symposium Series*, 60:67–77, 1964. [p329]

A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation of Nonorthogonal Problems. *Technometrics*, 12(1):55–67, 1970a. URL http://doi.org/10.2307/1267351. [p326, 328, 329, 330, 333]

A. E. Hoerl and R. W. Kennard. Ridge Regression: Application to Nonorthogonal Problems. *Technometrics*, 12(1):69–82, 1970b. URL http://doi.org/10.2307/1267352. [p329, 330]

A. E. Hoerl, R. W. Kennard, and K. F. Baldwin. Ridge Regression: Some Simulations. *Communications in Statistics*, 4(2):105–123, 1975. URL https://doi.org/10.1080/03610927508827232. [p328]

M. U. Imdad and M. Aslam. mctest: An R Package for Detection of Collinearity Among Regressors, 2018a. URL https://cran.r-project.org/web/packages/mctest/. R package version 1.2. [p328]

M. U. Imdad and M. Aslam. lmridge: Linear Ridge Regression with Ridge Penalty and Ridge Statistics, 2018b. URL https://cran.r-project.org/package=lmridge. R package version 1.2. [p326]

M. Imdadullah, M. Aslam, and S. Altaf. mctest: An R Package for Detection of Collinearity Among Regressors. *The R Journal*, 8(2):495–505, 2016. URL https://journal.r-project.org/archive/accepted/imdadullah-aslam-altaf.pdf. [p328]

S. I. Inc. The SAS System for Windows: Release 9.2, 2011. SAS Inst., Cary, NC. [p326]

B. Kan-Kilinc and O. Alpu. ltsbase: Ridge and Liu Estimates Based on LTS (Least Trimmed Squares) Method, 2013. URL https://CRAN.R-project.org/package=ltsbase. R package version 1.0.1. [p327]

B. Kan-Kilinc and O. Alpu. Combining Some Biased Estimation Methods with Least Trimmed Squares Regression and Its Application. *Revista Colombiana de Estadística*, 38(2):485–502, 2015. URL http://dx.doi.org/10.15446/rce.v38n2.51675. [p327]

J. D. Kasarda and W. F. Shih. Optimal Bias in Ridge Regression Approaches to Multicollinearity. *Sociologial Methods and Research*, 5(4):461–470, 1977. URL https://doi.org/10.1177/004912417700500405. [p330]

M. G. Kendall. *A Course in Multivariate Analysis*. Griffin, London, 1957. pp. 70–75. [p328]

R. W. Kennard. A Note on the Cp-Statistics. *Technometrics*, 13(4):899–900, 1971. URL https://doi.org/10.1080/00401706.1971.10488863. [p332]

G. Khalaf. A Comparison Between Biased and Unbiased Estimators. *Journal of Modern Applied Statistical Methods*, 12(2):293–303, 2013. URL http://10.22237/jmasm/1383279360. [p330]

G. Khalaf and G. Shukur. Choosing Ridge Parameter for Regression Problems. *Communications in Statistics-Theory and Methods*, 34(5):1177–1182, 2005. URL https://doi.org/10.1081/STA-200056836. [p330]

B. M. G. Kibria. Performance of Some New Ridge Regression Estimators. *Communications in Statistics-Simulation and Computation*, 32(2):419–435, 2003. URL https://doi.org/10.1081/SAC-120017499. [p330, 333]

L. R. Klein. *An Introduction to Econometrics*. Prentic Hall, Englewood, Cliffs, N. J., 1962. pp. 101. [p328]

J. Kmenta. *Elements of Econometrics*. Macmillan Publishing Company, New York, 2nd edition, 1980. pp. 431. [p326]

A. Koutsoyiannis. *Theory of Econometrics*. Macmillan Education Limited, 1977. [p328]

P. Kovács, T. Petres, and Tóth. A New Measure of Multicollinearity in Linear Regression Models. *International Statistical Review / Revue Internationale de Statistique*, 73(3):405–412, 2005. URL http://doi.org/10.1111/j.1751-5823.2005.tb00156.x. [p328]

J. F. Lawless and P. Wang. A Simulation Study of Ridge and Other Regression Estimators. *Communications in Statistics-Theory and Methods*, 5(4):307–323, 1976. URL https://doi.org/10.1080/03610927608827353. [p330, 333]

W. F. Lee. Model Estimation Using Ridge Regression with the Variance Normalization Criterion. Master thesis, Department of Educational Foundation, Memorial University of Newfoundland, 1979. [p332]

F. Leisch. *Creating R Packages: A Tutorial*. Compstat 2008-Proceedings in Computational Statistics, Physica Verlage, Heidelberg, Germay, 2008, 2008. URL ftp://cran.r-project.org/pub/R/doc/contrib/Leisch-CreatingPackages.pdf. [p326]

G. S. Maddala. *Introduction to Econometrics*. Macmillan Publishing Company, New York, 1988. [p328]

C. L. Mallows. Some Comments on Cp. *Technometrics*, 15(4):661–675, 1973. URL http://doi.org/10.2307/1267380. [p332]

D. W. Marquardt. Generalized Inverses, Ridge Regression, Biased Linear Estimation, and Nonlinear Estimation. *Technometrics*, 12(3):591–612, 1970. URL http://doi.org/10.2307/1267205. [p328, 330]

B. T. McCallum. Artificial Orthogonalization in Regression Analysis. *Review of Economics and Statistics*, 52(1):110–113, 1970. URL http://doi.org/10.2307/1927606. [p330]

G. C. McDonald and D. I. Galarneau. A Monte Carlo Evaluation of Some Ridge-Type Estimators. *Journal of the American Statistical Association*, 70(350):407–416, 1975. URL http://doi.org/10.2307/2285832. [p330]

D. C. Montgomery and E. A. Peck. *Introduction to Linear Regression Analysis*. John Wiley & Sons, New York, 1982. [p326]

S. Moritz and E. Cule. ridge: Ridge Regression with Automatic Selection of the Penalty Parameter, 2017. URL https://cran.r-project.org/web/packages/ridge. R package version 2.2. [p326]

G. Muniz and B. M. G. Kibria. On Some Ridge Regression Estimators: An Empirical Comparisons. *Communications in Statistics-Simulation and Computation*, 38(3):621–630, 2009. URL https://doi.org/10.1080/03610910802592838. [p333]

G. Muniz, B. M. G. Kibria, K. Mansson, and G. Shukur. On Developing Ridge Regression Parameters: A Graphical Investigation. *Statistics and Operations Research Transactions*, 36(2):115–138, 2012. [p333]

R. H. Myers. *Classical and Modern Regression with Application*. PWS-KENT Publishing Company, 2 edition, 1986. [p326]

NCSS 11 Statistical Software. Ridge Regression, 2016. URL http://ncss.wpengine.netdna-cdn.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf. NCSS, LLC. Kaysville, Utah, USA. [p326]

J. P. Newhouse and S. D. Oman. An Evaluation of Ridge Estimators. Rand Report, R-716-PR, 1971. [p330]

L. Nordberg. A Procedure for Determination of a Good Ridge Parameter in Linear Regression. *Communications in Statistics-Simulation and Computation*, 11(3):285–309, 1982. URL https://doi.org/10.1080/03610918208812264. [p330]

B. Obenchain. RXshrink: Maximum Likelihood Shrinkage via Generalized Ridge or Least Angle Regression, 2014. URL https://cran.r-project.org/web/packages/RXshrink. R package version 1.0-8. [p328]

R. L. Obenchain. Classical F-Tests and Confidence Regions for Ridge Regression. *Technometrics*, 19(4):429–439, 1977. URL http://doi.org/10.2307/1267882. [p330, 332]

A. Perperoglou. CoxRidge: Cox Models with Dynamic Ridge Penalties, 2015. URL https://cran.r-project.org/web/packages/CoxRidge. R package version 0.9.2. [p328]

B. S. Price. RidgeFusion: R Package for Ridge Fusion in Statistical Learning, 2014. URL https://cran.r-project.org/web/packages/RidgeFusion. R package version 1.0-3. [p328]

R Core Team. *Writing R Extensions*. R Foundation for Statistical Computing, 2015. Version R 3.2.3. [p326]

J. O. Rawlings, S. G. Pantula, and D. A. Dickey. *Applied Regression Analysis: A Research Tool*. Springer-Verlag, New York, 2nd edition, 1998. [p326]

S-PLUS. Insightful Corporation, Seattle, Wt., 2008. URL http://www.insightful.com. [p328]

A. K. M. E. Saleh and B. M. G. Kibria. Peformance of Some New Preliminary Test Ridge Regression Estimators and Their Properties. *Communications in Statistics-Theory and Methods*, 22(10):2747–2764, 1993. URL https://doi.org/10.1080/03610929308831183. [p330]

G. E. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978. URL https://doi.org/10.1214/aos/1176344136. [p332]

G. A. F. Seber and A. J. Lee. *Linear Regression Analysis*. John Wiley & Sons, New Jersey, 2 edition, 2003. [p326, 329]

B. Seifert. lpridge: Local Polynomial (Ridge) Regression, 2013. URL https://cran.r-project/web/packages/lpridge. R package version 1.0-7. [p328]

Shazam. *Reference Manual Version 11*, 2011. ISBN: 978-0-9570475-0-1. [p328]

X. Shen, M. Alam, F. Fikse, and L. Rönnegård. A Novel Generalized Ridge Regression Method for Quantitative Genetics. *Genetics*, 193(4):1255–1268, 2013. URL http://doi.org/10.1534/genetics.112.146720. [p328]

S. Singh and D. S. Tracy. Ridge-Regression Using Scrambled Responses. *Metrika*, LVII(1-2):147–157, 1999. [p330]

StataCorp. Stata Statistical Software: Release 8, 2014. URL http://www.stata.com. College Station, TX: StataCorp LP. [p327]

H. Theil. Specification Errors and the Estimation of Economic Relationships. *International Statistical Institute (ISI)*, 25(1/3):41–51, 1957. URL http://doi.org/10.2307/1401673. [p329]

H. Theil. *Principles of Econometrics*. John Wiley & Sons, New York, 1971. [p328]

R. A. Thisted. Ridge Regression, Minimax Estimation, and Empirical Bayes Methods. Technical Report 28, Stanford, Calif.: Division of Biostatistics, Stanford University, 1976. [p333]

R. E. Tripp. *Non-Stochastic Ridge Regression and Effective Rank of the Regressors Matrix*. Ph.d. thesis, Department of Statistic, Virginia Polytechnic Institute and State University., 1983. [p326]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, New York, 4th edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. ISBN 0-387-95457-0. [p326, 328, 333]

H. D. Vinod. Application of New Ridge Regression Methods to a Study of Bell System Scale Economics. *Journal of the American Statistical Association*, 71(356):835–841, 1976. URL http://doi.org/10.2307/2286847. [p330, 332, 339]

E. Wencheko. Estimation of the Signal-to-Noise in the Linear Regression Model. *Statistical Papers*, 41: 327–343, 2000. URL https://doi.org/10.1007/BF02925926. [p330]

D. W. Wichern and G. A. Churchill. A Comparison of Ridge Estimators. *Technometrics*, 20(3):301–311, 1978. URL http://doi.org/10.2307/1268139. [p330]

Muhammad Imdad Ullah
Ph.D scholar (Statistics),
Department of Statistics
Bahauddin Zakariya University, Multan, Pakistan
ORCiD ID: 0000-0002-1315-491X
mimdadasad@gmail.com

Muhammad Aslam
Associate Professor,
Department of Statistics
Bahauddin Zakariya University, Multan, Pakistan
aslamasadi@bzu.edu.pk

Saima Altaf
Assistant Professor,
Department of Statistics
Bahaudding Zakariya University, Multan, Pakistan
saimaaltaf@bzu.edu.pk

Geospatial Point Density

by Paul F. Evangelista and David Beskow

Abstract This paper introduces a spatial point density algorithm designed to be explainable, meaningful, and efficient. Originally designed for military applications, this technique applies to any spatial point process where there is a desire to clearly understand the measurement of density and maintain fidelity of the point locations. Typical spatial density plotting algorithms, such as kernel density estimation, implement some type of smoothing function that often results in a density value that is difficult to interpret. The purpose of the visualization method in this paper is to understand spatial point activity density with precision and meaning. The temporal tendency of the point process as an extension of the point density methodology is also discussed and displayed. Applications include visualization and measurement of any type of spatial point process. Visualization techniques integrate **ggmap** with examples from San Diego crime data.

Introduction

The **pointdensityP** package is an example of a tailored package that was created to visualize and quantify spatiotemporal point data (Evangelista and Beskow, 2018). The innovation in the presented methods lies in the implementation, simplification, and general improvements over other methods. Although designed for military analysis, the presented analysis and visualization methods work for any discrete point process that contains a spatial dimension. The point density method in this paper calculates event frequency intensity, or density, within the neighborhood of a given point, accounting for geospatial projection. Additionally, if the point process includes a temporal dimension, the temporal tendency can also be calculated. The temporal tendency is the average age of events within the neighborhood of each point, providing perspective on density trends over time.

The purpose of any type of map projection is simple—to understand a phenomenon within the context of our physical environment. The impetus of the work within this paper resulted from difficult-to-explain output from common spatial density or "hot spot" visualization techniques. Many of these spatial density approaches use a methodology that would be appropriate for a continuous valued phenomenon but does not apply well to discrete phenomenon. Many of the existing density visualization techniques apply a smoothing function that depicts activity where there should be no activity, essentially spreading the density across the area of interest without spatial specificity. Furthermore, many of the density approximation techniques, such as kernel density estimation, create an output value that is not easily interpreted. All too often the output simply provides a relative comparison, with colors representing values that indicate a higher or lower density without providing meaningful quantification of the measured phenomenon.

Related work

Spatial density measurement or "hot spot" analysis is widespread in geospatial literature. Baddeley et al. provide an authoritative overview of numerous spatial point pattern computational methods, to include density methods using **spatstat** (Baddeley et al., 2015). Within R, **kde2d** is probably the most commonly used spatial density function (Ripley et al., 2015; Venables and Ripley, 2002). This function creates a gaussian kernel density estimate for two-dimensional data. Wand and Jones discuss computing techniques for kernel estimators in Wand and Jones (1995) and Wand (1994), describing the technique of binning data prior to estimating the kernel density. The function **bkde2D** implements Wand's binned kernel density estimation technique (Wand and Ripley, 2014). The binning approach reduces the computing cost significantly. Assuming n data points that will create a density estimate for m points, the direct computation of the kernel estimate will be $O(nm)$. Binning maps n points to G grid points, where $G \ll n$, and reduces complexity to $O(Gm)$.

There are several problems related to kernel-based smoothing when applied to discrete spatio-temporal events. Ease of interpretation and outputs designed for a continuous valued process have already been mentioned. For the binned kernel density estimation, granularity of the mesh also quickly increases computational requirements. Kernel based smoothing calculates and returns density for every point in the mesh. Discrete event spatio-temporal phenomenon, especially social phenomenon, only apply to regions of space where people are active. For example, in the San Diego data analyzed in this paper, there is no need to calculate crime density tens of miles into the Pacific Ocean. However, the square mesh approach used in the kernel smoothing algorithms forces this calculation. The `pointdensity()` algorithm presented in this paper returns values only for event locations, improving clarity on both the location and density of event behavior. The following code creates an example of binned kernel density estimation, using **bkde2D**, applied to the San Diego crime data (San Diego Regional Data Library, 2015). Figure 1 contains the visual results.

```
SD <- read.table("incidents-5y.csv", sep = ",", header = TRUE)
x  <- cbind(SD$lon, SD$lat)
est <- bkde2D(x, bandwidth = c(.01, .01), gridsize = c(750, 800),
  range.x = list(c(-117.45, -116.66), c(32.52, 33.26)))
BKD_df <- data.frame(lat = rep(est$x2, each = 750), lon = rep(est$x1, 800),
  count = c(est$fhat))
map_base <- qmap(location = "32.9,-117.1", zoom = 10, darken = 0.3)
map_base + stat_contour(bins = 150, geom = "polygon", aes(x = lon, y = lat, z = count,
  fill = ..level..), data = BKD_df, alpha = 0.15)
  + scale_fill_continuous(name = "density", low = "green", high = "red")
```

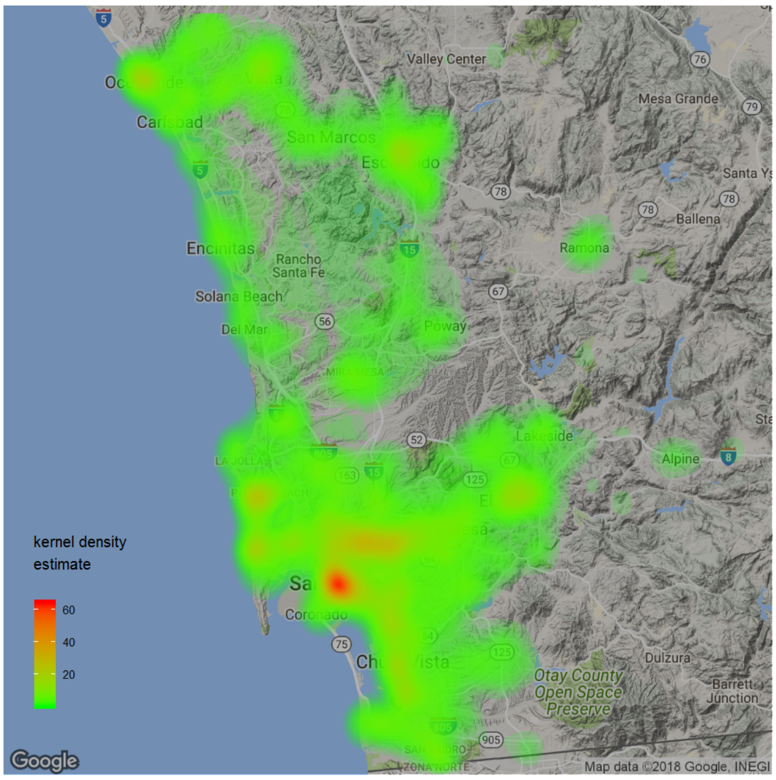


Figure 1: Two-dimensional binned kernel density estimation applied to San Diego crime data.

The method proposed in this paper requires two parameters:

- g, a grid size that represents the fraction of latitude and longitude degree separation in the grid or binning system that will be used for aggregation, and
- r, a positive integer value that represents the radius in grid steps for the neighborhood of interest, which yields a complexity of $O(n(2r)2)$.

Typically, $(2r)^2 \ll G$. As a comparison, Figure 1 resulted from **bkde2D** and compared 800,000 crime locations across a mesh that measured 800 x 750 = 600,000 = G, creating a mesh of points separated by approximately 0.1 km. In comparison, Figure 4 resulted from `pointdensity()` and uses a radius of 10 steps (1 kilometer total) and the same grid size of 0.1 km, comparing 800,000 crime locations each against a local neighborhood of $(2r)^2$ = 400 points. This reduced computational complexity achieved by the `pointdensity()` algorithm largely results from the advantage of assuming and managing an unbounded spatial extent.

Both **bkde2D** and `geom_bin2d()` from **ggplot2** require a bounded spatial extent. Density algorithms with a bounded spatial extent have two common disadvantages: they usually include unnecessary computations, and they always require pre-existing knowledge regarding the spatial extent. While this advantage is not completely analogous to the concept of unbounded data processing, there are similarities. Akidau et al. (2015) describe the fundamental aspects of unbounded data and the need for modern computing methods to accomodate this growing field. Since the `pointdensity()` algorithm processes data without the requirement for a spatial extent, it is possible to efficiently process data that includes significant spatial separation, a potentially useful trait depending upon the nature of the data under consideration.

The techniques discussed in this paper have been under development for several years and employed in several studies. Previous implementations involved other programming languages (Perl, Python) and leveraged ESRI's *ArcMap* to visualize results (Environmental Systems Research Institute (ESRI), 2018). Although plotting and graphics have long been a strong suit of R, geospatial visualization lagged behind many commercial geospatial information systems until the release of **ggmap** in 2011. **ggmap** is an R package where the authors present methods for plotting geospatial information by leveraging existing online map sources through their APIs and plotting the map as an image in R (Kahle and Wickham, 2013a,b). Similar to *ArcMap*, **ggmap** enables layers of geospatial data to be plotted over a georectified image for the sake of analysis and visualization. The aesthetic appeal of these downloaded images coupled with the capability to overlay sophisticated graphics motivated the creation of the techniques discussed in this paper.

Methodology

Although the claimed merits of the methods in this paper include simplicity and ease of understanding, there are two aspects of this algorithm that may not be immediately apparent without describing in detail. The first involves the method of discretization and geospatial projection used to manage and measure the density. And the second aspect involves the data management necessary to implement this discretization and efficiently calculate the density. Two data management practices will be discussed: the original use of hash data structures and the recent use of matrix-based data structures.

The pointdensity() algorithm

The `pointdensity()` algorithm applies elementary geometry with a geospatial projection in order to build the point density measurements. Figure 2 shows how the neighborhood radius, r, is projected across a square grid. Geographers refer to this as a cylindrical or mercator projection where longitudinal lines appear to remain equidistant. The mercator projection has been used for the sake of illustration, however the algorithm uses a spherical projection and great circle distances to achieve maximal accuracy. As previously mentioned, the algorithm applies a simple binning rule that significantly reduces computations. Density is only collected at the intersection of the grid lines shown in the figure, referred to as grid points. Assume that the center of the circle represents the grid point closest to an event or binned collection of events. The density of every grid point within r will increase by the amount of density associated with the center grid point. The algorithm iterates north and south within r along the latitudes of the grid and calculates the tolerance, t, of longitudinal grid lines that are within r. The spherical Pythagorean theorem is used to find t, providing an accurate binning tolerance irrespective of location on the globe (Veljan, 2018). This binning tolerance ensures that only grid points within r increase in density.

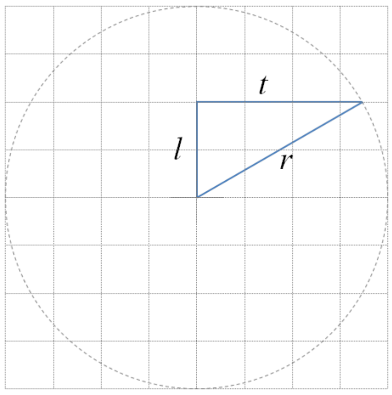


Figure 2: Calculating the longitudinal tolerance.

Figure 3 shows a toy example of the point density method examining only three points. This image shows points A, B, and C, and the radius is four grid steps. Assume that these three points are near the equator and the mercator projection is a reasonable representation of true distance to scale. This figure shows the final density count for every point within the neighborhood radius of the three events.

The original hash-based data structure approach is shown with pseudo-code in algorithm 1. The interested reader can find the implementation of the hash-based algorithm in **pointdensityP** version

```
1: Store $n$ points and let $lat_i$, $lon_i$, and $date_i$ represent the latitude, longitude, and date of
   each point, respectively. Let $g$ represent the grid size measured in degrees latitude, and
   $r$ represent the radius measured in grid steps. For each $lat_i$ and $lon_i$, round each to the
   nearest grid point, and store the rounded points as $tlat_i$ and $tlon_i$. Set $m = 0$.
2: **for** $i = 1$ to $n$ **do**
3:   **if** bin_density_hash$(tlat_i, tlon_i)$ exists **then**
4:     bin_density_hash$(tlat_i, tlon_i) + +$
5:     bin_temporal_hash$(tlat_i, tlon_i) + = date_i$
6:   **else**
7:     bin_density_hash$(tlat_i, tlon_i) = 1$
8:     bin_temporal_hash$(tlat_i, tlon_i) = date_i$
9:     active_grid_hash$(m) = (tlat_i, tlon_i)$
10:    $m + +$
11:  **end if**
12: **end for**
13: **for** $j = 1$ to $m$ **do**
14:   retrieve $tlat_j$ and $tlon_j$ from active_grid_hash$(j)$
15:   **for** $lat_t = tlat_j - rg$ to $tlat_j + rg$ **do**
16:     $t = \arccos(\cos(rg) / \cos(lat_t - tlat_j))/g$
17:     round $t$ to the nearest integer
18:     $t = t * g$
19:     **for** $lon_t = tlon_j - t$ to $tlon_j + t$ **do**
20:       density_hash $(lat_t, lon_t) + +$
21:       temporal_hash$(lat_t, lon_t) + =$ temporal_hash$(tlat_j, tlon_j)$
22:       $lon_t = lon_t + g$
23:     **end for**
24:     $lat_t = lat_t + g$
25:   **end for**
26: **end for**
27: **for** $i = 1$ to $n$ **do**
28:   round $(lat_t = lat_i/g)$ to the nearest integer
29:   $lat_t = lat_t * g$
30:   round $(lon_t = lon_i/g)$ to the nearest integer
31:   $lon_t = lon_t * g$
32:   temporal_hash$(lat_t, lon_t)$ = temporal_hash$(lat_t, lon_t)$/density_hash$(lat_t, lon_t)$
33:   print $lat_i$, $lon_i$, density_hash$(lat_t, lon_t)$, temporal_hash$(lat_t, lon_t)$
34: **end for**
```

Algorithm 1: Hash-based point density algorithm

0.2.1. Discussing the original use of hash data structures for the `pointdensity()` algorithm provides two insights: the hash-based approach provides an accessible explanation of the `pointdensity()` computational approach; and secondly, a comparison of the hash-based method to the matrix-based method illustrates a significant difference in computing time when using hash-based data retrieval compared to matrix-based data retrieval. Hash data structures, or associative arrays, are ubiquitous in programming, but considered somewhat of a late arrival to R from a programming standpoint. However, many have pointed out that R's native environment data structures are inherently a hash data structure and can be leveraged as such. Brown (2013) wrote an R package specifically designed to "fully implement" hash data structures in R. The use of hash data structures for the `pointdensity()` algorithm reduced unneeded computations and stored geographic data only within the neighborhood radius of event points. While the use of hash data structures improved the computational complexity from $O(Gm)$ to $O(n(2r)^2)$, the use of matrices and carefully indexed data proved to be an even faster solution. Both computing methods are explained in this paper.

Use of the hash-based data structure achieved desired results, reducing computational complexity while returning an adequate approximation of density. However, the speed of the computation was not satisfactory for larger datasets. While retaining the binned density concept, the algorithm was reduced to one loop of length $= n$ event points using matrix and array facilities. This latter approach will be referred to as the matrix-based approach. The implementation of the matrix-based approach can be found in **pointdensityP** version 0.3.4.

The general process of the `pointdensity()` algorithm reduces a list of n points to a list of m grid

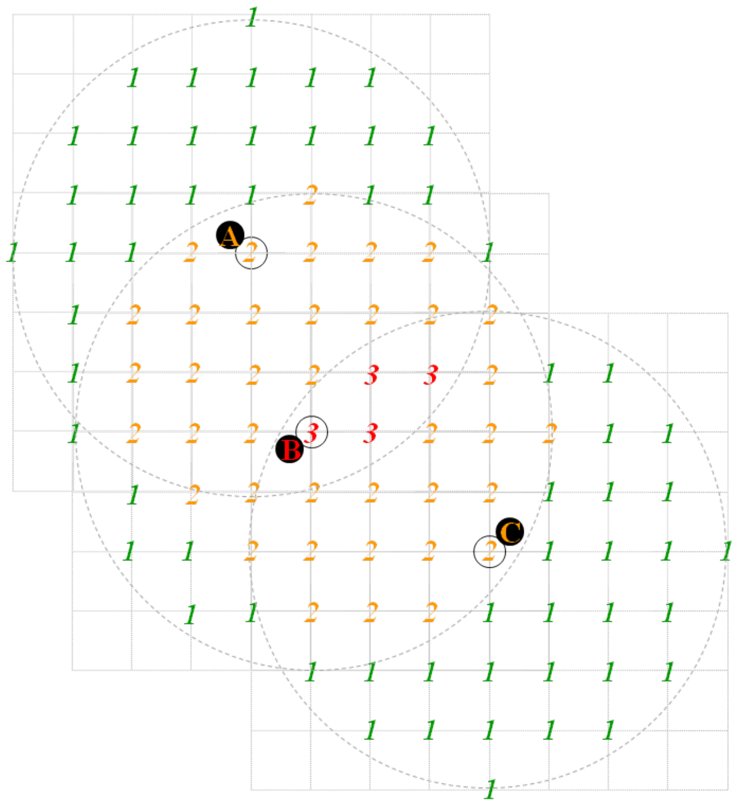


Figure 3: Visualizing the point density algorithm.

points on a mesh. The calculation of density and temporal tendency remains relative to the mesh. Once the overall `pointdensity()` algorithm is complete, it is necessary to assign the density and temporal tendency back to the original list of data. The hash-based algorithm creates intuitive data structures and data retrieval to relate the list of n points to the list of m grid points. Unfortunately, the retrieval of information from the hash tables proved increasingly time consuming as the datasets grew. In order to overcome this problem while continuing to manage the requirement to relate n original points to m grid points, a careful sort and index process, using matrix-based facilities and the **data.table** package, replaced the hash-based storage and retrieval method (Dowle and Srinivasan, 2017). A brief explanation of the improved computing approach follows.

The original list of n points, sorted by their latitude and longitude, reduces to m points once every point is rounded to the nearest point on the mesh. All of the aggregation and sorting leverages functions from the **data.table** package. For each of the m points on the mesh, an initial density of one or greater exists. This initial density is retained through the algorithm. After calculating the final density for each point, this initial density is used to re-expand the list to the original size of n points, in the original order, to return the final density count and temporal tendency.

Each of the m points processes through a density calculation function. The function to calculate the local density surrounding an event point is named `calc_density()` in the **pointdensityP** package. The function starts with enumeration of all latitudes within the radius of the event point, annotated as `lati` and `loni`. These latitudes are stored in a vector:

```
lat.vec <- seq(lati - radius * grid_size, lati + radius * grid_size, grid_size)
```

The variable `radius` represents the radius measured in grid steps, and `grid_size` represents the size of the grid measured in degrees latitude. `lati` is the latitude of the grid point located closest to the event point. `lat.vec` is now an array of each latitude contained in the neighborhood. For each latitude, there is a longitudinal distance that represents the tolerance of the radius. This tolerance is computed using the spherical Pythagorean theorem, as previously discussed. `lat.vec.t` will contain the longitudinal tolerance for each latitude:

```
lat.vec.t <- acos(cos(radius * grid_size) / cos(lat.vec - lati))
lat.vec.t <- lat.vec.t / cos(lat.vec * 2 * pi / 360)
lat.vec.t <- round(lat.vec.t / grid_size, 0) * grid_size
```

The final line of code above rounds the tolerance to the nearest longitude grid line on the mesh.

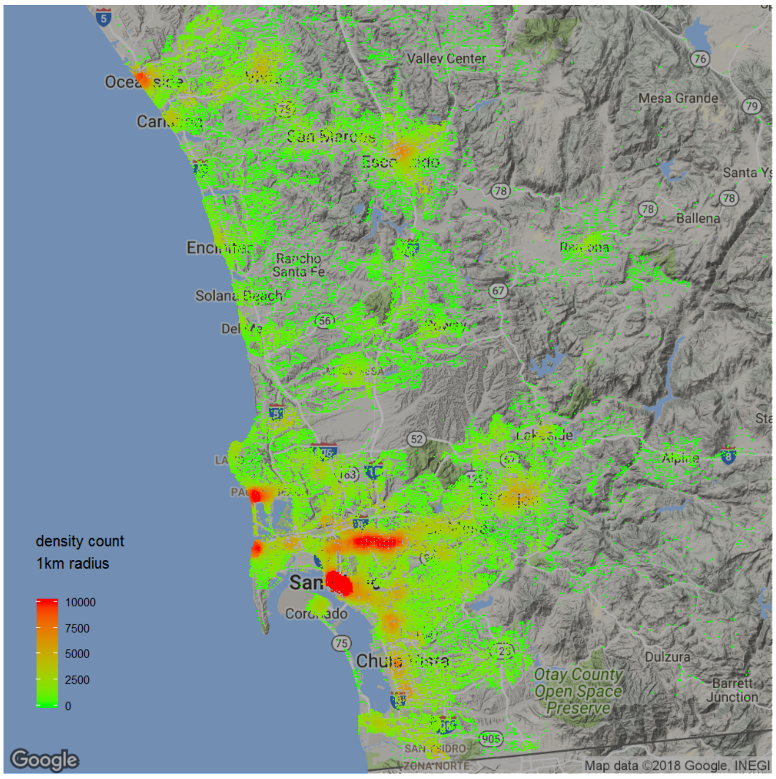


Figure 4: Point density visualization of San Diego crime.

The calculation of `lon.vec` occurs in the same manner as `lat.vec`, which is then further expanded to a matrix, `lon.mat`. This matrix contains the longitudinal value of every point within a square mesh that includes `(2 * radius + 1)^2` points, with the event point at the center. After subtracting `lati` from this matrix, the absolute value of the matrix then contains the distance between the longitude of the event point and the longitude of every point in the mesh. The final subtraction of `lat.vec.t` then returns a matrix that indicates all points outside of the radius as a negative value. After setting all negative elements to zero and all positive values to 1, the Hadamard product of this binary matrix and the original matrix of longitudinal values contained in `lon.mat` returns a matrix that contains a longitudinal value for every point within the radius and a value of 0 for every grid point outside of the radius. Combining these non-zero longitudinal values with the latitude vector returns a list of points within the radius of the event grid point. The final step of the local density calculation involves assigning the temporal value, or sum of temporal values if there is more than one event that rounds to the event grid point, and assigning the original density of the event point to every grid point in the neighborhood. These steps distribute the density and the temporal value of the event point across the neighborhood, completing function `calc_density()` and creating a list of grid points prepared for aggregation without processing a for-loop in R.

As each of the m points processes through `calc_density()`, the list of returned points adds to a **data.table** object which is then aggregated, summing both the density and temporal value for each unique grid point in the **data.table** object. After processing each of the m points and aggregating the returned results, every grid point with a positive density exists in the **data.table** object. The m grid points associated with one of the original n points are retained and re-sorted into their original order, and all other points are discarded. The initial density of every grid point is then added as an additional column to the **data.table**, which is then expanded to length n with the following command:

```
nfinal <- final.1[rep(seq(1, nrow(final.1)), final.1$yy_count)]
```

The **data.table** `final.1` contains the grid point location, density, and sum of temporal values. The field `yy_count` contains the number of original points that were associated with the event's grid point location. After this expansion, a sorted list of n events exists that includes the density and temporal sum of every original point. The temporal sum is then divided by the density. The `pointdensity` algorithm finally returns every original point location, a density value, and the temporal tendency of the point.

Figure 4 displays the calculated density for the San Diego crime data (San Diego Regional Data Library, 2015). Visualizing only the points in the location where the crime occurred provides spatial specificity, and the coloring provides an explainable measure of the density. Figure 4 results from the following commands:

```
SD_density <- pointdensity(df = SD_sub, lat_col = "lat", lon_col = "lon",
  date_col = "date", grid_size = 0.1, radius = 1)
SD_density$count[SD_density$count > 10000] <- 10000
  ## creates discriminating color scale
map_base + geom_point(aes(x = lon, y = lat, colour = count), shape = 16, size = 2,
  data = SD_density) + scale_colour_gradient(low = "green", high = "red")
```

The difference in speed between the hash-based approach and the newer matrix-based approach was significant. The matrix-based approach eliminates the need for information retrieval from large hash tables, the most expensive aspect of the hash-based approach. Table 1 compares **pointdensityP** version 0.2.1 (hash-based) against version 0.3.2 (matrix-based).

data records	10	10^2	10^3	10^4	10^5	10^6
hash-based time (s)	0.06	0.14	1.19	10.60	265.65	857.15
matrix-based time (s)	0.03	0.15	1.19	7.97	33.35	74.88

Table 1: Time comparison between hash-based algorithm and matrix-based algorithm.

Temporal tendency of event data

A byproduct of the original `pointdensity()` algorithm was the concept of computing the temporal tendency for the events in the neighborhood of every event. Given a set of n events with a spatial and temporal dimension, it is possible to calculate an average event date for the entire dataset or any subset of the data. This is an elementary calculation, however when combined with the neighborhood aggregation approach used in the `pointdensity()` algorithm, it is possible to expose previously undetected trends and patterns.

Figure 5 shows a toy example of two time series plots that span an equivalent period. Clearly the plot on the left will have a smaller temporal tendency due to the heavier density in the earlier periods. This is unarguably a simplistic measure for detecting differences in trends, however it has proven to be a reliable technique in practice.

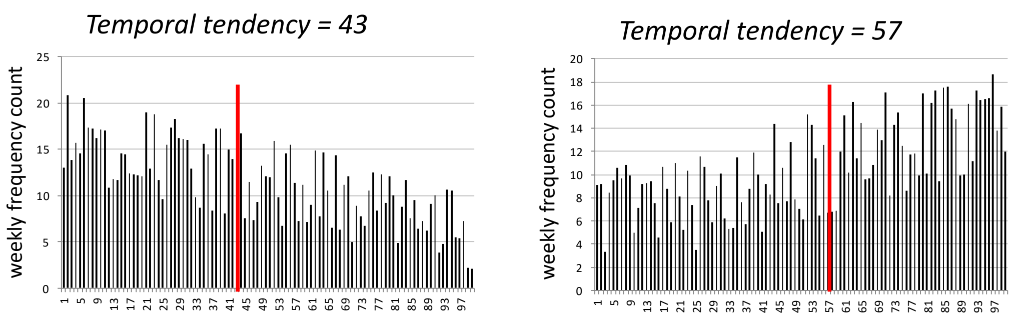


Figure 5: A toy example of the temporal tendency discriminating between an increasing and decreasing trend.

The temporal tendency represents the average age of all events within the neighborhood. This measurement also creates a naive measurement for the relative temporal trend, increasing or decreasing, in the neighborhood. A neighborhood with an increasing trend will have a more recent temporal tendency since a higher density of events occurred more recently. A neighborhood with a decreasing trend will have an older temporal tendency, representing a higher density of events occurring earlier in the time period.

The spatial patterns that emerge from the temporal tendency projection provide the most compelling aspect of the temporal tendency calculation. Some tuning of the color scale is necessary to create visual discrimination. The following lines of code create the underlying temporal tendency plot and histograms shown in Figure 6:

```
SD_temp_tend <- SD_density[SD_density$date_avg > 0]
SD_temp_tend$dateavg[SD_temp_tend$dateavg < 14711] <- 14711
```

```
SD_temp_tend$dateavg[SD_temp_tend$dateavg > 14794] <- 14794
map_base + geom_point(aes(x = lon, y = lat, colour = dateavg), shape = 16, size = 2,
  data = SD_temp_tend) + scale_colour_gradient(low = "green", high = "red")
```

Histograms result from the following:

```
x <- as.Date(SD$date)
hist(x, "weeks", format = "%d %b %y", freq = TRUE)
mid_city <- subset(SD, lat > 32.69 & lon > -117.14 & lat < 32.79 & lon < -117.08)
x <- as.Date(mid_city$date)
hist(x, "weeks", format = "%d %b %y", freq = TRUE)
encinitas <- subset(SD, lat > 33 & lon > -117.32 & lat < 33.09 & lon < -117.27)
x <- as.Date(encinitas$date)
hist(x, "weeks", format = "%d %b %y", freq = TRUE)
```

Figure 6 shows the results of temporal tendency projection for the San Diego crime data. Regions of disparate activity become apparent with this projection, providing an opportunity to detect areas that are "heating up" compared to areas that are "cooling down". The overall San Diego crime dataset shows an aggregate decreasing trend for 2007 - 2012. Box 1, Encinitas, shows a slightly increasing trend. Mid-City is a region with one of the strongest decreasing trends.

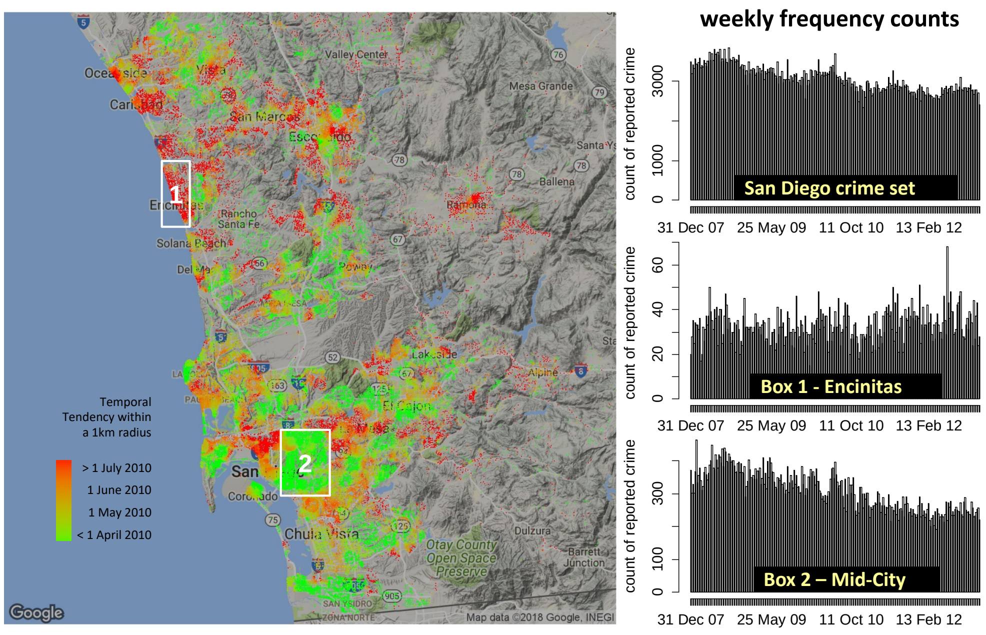


Figure 6: Temporal tendency projection of San Diego crime data.

Simple linear regression is a common method to detect trends. The following code creates a linear model and summary of the model's utility for any of the histograms discussed above:

```
res <- hist(x, "weeks", format = "%d %b %y", freq = TRUE)
xres <- res$breaks[1:(length(res$breaks) - 1)]
summary(lm(res$counts ~ xres))
```

The simple linear regression estimate for the slope coefficient parameter was significant for each of the three regions explored. The map and histograms in Figure 6 highlight the two sub-regions analyzed, Encinitas and Mid-City, as well as the San Diego super-set region that includes all of the data. The trends for each of these regions are apparent from the histograms, and the model utility test results shown in Table 2 for each of these regions shows that there is statistically significant evidence that the slope coefficients are not zero, indicating that a trend exists. This test provides both evidence of the existence of a trend and also quantifies the expected rate of change. The San Diego crime rate in its entirety is dropping by about one crime every two days during this period. Encinitas, lit up in red on the map, shows a relatively more recent temporal tendency when compared to crime across the entire San Diego region, indicating that the rate of crime change in Encinitas will be increasing at a greater rate when compared to the entire San Diego region. The Encinitas histogram shows evidence of a slight increase, and the model utility test for the slope coefficient provides evidence that the slope is greater than zero. On the opposite end of the spectrum, Mid-City averages a crime rate decrease that drops by 1 crime every 10 days. The green coloring on the map clearly highlights an older temporal tendency in this region, which is reinforced by the histogram and regression analysis.

	Estimate	Std. Error	t statistic	Pr(>|t|)
San Diego Total	-0.5077	0.0204	-24.89	<2e-16
Encinitas	0.0023	0.0008	2.88	0.00435
Mid-City	-0.0958	0.0037	-25.81	<2e-16

Table 2: Simple linear regression slope coefficient estimates for weekly crime frequencies shown in Figure 6.

Realize that the temporal tendency calculation is sensitive in areas with a low density. When visualizing the map-based plot, it is possible to compare regions with minimal density with regions that have a much more significant density. For this reason, it is often advisable to use the temporal tendency visualization alongside the point density visualization and possibly restrict the temporal tendency to areas with a minimum density threshold.

Conclusion

Both the temporal tendency projection and point density projection help illustrate regions of disparate activity. These are regions where event behavior is significantly different, oftentimes warranting investigation into underlying causes. The temporal tendency analysis provides a useful perspective on a commonly elusive dimension of spatio-temporal data: change. The most important insights involving spatio-temporal phenomenon commonly relate to change over time.

Future directions for this work include the calculation, exploration, and visualization of other summary statistics that summarize spatio-temporal behavior. For example, estimates for the variance and skew of the temporal behavior readily extends from the aggregation of the squared and cubed temporal values. Analysis of these measurements could provide better fidelity on trends. Lastly, automating the detection of boundaries of regions of disparate activity could provide significant contributions in identifying cause and effect of spatio-temporal behavior.

Acknowledgments

The authors would like to express appreciation for the thoughtful comments and review of this article by the editors of the R Journal and Ian Irmischer. The opinions expressed in this article are the authors' own and do not reflect the view of the United States Army or the United States government.

Bibliography

T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernandez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc. VLDB Endow.*, 8(12):1792–1803, 2015. ISSN 2150-8097. URL `https://doi.org/10.14778/2824032.2824076`. [p348]

A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press, London, 2015. URL `https://doi.org/10.1201/b19708`. [p347]

C. Brown. *Hash: Full Feature Implementation of Hash/Associated Arrays/Dictionaries*, 2013. URL `http://CRAN.R-project.org/package=hash`. R package version 2.2.6. [p350]

M. Dowle and A. Srinivasan. *Data.table: Extension of 'data.frame'*, 2017. URL `https://CRAN.R-project.org/package=data.table`. R package version 1.10.4-3. [p351]

Environmental Systems Research Institute (ESRI). ArcGIS Release 10.6.1, 2018. [p349]

P. Evangelista and D. Beskow. *pointdensityP: Point Density for Geospatial Data*, 2018. URL https://CRAN.R-project.org/package=pointdensityP. R package version 0.3.4. [p347]

D. Kahle and H. Wickham. ggmap: Spatial Visualization with Ggplot2. *The R Journal*, 5(1):144–161, 2013a. [p349]

D. Kahle and H. Wickham. *Ggmap: A Package for Spatial Visualization with Google Maps and OpenStreetMap*, 2013b. URL http://CRAN.R-project.org/package=ggmap. R package version 2.3. [p349]

B. D. Ripley, W. N. Venables, D. M. Bates, K. Hornik, A. Gebhardt, and D. Firth. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2015. URL http://CRAN.R-project.org/package=MASS. R package version 7.3-37. [p347]

San Diego Regional Data Library. San Diego region crime incidents 2007-2013, 2015. URL https://s3.amazonaws.com/s3.sandiegodata.org/repo/clarinova.com/crime-incidents-casnd-7ba4-r3/incidents-5y.csv. Accessed: 2018-03-01. [p348, 352]

D. Veljan. The 2500-Year-Old Pythagorean Theorem. *Mathematics Magazine*, 73 (4):0 259–272, 2018. URL https://doi.org/10.1080/0025570x.2000.11996853. [p349]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, New York, 4th edition, 2002. [p347]

M. Wand. Fast Computation of Multivariate Kernel Estimators. *Journal of Computational and Graphical Statistics*, 3(4):433–445, 1994. [p347]

M. Wand and B. Ripley. *KernSmooth: Functions for Kernel Smoothing for Wand & Jones (1995)*, 2014. URL http://CRAN.R-project.org/package=KernSmooth. R package version 2.3-13. [p347]

M. P. Wand and M. C. Jones. *Kernel Smoothing*. Monographs on statistics and applied probability. Chapman & Hall/CRC, Boca Raton (Fla.), London, New York, 1995. ISBN 0-412-55270-1. [p347]

Paul F. Evangelista
United States Military Academy
West Point, NY
United States
paul.evangelista@westpoint.edu

David Beskow
United States Military Academy
West Point, NY
United States
david.beskow@westpoint.edu

Consistency Cubes: a fast, efficient method for exact Boolean minimization.

by Adrian Dusa

Abstract A lot of effort has been spent over the past few decades in the QCA methodology field, to develop efficient Boolean minimization algorithms to derive an exact, and more importantly complete list of minimal prime implicants that explain the initial, observed positive configurations.

As the complexity grows exponentially with every new condition, the required computer memory goes past the current computer resources and the polynomial time required to solve this problem quickly grows towards infinity.

This paper introduces a new alternative to the existing non-polynomial attempts. It completely solves the memory problem, and preliminary tests show it is exponentially hundreds of time faster than eQMC, the current "best" algorithm for QCA in R, and probes into a territory where it competes and even outperforms engineering algorithms such as Espresso, for exact minimizations.

While speed is not much of an issue now (eQMC is fast enough for simple data), it might prove to be essential when further developing towards all possible temporal orders, or searching for configurations in panel data over time, combined with / or automatic detection of difficult counterfactuals etc.

Context

QCA (Qualitative Comparative Analysis) is a Boolean minimization method that seeks to find the smallest causal configuration that is associated with (sufficient for) the presence of an outcome of interest. It has been introduced in the social sciences literature by Ragin (1987), and applies an algorithm firmly introduced in the engineering field by McCluskey (1956), building on the previous work of Quine (1952, 1955).

The input for such a procedure is a truth table, which presents all possible combinations of presence (coded with 1) and absence (coded with 0) for all causal conditions, plus an additional column to specify in which cases the output is present and respectively absent.

With four causal conditions, a complete matrix with all their possible combinations could be generated using these commands:

```
> library(QCA)
> createMatrix(rep(2, 4))
      [,1] [,2] [,3] [,4]
 [1,]    0    0    0    0
 [2,]    0    0    0    1
 [3,]    0    0    1    0
 [4,]    0    0    1    1
 [5,]    0    1    0    0
 [6,]    0    1    0    1
 [7,]    0    1    1    0
 [8,]    0    1    1    1
 [9,]    1    0    0    0
[10,]    1    0    0    1
[11,]    1    0    1    0
[12,]    1    0    1    1
[13,]    1    1    0    0
[14,]    1    1    0    1
[15,]    1    1    1    0
[16,]    1    1    1    1
```

For all rows / combinations of causal conditions that enter the analysis (that is, where an additional column with the output coded as 1), the task is to:

- compare all possible pairs to determine if they can be minimized (only those which differ by exactly one literal)
- compile a list of implicants as the result of minimizing pairs of rows
- iteratively compare all pairs of the surviving implicants and minimize further
- until nothing more can be minimized, thus obtaining the so-called "prime implicants"

In this example, the last two rows can be minimized (they differ only in the fourth column), with the following result:

```
1  1  1  0
1  1  1  1
----------
1  1  1  x
```

A computer intensive, iterative process can determine a final solution in a polynomial time for a relatively small number of conditions. But each new condition included in the analysis exponentially increases the complexity of the problem to at least the powers of 3, and even more for multi-value conditions.

Although the initial truth table is specified in base 2, the calculations are in fact done in base 3 because an additional bit of information is needed to specify which column has been minimized (coded "x" in the example above).

All solutions are found in the so-called "implicant matrix", which for binary crisp conditions has 3^k rows and k columns, where k is the number of conditions. Solving time increases exponentially with each new condition added in the analysis, and quickly grows towards infinity when k reaches a certain value.

Memory consumption is obviously a big issue, as generating the entire implicant matrix quickly depletes all available memory and make the computer stop because of lack of resources. Such an approach is bound to be inefficient even for moderately sized situations (11 or 12 causal conditions), after which finding a solution in polynomial time becomes impossible.

Previous attempts

A first attempt to find a non-polynomial complete solution was presented in "A mathematical approach to the Boolean minimization problem" (Dușa, 2010). The basic novelty of that approach was the use of a vector of decimal row numbers instead of the entire implicant matrix. This shortcut partially solved the memory use (k times lower than a complete matrix), while the solving time was significantly reduced by using a basic mathematical operation `2x - y` to minimize two rows using their decimal representations:

```
79     2  2  2  1
80     2  2  2  2
-----------------
78     2  2  2  0
```

In this example, the original binary number 0 for the absence is changed to 1 in the base 3 implicant matrix, the original binary number 1 for the presence is changed to the number 2, while the number 0 is reserved to signal a minimized condition. Applying the `2x - y` operation, the minimized implicant is found on the line `158 - 80 = 78`.

These kinds of improvements helped a lot, at that time obtaining a solution in a significantly lower time and probing into a previously unexplored space of 14 to 15 causal conditions with exact results, the complexity growing three times with each new condition.

However, it was still a problem finding and comparing all possible pairs of minimizable row numbers (not all pairs of numbers can be minimized, because as not all pairs of rows in the implicant matrix differ by exactly one column). A mathematical shortcut was used to find minimizable pairs using the powers of 3 after transforming the row into base 3, at the cost of more computing time.

Even if using vectors of row numbers instead of matrices, after the ceiling of 14 to 15 causal conditions the memory was still an issue and solving time also progressed towards infinity. In addition, this method was restricted to binary data only, whereas QCA also needed to use multi-valued data.

The next attempt to solve this problem was represented by the so-called "enhanced Quine-McCluskey" procedure (Dușa and Thiem, 2015), which improved on the classical minimization algorithm on both speed and memory, and this algorithm is the current "best" workhorse for QCA family of R packages.

The foundation of the second attempt is represented by an important observation which is also valid for the algorithm to be presented in this paper: the final list of prime implicants are supersets of those cases with a positive output, and at the same time they cannot be supersets of the cases with a negative output. To better understand this concept, the entire truth table can be generated and presented as below:

```
> set.seed(1234)
> tt <- data.frame(createMatrix(rep(2, 4)), stringsAsFactors = FALSE,
           OUT = sample(c(1, 0, "?"), 16, replace = TRUE, prob = c(.19, .18, .63)))
> colnames(tt)[1:4] = LETTERS[1:4]
> tt
   A B C D  OUT
1  0 0 0 0   ?
2  0 0 0 1   ?
3  0 0 1 0   ?
4  0 0 1 1   ?
5  0 1 0 0   0
6  0 1 0 1   1
7  0 1 1 0   ?
8  0 1 1 1   ?
9  1 0 0 0   1
10 1 0 0 1   ?
11 1 0 1 0   1
12 1 0 1 1   ?
13 1 1 0 0   ?
14 1 1 0 1   0
15 1 1 1 0   ?
16 1 1 1 1   0
```

For some rows, the output is positive (coded with "1"), for other rows it is absent (coded with "0") while for the rest of the truth table the output is unknown (and coded with "?"). That happens because the number of all possible combinations (especially when the number of causal conditions k increases), is far greater than what we can observe in reality. In other words, as Ragin and Sonnett (2005) put it, the diversity is limited and it can be reduced to this part of the truth table (sorted decreasingly by the output):

```
> tt <- tt[tt$OUT != "?", ]
> (tt <- tt[order(tt$OUT, decreasing = TRUE), ])
    A B C D  OUT
6   0 1 0 1   1
9   1 0 0 0   1
11  1 0 1 0   1
5   0 1 0 0   0
14  1 1 0 1   0
16  1 1 1 1   0
```

Minimizing the three rows with a positive output is a trivial task and doesn't require a special computer program. The difficulty starts when employing the so-called "counterfactual analysis" to determine the most parsimonious solution, by assuming the output is also positive for the configurations with no empirical information (called "remainders" in QCA, also known as "don't cares").

But instead of calculating all possible pairs of minimizable configurations where the output is not negative (either positive, or unknown, as in the previous attempt), the eQMC algorithm focuses only on the known sub-matrix and applies the following procedure:

- find all possible superset implicants of the rows with a positive output
- find all possible superset implicants of the rows with a negative output
- calculate the differences, preserving those implicants where the output is positive
- remove the redundant implicants: those which already have supersets in the surviving vector of differences

A superset is a simpler, less complex logical expression that covers a specific conjunction. For example, the causal condition A is a superset of the intersection A*B. In the implicant matrix, this would be:

```
        A   B   C   D
54      2   0   0   0      A
72      2   2   0   0      A*B
```

Determining specific supersets of specific row numbers (in this case, that row number 54 is a superset of row number 72) uses the same mathematical tricks as in the previous attempt, employing a vector with the powers of 3 for binary data: 3^3, 3^2, 3^1, 3^0.

This new procedure brought a great deal of speed compared with the previous one, consuming a lot less memory (for 15 causal conditions, about 15 MB compared to 1.4 GB). It also made it possible to explore up to 18 conditions at once, therefore additional three conditions meaning $3^3 = 27$ times more complex situations than before.

Past that threshold, the eQMC algorithm reaches its maximum possibilities. The culprit is less represented by memory consumption, although at an extremely high number of causal conditions the generated vectors of row numbers for the supersets can potentially grow larger than available memory. Possibly problematic, after 19 causal conditions the row numbers cannot even be represented in 32 bit computers, which have an upper representation limit of 2^{31}, that is still higher than 3^{19} but not enough compared to 3^{20} for the implicant matrix with 20 causal conditions.

The more serious issue was the calculation time, especially at the final step to remove the redundant implicants from the surviving differences: having that vector sorted in increasing order, the procedure involved an inverse process of finding all subsets for each surviving implicant, starting from the lowest, and iteratively remove those which are found in the rest of the vector.

That process involved a backward transformation from the base 10 row numbers to their base 3 equivalent then applying specific mathematical operations to find all their subsets. For millions or tens of millions of implicants, even this "quick" process can take a lot more time than expected.

Consistency Cubes: the newest approach

Compared to the first attempt, eQMC excelled because it focused on the observed data only. Indirectly, it probes into the implicants' space during the calculation of supersets and subsets, which requires some memory and costs a lot of time. But the focus on the observed data (ignoring the remainders but obtaining exactly the same solutions), has to be the right approach and still stays at the foundation of the new consistency cubes (CCubes) method, which starts with a simple observation:

Definition 3. *A prime implicant is the simplest possible, non-redundant, fully consistent superset of any positive output configuration.*

Fully consistent also means that it can never be a superset of any observed negative configuration. The simplest possible superset expression is a single condition (either its presence, or its absence). Figure 1 presents the graphical representation of the generated truth table `tt` above, and shows that no single condition qualify as a prime implicant in its presence (lit cubes above and below the middle separator). The only condition that is fully consistent in its *absence* is B, but since it does not cover all observed positive configurations, the search must be continued at the next level of complexity.

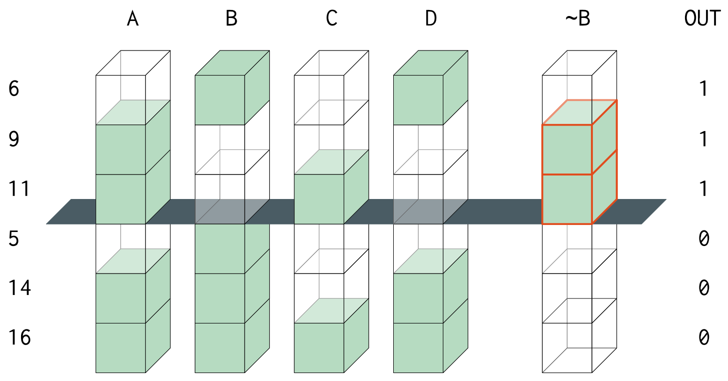


Figure 1: Individual presence inconsistencies, absence of B consistent

The next simplest superset expressions can be found in all combinations of 2 conditions out of 4, and for each such combination a normal computer program would search within all pairs of their respective levels. If conjunctions of two conditions will not prove to be sufficient, the next simplest are superset expressions formed with 3 conditions out of 4, and so on.

For any general case involving multi-value causal conditions, the complete search space S (containing the total number of possibilities to check using a polynomial approach) is equal to the sum of all combinations of c selected conditions out of k, times the product of their respective number of levels l_s for each such selected combination:

$$S_{MV} = \sum_{c=1}^{k} \binom{k}{c} \prod_{s=1}^{c} l_s \tag{1}$$

The CS (crisp sets) space is just a special case with all binary causal conditions having two levels each, 0 and 1, where this equation gives the same result as $3^k - 1$ that describes the implicant matrix having exactly this number of all possible expressions which Ragin (2000, pp.132-136) calls "groupings". These are all rows from the entire 3^k implicant matrix, minus the first one which indicates a situation when all causal conditions are minimized:

$$S_{CS} = \sum_{c=1}^{k} \binom{k}{c} 2^c = 3^k - 1 \tag{2}$$

Ragin uses a similar technique and lists all possible combinations of causal conditions and their levels in Table 5.4 (p.134). But he applied his calculation backwards, starting from the most complex expressions of four conditions out of four, towards all combinations of levels from the most simple situations with one separate condition at a time.

As it will be shown in the next section, starting from the simplest expression towards the most complex, has a direct consequence over the performance of this algorithm. Fast forward, the properties of Equation 1, combined with the requirements from Definition 3 that describe a minimal prime implicant, are key ingredients behind the speed of the new CCubes algorithm.

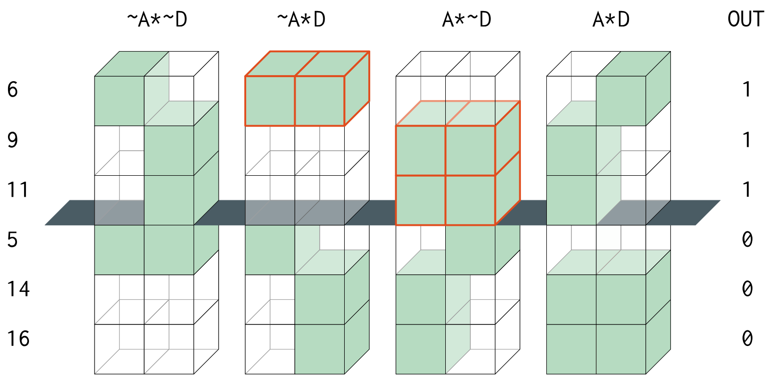


Figure 2: Conjunctive prime implicants

Increasing the complexity to level `c = 2`, Figure 2 above shows all possible combinations of levels for the pair of conditions `A` and `D`, out of which two qualify as prime implicants: `~A*D` and `A*~D` are both associated with a positive output above the middle grey separator, and at the same time not associated with a negative output below the separator.

Search space

An exhaustive exploration of the entire search space, through all possible combinations of columns, and all their subsequent combinations of levels, is a polynomial approach which is bound to consume time. Figure 2 displays all combinations of levels from columns `A` and `D`, but there are 6 such pairs of columns, each having 4 pairs of levels. For each pair of columns, a basic program would have to spend time comparing 2 columns, times 4 pairs of levels, times 6 rows which gives 48 computer cycles.

The search space described in Equation 1 can be partitioned into successive layers of complexity, from a single condition for `c = 1` to the most complex conjunctions for `c = k`. Within each layer, it is not necessary to search within all possible combination of levels, because the relevant ones can already be found in the observed data. This is a fortunate situation that alleviates the memory problem because, unlike the previous two attempts, no other memory needs to be used in the process.

The particular arrangement of the cube components in Figure 2 offers a first hint: out of the four pairs, only two are meaningful, and they need not be displayed into separate combinations of levels, as they can already be detected in the original structure of the observed data.

```
> tt[, c("A", "D", "OUT")]
   A D OUT
6  0 1   1
9  1 0   1
11 1 0   1
5  0 0   0
14 1 1   0
16 1 1   0
```

The conjunction ∼A*D (which is the binary combination '01', on the first line number 6), and the conjunction A*∼D (binary combination '10' on the second and third lines, numbers 9 and 11) are both prime implicants because they are found in the observed positive cases and not in the negative ones.

In this example, the search space can be shortened to only two columns times 6 rows, already four times quicker than a polynomial approach. But as it turns out, the search space can be shortened even further using the mathematical tricks from the previous approaches, transforming from any (multiple) base (in this example with all conditions in base 2) to their decimal representation:

```
> data.frame(AD = as.matrix(tt[, c("A", "D")]) %*% c(2^1, 2^0), OUT  = tt$OUT)
   AD OUT
6   1   1
9   2   1
11  2   1
5   0   0
14  3   0
16  3   0
```

Instead of the exhaustive 48 computer cycles, the same conclusion can be derived by examining only these 6 rows, where decimal numbers 1 and 2 (corresponding to the binary combinations 01 and 10) are not found among the observed negative cases. In this simple example, the gain might be minimal, but for more complex datasets with conjunctions of 5 or 6 causal conditions, these sort of transformations dramatically reduce the search space, with a significant boost of the overall speed.

This technique of partitioning the search space is fundamentally different from the previous two attempts, which approached the implicant matrix in the increasing order of its decimal row numbers. The implicant matrix is by definition unpartitioned, and the example below display all levels for each individual condition, equivalent to the simplest level of complexity in Figure 1:

```
> IM <- createMatrix(rep(3, 4)) # base 3 for 4 causal conditions
> rownames(IM) <- seq(nrow(IM))
> colnames(IM) <- LETTERS[1:4]
> IM[apply(IM, 1, function(x) sum(x > 0) == 1), ]
   A B C D
2  0 0 0 1
3  0 0 0 2
4  0 0 1 0
7  0 0 2 0
10 0 1 0 0
19 0 2 0 0
28 1 0 0 0
55 2 0 0 0
```

The presence of a single condition A is found all the way on the decimal row number 55, therefore the entire implicant matrix has to be scanned to find even the most simple conjunctions, and that takes a lot of time. By contrast, and in line with Definition 3, the same search space can be partitioned in successive layers of complexity before starting the search process, as shown in Equation 1.

This is one of the key novelties of the CCubes algorithm, that boosts its speed to unprecedented levels while maintaining the same overall requirement to arrive at exact and complete solutions. It does that by exploring the same space (the implicant matrix), but using a different method. As an added bonus, the new algorithm uses the best features from all attempts, by partitioning the search space and at the same time using the decimal representation from eQMC, when comparing which numbers are found in the positive part and (not) in the negative part of the truth table.

Search depth

Another factor that can increase the speed of the CCubes algorithm is the search depth, which seeks to terminate the partitioned search space as soon as possible. To satisfy Definition 3, the algorithm has to search through all possible combinations of c selected conditions out of the total k, starting from a single condition with c = 1 up to the most complex conjunctions with c = k.

But parsimonious prime implicants are never found among the most complex expressions, instead they are always found among the simplest possible ones. A "simple" expression, for say 4 or 5 initial causal conditions is a single condition or a conjunction of two conditions. For a more complex dataset with 20 causal conditions, the word "simple" has a different meaning, with possible conjunctions of up to 4 or 5 conditions each.

More complex expressions are either not contributing to the minimization, or they are subsets of other, less complex expressions which have already been found in the previous iterations of `c` selected conditions. As the redundant prime implicants are to be removed anyways (a key part of the current eQMC procedure), they will never qualify as "minimal" prime implicant expressions. This is why Definition 3 explicitly mentions "non-redundant" expressions that do qualify as prime implicants.

To maximise speed, the search depth has to be stopped as soon as there is proof that no more prime implicants will be found at more complex expressions. A brute force traditional algorithm searches the entire search space from equation 1. In the binary crisp sets analysis, it involves an exhaustive search of all 3^k possible combinations in the implicants matrix. But that is a huge space, especially at a large number of causal conditions, and most of the verifications are void. It is one of the reasons why the traditional algorithms are inherently slow.

A brute force, exhaustive search will always find all non-redundant prime implicants, at the cost of a larger search time. The task is to find those prime implicants without spending additional time for unnecessary, void verifications. As it turns out, the design of the equation 1 is also the answer for this task.

Using the truth table `tt` presented in this paper, any minimization algorithm will find exactly four prime implicants: `~B`, `~A*D`, `A*~D` and `C*~D`, extracted from an implicants matrix with $3^4 - 1 = 80$ rows.

The simplest prime implicant is `~B`, which has the biggest number of subsets in the implicants matrix. More complex prime implicants have fewer subsets, and there is a method to determine the exact number of subsets for any given prime implicant.

The prime implicant `~B` can be written as `0100` (where `1` means absence of `B`, and `0` means all other conditions are minimized), and using a similar product formula as that from equation 1 it can be determined it has $27 - 1 = 26$ subsets. There are three minimized conditions in this prime implicant (`A`, `C` and `D`), each having 2 + 1 levels in the implicants matrix, and the number of subsets can be obtained with the product of the number of levels for the minimized conditions:

$$\prod_{c=1}^{3}(l_c + 1) = 3^3 = 27 \tag{3}$$

27 rows from the implicant matrix contain `~B`, out of which 26 are subsets for which further verifications would consume unnecessary time. The second prime implicant `~A*D` has 8 subsets, out of which 3 are already covered by `~B`, making 31 unique subsets. The third has 8 subsets as well, out of which 3 are already covered by the first two, making 36 unique subsets, and the fourth has 8 subsets out of which 5 are already covered by the first three, making a grand total of 39 unique subsets for all found prime implicants.

Out of the 80 rows of the implicants matrix to verify, it is certain that 39 will turn void results, which means the search could have been terminated after about half of the search space. This procedure is valid for any number of input causal conditions and for any number of non-redundant prime implicants, leading to the general formula:

$$S_{MV} = \sum_{c=1}^{k} \binom{k}{c} \prod_{s=1}^{c} l_s = P + U + N \tag{4}$$

Here P is the number of prime implicants, U is the total number of unique subsets (of all found P prime implicants), and N is the number of non (prime) implicants. The new algorithm verifies all P rows, most of the N rows, and a small percentage of the U unique subsets. It stops when there is proof that all remaining rows are either U or N: when the prime implicants chart can be solved at a certain complexity level c, and no more prime implicants are identified at complexity level $c + 1$.

This is a crude, but very effective rule to make sure the search is exhaustive, although only a fraction of the entire implicants matrix is actually verified. U represents the largest part of the implicants matrix, and is not necessary to be verified. Avoiding the search through this void part of the implicants matrix dramatically increases the speed of the CCubes algorithm.

To find all possible P prime implicants in a shorter time is already a major achievement, but the next section presents a method to stop the search even sooner, for the default option of an absolute minimal solution corresponding to the setting `min.pin = TRUE` in function `minimize()`.

The prime implicants chart gives a better understanding of how many prime implicants are needed to cover all observed positive cases. By iteratively solving this chart after each partition of complexity, the search space can be terminated as soon as the number of prime implicants needed to solve the chart reaches a minimum.

Solving the prime implicants chart

Once the prime implicants are found, the next and final step of the Quine-McCluskey procedure is to find the minimal disjunctions of expressions (sums of products) that cover all the initial positive configurations in the observed data. The hypothetical example above generates four prime implicants:

```
> PIs <- translate("~B + ~A*D + A*~D + C*~D")
> PIs

       A  B  C  D
~B        0
~A*D   0        1
A*~D   1        0
C*~D         1  0
```

The PI chart is obtained by distributing these four prime implicants on each of the three observed, positive configurations. In this chart representation, the symbol x indicates the prime implicant is a superset of a specific initial configuration (it covers a specific column) and the symbol - indicates the prime implicant is not a superset of (does not cover) that specific configuration:

```
> pic <- makeChart(rownames(PIs), c("~A*B*~C*D", "A*~B*~C*~D", "A*~B*C*~D"))
> pic

      ~A*B*~C*D  A*~B*~C*~D A*~B*C*~D
~B        -          x          x
~A*D      x          -          -
A*~D      -          x          x
C*~D      -          -          x
```

None of the generated prime implicants is able to explain all three initial configurations, because none is covering all three columns. The final solution will be a disjunction of at least 2 prime implicant expressions, where ∼A*D is an essential part of the solution because it is the only one which covers the first column.

The expression C*∼D, although minimal and non-redundant, is found to be irrelevant because it covers a column which is already covered by two other prime implicants (the so called principle of "row dominance"): both ∼B and A*∼D dominate C*∼D because they cover more columns.

The final solution is a disjunction of exactly 2 prime implicants, and there are two alternative such solutions:

```
> minimize(tt, "OUT", include = "?", use.tilde = TRUE)

M1: ~A*D + (~B) <=> OUT
M2: ~A*D + (A*~D) <=> OUT
```

The package **QCA** (Dușa, 2019; Thiem and Dușa, 2012) in R (R Core Team, 2017) used to have a dependency on package **lpSolve**, to determine the minimum number of prime implicants through linear programming. There is, however, another direct method of determining this minimum number, involving a two phase procedure:

1. iteratively simplify the chart, by:
   - eliminating dominated rows
   - eliminating dominating columns
2. iteratively find all sets of (the rest of the) rows that maximally cover the (rest of the) columns

On each iteration in the first phase, the chart is progressively simplified, and on each iteration in the second phase, a new prime implicant is introduced to each previous maximal set, and the procedure stops with the first set to win and cover all columns. There is a dedicated function to find this minimum number m, equal to the output of the previous linear programming:

```
> findmin(pic)
[1] 2
```

For this particular purpose, the new procedure is faster than linear programming but the gain is negligible. The real purpose is to solve the prime implicants chart from within the CCubes algorithm after each partition of complexity c out of k from Equation 1, to determine:

- if the generated prime implicants can explain all initial positive configurations (all columns in the prime implicants chart are covered by at least one row)
- more importantly, if the number of prime implicants necessary to explain all positive configurations reaches a minimum (when at complexity `c + 1` the number `m` remains the same).

These factors terminate the search space at the depth where the new prime implicants do not contribute to minimally solving the PI chart, thus demonstrating that not all prime implicants are relevant. Only those which are found at the lowest levels of complexity from the search space matter.

This is rather easy to explain: the more complex a prime implicants is, the more specific it becomes and covers fewer columns from the PI chart (initial combinations). A more and more specific conjunction is a smaller and smaller intersection that covers less and less from the Universe, as shown in the following Venn diagram using the package **venn** (Dușa, 2018):

```
> library(venn)
> venn("~A*D + ~B", snames = "A, B, C, D", zcolor = "red, #008833")
```

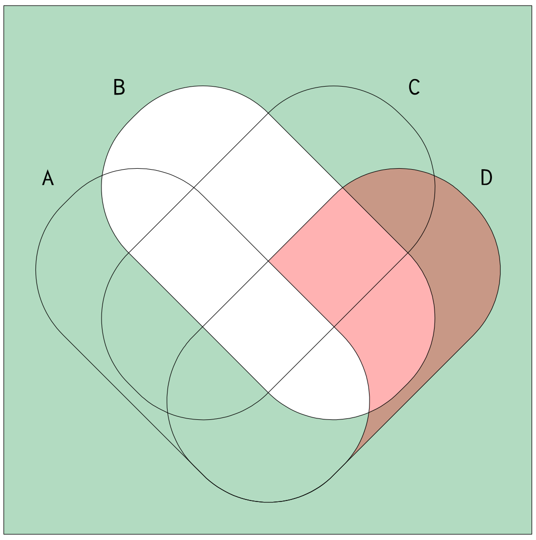


Figure 3: Venn diagram with coverage of ∼`A*D` compared to ∼`B`

Figure 3 above shows the green area corresponding to ∼`B` of complexity level 1, is much larger than the red area corresponding to ∼`A*D` of complexity level 2. A more specific intersection, for example ∼`A*C*D` at complexity level 3, covers an even smaller area (it is actually a subset of ∼`A*D`).

Less complex prime implicants have a much higher chance of covering more columns from the prime implicants chart, which explains why the most complex PIs are almost never part of the minimal solution. For 20 causal conditions, tests have revealed that prime implicants are found up to the complexity level 7, while the minimal solutions don't use prime implicants from a complexity level more than 4.

This is an important observation: although non redundant PIs can be found at higher complexity levels, the search can be stopped at lower complexities, as soon as there is proof the prime implicants chart is solved using a minimal number of prime implicants. Terminating the search space at lower complexity levels dramatically shortens the time necessary to find all possible minimal solutions.

Performance benchmark

All three algorithms (QMC, eQMC and CCubes) are now available in function `minimize()` via the new argument `method`. The motivation is partly due to the existence of multiple competing algorithms and users should choose which one to use, but more importantly to allow for testing and benchmarking, controlling for input and return exactly the same output. The main difference maker is the speed.

In the world of Boolean minimization, the acknowledged champion software is Espresso (Brayton et al., 1984), built more than three decades ago at the University of California, Berkeley. It is a heuristic algorithm that finds a minimal solution with record speed, but unsuitable for social science research. While in circuit design the goal is to find one minimal solution, QCA needs to find all possible causal paths that are sufficient for an outcome, thus involving all possible prime implicants that form a PI chart.

There is an R package on CRAN that implements Espresso version 2.3, called **LogicOpt** (Stiehl, 2016), a perfect opportunity to test the performance of CCubes in terms of speed and solutions found. First of all, a data generation function is needed to make sure all algorithms receive the same input:

```
> gendat <- function(ncols, nrows, seed = 12345) {
    set.seed(seed)
    dat <- unique(matrix(sample(0:1, ncols*nrows, replace = TRUE), ncol = ncols))
    colnames(dat) <- LETTERS[seq(ncols)]
    return(as.data.frame(dat[order(dat[, ncols], decreasing = TRUE), ]))
  }
```

For simplicity, all datasets are restricted to binary crisp having 10 unique configurations (a usual number in QCA research, given the known limited diversity problem), and the number of columns are given by argument ncols out of which the last column will play the role of the output. For a small number of say 10 conditions all three algorithms can be tested, for a large number of 16 conditions QMC is unable to perform, and above 20 conditions CCubes is the only algorithm left.

```
> dat <- gendat(ncols = 11, nrows = 10)
> minimize(dat, outcome = "K", include = "?", method = "QMC")
M1: C + (g) <=> K
M2: C + (b*d) <=> K
M3: C + (d*e) <=> K
M4: C + (d*H) <=> K
M5: C + (d*J) <=> K
M6: C + (e*J) <=> K

> all.equal(
      capture.output(minimize(dat, outcome = "K", include = "?", method = "QMC")),
      capture.output(minimize(dat, outcome = "K", include = "?", method = "eQMC"))
  )
[1] TRUE

> all.equal(
      capture.output(minimize(dat, outcome = "K", include = "?", method = "eQMC")),
      capture.output(minimize(dat, outcome = "K", include = "?", method = "CCubes"))
  )
[1] TRUE
```

The examples above show the solutions are identical for all three algorithms. But function minimize() does many other things besides the logical minimization: calculating parameters of fit, deriving simplifying assumptions, easy and difficult counterfactuals etc. To compare the speed of the algorithms themselves, another function is needed, that arranges the input for the specificities of each.

```
> test <- function(dat, method = "", min.pin = FALSE) {
    dat <- as.matrix(dat)
    nconds <- ncol(dat) - 1
    noflevels <- rep(2, nconds)
    output <- dat[, nconds + 1]
    switch(method,
    "QMC" = {
        mbase <- rev(c(1, cumprod(rev(noflevels))))[-1]
        neg <- drop(dat[output == 0, seq(nconds)] %*% mbase) + 1
        .Call("QMC", createMatrix(noflevels)[-neg, ] + 1, noflevels, PACKAGE = "QCA")
    },
    "eQMC" = {
        mbase <- rev(c(1, cumprod(rev(noflevels + 1))))[-1]
        pos <- dat[output == 1, seq(nconds)]
        neg <- dat[output == 0, seq(nconds)]
        rows <- sort.int(setdiff(findSupersets(input = pos + 1, noflevels + 1),
                                 findSupersets(input = neg + 1, noflevels + 1)))
        .Call("removeRedundants", rows, noflevels, mbase, PACKAGE = "QCA")
    },
    "CCubes" = {
        .Call("ccubes", list(dat, min.pin), PACKAGE = "QCA")
    })
  }
```

The output of interest for all three algorithms is the list of prime implicants, either as a matrix for QMC and CCubes, or as a numerical vector of corresponding row numbers for eQMC. The output itself is therefore not identical, but the final solutions are. What matters here is the timings, on a MacBook Pro with an Intel Core i7 processor at 2.7 GHz with 16GB of RAM, at 1600 MHz DDR3.

```
> system.time(test(dat, method = "QMC"))
   user  system elapsed
  4.438   0.010   4.448
> system.time(test(dat, method = "eQMC"))
   user  system elapsed
  0.013   0.000   0.031
> system.time(test(dat, method = "CCubes"))
   user  system elapsed
  0.000   0.000   0.001
```

For as little as 10 causal conditions, the classical Quine-McCluskey algorithm is already the slowest, while eQMC performs very decent. The real test begins 16 conditions and above:

```
> dat <- gendat(ncols = 17, nrows = 10)
> system.time(test(dat, method = "eQMC"))
   user  system elapsed
  6.735   0.996   7.676
> system.time(test(dat, method = "CCubes"))
   user  system elapsed
  0.010   0.000   0.011
```

Past 18 causal conditions, the eQMC algorithm reaches its upper limit, while the CCubes algorithm performs effortlessly:

```
> dat <- gendat(ncols = 26, nrows = 10)
> system.time(test(dat, method = "CCubes"))
   user  system elapsed
  0.482   0.001   0.489
```

Obviously, some truth tables are more complex, the speed depending not only on the number of causal conditions but also on the number of input configurations and their complexity. In this example, the truth table is a very simple one with 5 positive and 5 negative configurations, but more unique configurations added to the truth table have an obvious impact, even with fewer causal conditions:

```
> dat <- gendat(ncols = 21, nrows = 20)
> system.time(cc <- test(dat, method = "CCubes"))
   user  system elapsed
  0.930   0.003   0.946
```

Comparing with Espresso is the more serious challenge. Despite its heuristic nature, it also has a particular implementation that can find all relevant prime implicants, known in the literature under the name of "Espresso-exact" and adapted in the package LogicOpt for comparison with package QCA. Due to historical reasons, Espresso-exact generates not one set of prime implicants, but three: EPs - essential prime implicants, PRPs - partially redundant prime implicants and TRPs - totally redundant prime implicants.

The relevant function in package LogicOpt is called `logicopt()`, with a control argument named `mode` that can take three values: `"primes"` (which leverages the PI generation C-code that is a part of "Espresso-exact"), `"multi-min"` (which finds the minimum set of non-redundant solutions that cover all the EPs and the minimal set of PRPs) and `"multi-full"` (that finds additional coverings of prime implicants beyond what is found in multi-min, and an exhaustive covering of all PRPs). However, since the number of coverings (solutions) is potentially very large, the package LogicOpt employs an arbitrary upper limit of 50 coverings, therefore the output of these two arguments is not directly comparable with the exhaustive number of solutions found by package QCA.

The only way to compare with Espresso-exact is to employ the argument `mode = "primes"` and measure the time taken to identify the prime implicants.

Out of the three sets of prime implicants from Espresso-exact, only the EPs and PRPs play an active role in finding the minimal solutions when solving the PI chart, however it's implementation spends additional time to find the set of TRPs. On the other hand, CCubes does not bother with the TRPs and finds only the relevant set of PRPs (the essential PIs are identified when solving the PI chart).

```
> library(LogicOpt)
> system.time(lo <- logicopt(dat, 20, 1, find_dc = TRUE, mode = "primes"))
   user  system elapsed
  3.030   0.012   3.034
> nrow(cc)
[1] 1931
> lo[[2]][2]
[1] 1931
```

There are exactly 1931 prime implicants identified by both Espresso and CCubes, which is a cross-validation proof between these two algorithms. In terms of speed, CCubes seems to be faster and it can be faster still, by choosing to stop the search when the PI chart can be solved with a minimal number of PIs, using the argument `min.pin`:

```
> system.time(cc <- test(dat, method = "CCubes", min.pin = TRUE))
user  system elapsed
  0.045   0.000   0.045
> nrow(cc)
[1] 687
```

Out of the 1931 prime implicants, only 687 really contribute to minimally solving the PI chart, with the search being terminated a lot sooner with the same final solutions. A more intensive test of speed was carried by generating 1000 random datasets changing the seed in function `gendat()`:

```
> set.seed(12345)
> seeds <- sample(1000000000, 1000)
> result <- matrix(NA, nrow = length(seeds), ncol = 3)
> for (i in seq(length(seeds))) {
    dat <- gendat(21, 20, seeds[i])
    result[i, ] <- c(system.time(test(dat, method = "CCubes"))[[3]],
        system.time(logicopt(dat, 20, 1, find_dc = TRUE, mode = "primes"))[[3]],
        system.time(test(dat, method = "CCubes", min.pin = TRUE))[[3]])
  }
```

On average, CCubes spends 0.941 seconds per dataset, while Espresso-exact spends 2.017 seconds. The timings are more homogeneous for CCubes, with a standard deviation of 0.550 seconds, compared to 1.727 seconds for Espresso-exact. Using the argument `min.pin`, CCubes spends an impressive average of 0.571 seconds (over 3.5 times faster than Espresso-exact). In some cases it can be thousands of times faster, while slower in only 5.7% of cases, depending on how many complex PIs contribute to solving the PI chart (more time spent when approaching the total number of PIs).

While the cross-validation between Espresso-exact and CCubes lies in the number of found PIs, the purpose of the QCA minimization is to make sure all possible (minimal) solutions are found. From this perspective, CCubes is the only algorithm in R that can do that out of the box, but it is not the only one to operate with a high number of causal conditions.

Another software that is capable to output complete and exhaustive solutions is called Tosmana (Cronqvist, 2017), that implements another very different algorithm called GBA - Graph Based Agent. A direct comparison is not possible in this paper because Tosmana does not operate in R, but according to all tests the solutions are identical, with a superior speed for CCubes.

A cross-validation with QMC (up to 10-11 conditions), eQMC (up to 16-18 conditions), Tosmana (around 20 conditions) and Espresso-exact (up to 30 conditions but indirectly via the number of prime implicants) should be sufficient for any QCA researcher seeking a confirmation that CCubes and the other algorithms used in package **QCA** are exact and exhaustive.

Summary

A cube is understood as a multi-dimensional matrix. Just as sets can be either subsets or supersets of other sets, cubes can sometimes be formed with more complex, and sometimes with less complex cubes. The cube should be defined and interpreted relative to a certain context: it can be either a single condition, or a conjunction of causal conditions, or the entire dataset where that conjunction was extracted from. Figure 2 offers a graphical exemplification of a "cube": the simplest structure can be one single red colored component, but if thinking about an entire conjunction as a "single" component, then ∼`A*D` can be considered a cube as well, and the entire structure where this particular conjunction resides can also be considered a cube.

Such a conceptualization of a cube is important for the social sciences and it can be even better understood by noting that, at its heart, the CCubes algorithm is only an extension of Mill's joint method of agreement and difference:

> If two or more instances in which the phenomenon occurs have only one circumstance in common, while two or more instances in which it does not occur have nothing in common save the absence of that circumstance; the circumstance in which alone the two sets of instances differ, is the effect, or cause, or a necessary part of the cause, of the phenomenon.
>
> John Stuart Mill (1843) A System of Logic, vol. 1, pp. 463.

Usually, researchers think about a "circumstance" as a single causal condition, but a conjunction of causal conditions can also be considered a circumstance, or in this case a "cube". Instead of highly expensive, base to base mathematical transformations, this new algorithm merely applies the fundamentals of the comparative method.

It seems that, after all, simple is better. And a whole lot quicker.

Acknowledgements

This new algorithm appeared while preparing a presentation for the Logic Seminar, organised at the Faculty of Mathematics and Informatics, University of Bucharest. The author is grateful to all participants of that Seminar, and especially to Professor Ioana Leuștean and students Irina Gutanu and Matei Bejan, for their valuable ideas and fine tuning my C programming.

The paper was also presented at the 5th International QCA Expert Workshop in Zürich, December 2017.

Special thanks go to Lasse Cronqvist and to William Stiehl for their generous time while attempting to cross-validate the results, a number of important ideas from this paper being generated during this process.

Bibliography

R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1984. ISBN 0898381649. [p365]

L. Cronqvist. TOSMANA - TOol for SMAll-N Analysis (version 1.54). University of Trier, 2017. URL http://www.tosmana.net. [p368]

A. Dușa. A mathematical approach to the boolean minimization problem. *Quality & Quantity*, 44(1): 99–113, 2010. URL https://doi.org/10.1007/s11135-008-9183-x. [p358]

A. Dușa. *Venn: Draw Venn Diagrams*, 2018. URL https://CRAN.R-project.org/package=venn. R package version 1.7. [p365]

A. Dușa. *QCA with R. A Comprehensive Resource*. Springer-Verlag, New York, 2019. URL https://doi.org/10.1007/978-3-319-75668-4. [p364]

A. Dușa and A. Thiem. Enhancing the minimization of Boolean and multivalue output functions with eQMC. *Journal of Mathematical Sociology*, 39:92–108, 2015. URL https://doi.org/10.1080/0022250x.2014.897949. [p358]

E. J. McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 5:1417–1444, 1956. [p357]

W. V. O. Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8): 521–531, 1952. [p357]

W. V. O. Quine. A way to simplify truth functions. *The American Mathematical Monthly*, 62(9):627–631, 1955. [p357]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL http://www.R-project.org/. ISBN 3-900051-07-0. [p]

C. Ragin. *The Comparative Method. Moving beyond Qualitative and Quantitative Strategies*. University Of California Press, Berkeley, Los Angeles & London, 1987. [p357]

C. Ragin. *Fuzzy Set Social Science*. University of Chicago Press, Chicago and London, 2000. [p361]

C. C. Ragin and J. Sonnett. Between complexity and parsimony: Limited diversity, counterfactual cases, and comparative analysis. In S. Kropp and M. Minkenberg, editors, *Vergleichen in Der Politikwissenschaft*. VS Verlag für Sozialwissenschaften, 2005. URL https://doi.org/10.1007/978-3-322-80441-9_9. [p359]

W. Stiehl. *Truth Table Logic Optimizer*, 2016. URL https://CRAN.R-project.org/package=LogicOpt. R Package Version 1.0.0. [p366]

A. Thiem and A. Dușa. Introducing the QCA Package: A Market Analysis and Software Review. *Qualitative & Multi-Method Research*, 10(2):45–49, 2012. [p364]

Adrian Dusa
University of Bucharest, Department of Sociology
Soseaua Panduri nr. 90-92, sala 008, Bucharest sector 5
Romania
dusa.adrian@unibuc.ro

sdpt3r: Semidefinite Quadratic Linear Programming in R

by Adam Rahman

Abstract We present the package **sdpt3r**, an R implementation of the Matlab package SDPT3 (Toh et al., 1999). The purpose of the software is to solve semidefinite quadratic linear programming (SQLP) problems, which encompasses problems such as D-optimal experimental design, the nearest correlation matrix problem, and distance weighted discrimination, as well as problems in graph theory such as finding the maximum cut or Lovasz number of a graph.

Current optimization packages in R include **Rdsdp**, **Rcsdp**, **scs**, **cccp**, and **Rmosek**. Of these, **scs** and **Rmosek** solve a similar suite of problems. In addition to these solvers, the R packages **CXVR** and **ROI** provide sophisticated modelling interfaces to these solvers. As a point of difference from the current solvers in R, **sdpt3r** allows for log-barrier terms in the objective function, which allows for problems such as the D-optimal design of experiments to be solved with minimal modifications. The **sdpt3r** package also provides helper functions, which formulate the required input for several well-known problems, an additional perk not present in the other R packages.

Introduction

Convex optimization is a well traversed field with far reaching applications. While perhaps unfamiliar to those in the statistical sciences, many problems important to statisticians can be formulated as a convex optimization, perhaps the most well known of which would be the least squares problem. More specifically, many problems in statistics can be formulated as a subset of these convex optimization problems, known as *conic linear optimization problems*.

One such example would be the nearest correlation matrix problem (Higham, 2002), which was first considered when attempting to find correlations between stocks, where incomplete data on daily stock returns are not unusual. Pairwise correlations are only computed when data is available for both pairs of stocks under consideration, resulting in a correlation matrix that contains pairwise correlations, but is not necessarily positive semidefinite - an *approximate* correlation matrix. The goal is to then find the correlation matrix that is nearest to the approximate correlation matrix in some way.

Other examples of problems that can be formulated in terms of a conic linear optimization problem include D-optimal experimental design (Smith, 1918), classification using distance weighted discrimination (Marron et al., 2007), minimum volume ellipsoids (John, 2014), and problems in educational testing (Chu and Wright, 1995).

Problems in related fields can also be solved, including finding the maximum cut (or maximum k-cut) of a graph, finding the upper bound of the Shannon entropy of a graph, also known as the Lovasz number (Vandenberghe et al., 1998), as well as problems in control theory, Toeplitz matrix approximation, and Chebyshev approximation.

For the purpose of solving these conic linear optimization problems, we introduce the R package **sdpt3r**, an implementation of the Matlab package SDPT3 by Toh et al. (1999). Of the R packages available to perform conic optimization, **sdpt3r** is among the most general. **Rdsdp** (Zhu and Ye, 2016) & **Rcsdp** (Bravo, 2016) are capable of solving semidefinite conic optimization problems, while **cccp** (Pfaff, 2015) solves linear and quadratic conic optimization problems. The **sdpt3r** package allows for all of linear, quadratic, and semidefinite conic optimization to be solved simultaneously (i.e., a problem with any combination of semidefinite, quadratic, or linear cones can be solved). Two comparable packages, **scs** (O'Donoghue et al., 2017) and **Rmosek** (Friberg, 2012), solve a similar suite of problems. Additionally, the R packages **CXVR** (Fu et al., 2017) and **ROI** (Theußl et al., 2017) provide sophisticated modelling interfaces to these solvers.

As a point of difference, **scs** and **Rmosek** allow for the exponential and power cones to be included in the constraints, while **sdpt3r** handles log-barrier terms in the objective function directly. The inclusion of log-barrier terms allows for the D-optimal design of experiments and minimum volume ellipsoid problems to be solved with minimal modifications. In addition, **sdpt3r** provides helper functions which directly solve a number of well known problems (such as the "max cut" or nearest correlation matrix problems) with minimal input burden on the user. This additional functionality is not found in either **scs** or **Rmosek** (although **scs** can be used with the **CVXR** package).

This paper is structured as follows. In Section 52.2 we discuss in greater detail the mathematical formulation of the linear conic optimization problem, and introduce three examples to explore the increasing generality of the problem to be solved. Section 52.3 discusses the R implementation of

sdpt3r, and the main function by which conic linear optimization problems are solved, `sqlp`, including the required input, and the output generated. The same examples used in Section 52.2 will be used to demonstrate how a standard conic linear optimization problem can be converted to a form solvable by `sqlp`. Section 52.4 presents the classic form of several other well known problems that can be solved using **sdpt3r**, as well as the helper functions available to convert them to the appropriate form. Finally Section 52.5 provides some closing remarks.

Conic linear optimization

At its simplest, a conic linear optimization problem has the following standard form (Tütüncü et al., 2003):

$$\begin{array}{lrcl} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C},\, \mathbf{X} \rangle & & \\ \text{subject to} & & & \\ & \langle \mathbf{A}_k,\, \mathbf{X} \rangle & = & \mathbf{b}_k, \quad k = 1, ..., m \\ & \mathbf{X} & \in & \mathcal{K} \end{array} \tag{1}$$

where $\mathcal{K}$ is a cone. Generally, $\mathcal{K}$ is either a

- Semidefinite Cone - $\mathcal{S}^n = \{\mathbf{X} \in \mathcal{R}^{n \times n} : \mathbf{X} \succeq 0, \mathbf{X}_{ij} = \mathbf{X}_{ji}\ \forall\ i \neq j\}$
- Quadratic Cone - $\mathcal{Q}^n = \{\mathbf{x} = [x_0; \tilde{\mathbf{x}}] \in \mathcal{R}^n : x_0 \geq \sqrt{\tilde{\mathbf{x}}^\mathsf{T} \tilde{\mathbf{x}}}\}$
- Linear Cone - $\mathcal{L}^n$ - non-negative orthant of $\mathcal{R}^n$

Here, $\tilde{\mathbf{x}} = [x_1, \ldots, x_{n-1}]$, and $\langle \cdot, \cdot \rangle$ represents the standard inner product in the appropriate space. In the semidefinite cone the inner product is $\langle \mathbf{X},\, \mathbf{Y} \rangle = vec(\mathbf{X})^\mathsf{T} vec(\mathbf{Y})$, where the operator *vec* is the by-column vector version of the matrix $\mathbf{X}$, that is, for the $n \times n$ matrix $\mathbf{X} = [x_{ij}]$, $vec(\mathbf{X})$ is the $n^2 \times 1$ vector $[x_{11}, x_{12}, x_{13}, \ldots, x_{(n-1)n}, x_{nn}]^\mathsf{T}$. Note that *vec* does not require a square matrix in general.

One of the simplest problems that can be formulated in terms of a conic linear optimization problem is finding the maximum cut of a graph. Let $\mathbf{G} = [\mathbf{V}, \mathbf{E}]$ be a graph with vertices $\mathbf{V}$ and edges $\mathbf{E}$. A *cut* of the graph $\mathbf{G}$ is a partition of the vertices of $\mathbf{G}$ into two disjoint subsets $\mathbf{G}_1 = [\mathbf{V}_1, \mathbf{E}_1]$, $\mathbf{G}_2 = [\mathbf{V}_2, \mathbf{E}_2]$, with $\mathbf{V}_1 \cap \mathbf{V}_2 = \varnothing$. The size of the cut is defined to be the number of edges connecting the two subsets. The *maximum cut* is defined to be the cut of a graph $\mathbf{G}$ whose size is at least as large as any other cut. For a weighted graph object, we can also define the maximum cut to be the cut with weight at least as large as any other cut.

Finding the maximum cut is referred to as the **Max-Cut Problem**, and was one of the first problems found to be NP-complete, and is also one of the 21 algorithms on Karp's 21 NP-complete problems (Karp, 1972). The Max-Cut problem is also known to be *APX hard* (Papadimitriou and Yannakakis, 1991), meaning in addition to there being no polynomial time solution, there is also no polynomial time approximation.

Using the semidefinite programming approximation formulation of Goemans and Williamson (1995), the Max-Cut problem can be approximated to within an *approximation constant*. For a weighted adjacency matrix $\mathbf{B}$, the objective function can be stated as

$$\begin{array}{lrcl} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C}, \mathbf{X} \rangle & & \\ \text{subject to} & & & \\ & diag(\mathbf{X}) & = & \mathbf{1} \\ & \mathbf{X} & \in & \mathcal{S}^n \end{array}$$

where $\mathcal{S}^n$ is the cone of symmetric positive semidefinite matrices of size n, and $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. Here, we define $diag(\mathbf{a})$ for an $n \times 1$ vector $\mathbf{a}$ to be the diagonal matrix $\mathbf{A} = [A_{ij}]$ of size $n \times n$ with $A_{ii} = a_i,\ \ i = 1, \ldots, n$. For a matrix $\mathbf{X}$, $diag(\mathbf{X})$ extracts the diagonal elements from $\mathbf{X}$ and places them in a column-vector.

To see that the Max-Cut problem is a conic linear optimization problem it needs to be written in the same form as Equation 1. The objective function is already in a form identical to that of Equation 1, with minimization occurring over $\mathbf{X}$ of its inner product with a constant matrix $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. There are n equality constraints of the form $x_{kk} = 1,\ \ k = 1, ..., n$, where x_{kk} is the k^{th} diagonal element of $\mathbf{X}$, and $b_k = 1$ in Equation 1. To represent this in the form $\langle \mathbf{A}_k,\, \mathbf{X} \rangle = x_{kk}$, take $\mathbf{A}_k$ to be

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

Now $\langle \mathbf{A}_k, \mathbf{X}\rangle = vec(\mathbf{A}_k)^{\mathsf{T}} vec(\mathbf{X}) = x_{kk}$ as required, and the Max-Cut problem is specified as a conic linear optimization problem.

Allowing for optimization to occur over only one variable at a time is quite restrictive, as only a small number of problems can be formulated in this form. Allowing optimization to occur over multiple variables simultaneously would allow for a broader range of problems to be solved.

A separable set of variables

The conic linear optimization problem actually covers a much wider class of problems than those expressible as in Equation 1. Variables can be separated into those which are constrained to a semidefinite cone, $\mathcal{S}$, a quadratic cone, $\mathcal{Q}$, or a linear cone, $\mathcal{L}$. The objective function is a sum of the corresponding inner products of each set of variables. The linear constraint is simply a sum of variables of linear functions of each set. This more general version of the conic linear optimization problem is

$$\begin{array}{ll} \underset{\mathbf{X}^s,\mathbf{X}^q,\mathbf{X}^l}{\text{minimize}} & \sum_{j=1}^{n_s}\langle \mathbf{C}_j^s, \mathbf{X}_j^s\rangle + \sum_{i=1}^{n_q}\langle \mathbf{C}_i^q, \mathbf{X}_i^q\rangle + \langle \mathbf{C}^l, \mathbf{X}^l\rangle \\ \text{subject to} & \\ & \begin{array}{rcl} \sum_{j=1}^{n_s} (\mathbf{A}_j^s)^{\mathsf{T}} svec(\mathbf{X}_j^s) + \sum_{i=1}^{n_q} (\mathbf{A}_i^q)^{\mathsf{T}} \mathbf{X}_i^q + (\mathbf{A}^l)^{\mathsf{T}} \mathbf{X}^l & = & \mathbf{b} \\ \mathbf{X}_j^s & \in & \mathcal{S}^{s_j}\ \forall j \\ \mathbf{X}_i^q & \in & \mathcal{Q}^{q_i}\ \forall i \end{array} \end{array} \tag{2}$$

Here, *svec* takes the upper triangular elements of a matrix (including the diagonal) in a column-wise fashion and vectorizes them. In general for an $n \times p$ matrix $\mathbf{X} = [x_{ij}]$, $svec(\mathbf{X})$ will have the following form $[x_{11}, x_{12}, x_{22}, x_{13}, ..., x_{(n-1)p}, x_{np}]^{\mathsf{T}}$. Recall that matrices in $\mathcal{S}$ are symmetric, so it is sufficient to constrain only the upper triangular elements of the matrix $\mathbf{X}^s$. For this formulation, $\mathbf{A}_j^s$, $\mathbf{A}_i^q$ and $\mathbf{A}^l$ are the constraint matrices of the appropriate size.

Some important problems in statistics can be formulated to fit this form of the optimization problem.

The nearest correlation matrix

First addressed by Higham (2002) in dealing with correlations between stock prices, difficulty arises when data is not available for all stocks on each day, which is unfortunately a common occurrence. To help address this situation, correlations are calculated for pairs of stocks only when data is available for both stocks on any given day. The resulting correlation matrix is only approximate in that it is not necessarily positive semidefinite.

This problem was cast by Higham (2002) as

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & ||\mathbf{R} - \mathbf{X}||_F \\ \text{subject to} & \\ & \begin{array}{rcl} diag(\mathbf{X}) & = & \mathbf{1} \\ \mathbf{X} & \in & \mathcal{S}^n \end{array} \end{array}$$

where $\mathbf{R}$ is the approximate correlation matrix and $||\cdot||_F$ denotes the Frobenius norm. Unfortunately, the Frobenius norm in the objective function prevents the problem being formatted as a conic linear optimization problem.

Since the matrix $\mathbf{X}$ is constrained to have unit diagonal and be symmetric, and the matrix $\mathbf{R}$ is an approximate correlation matrix, meaning it will also have unit diagonal and be symmetric, we can re-write the objective function as

$$||\mathbf{R} - \mathbf{X}||_F = 2 * ||svec(\mathbf{R}) - svec(\mathbf{X})|| = 2 * ||\mathbf{e}||$$

Now, introduce a variable e_0 such that $e_0 \geq ||\mathbf{e}||$, and define $\mathbf{e}^* = [e_0;\ \mathbf{e}]$. The vector $\mathbf{e}^*$ is now restricted to be in the quadratic cone $\mathcal{Q}^{n(n+1)/2+1}$. This work leads to the formulation of Toh et al. (1999)

$$\begin{array}{lrcl}
\underset{\mathbf{e}^*,\,\mathbf{X}}{\text{minimize}} & e_0 & & \\
\text{subject to} & & & \\
 & svec(\mathbf{R}) - svec(\mathbf{X}) & = & [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\ \mathbf{e}^* \\
 & diag(\mathbf{X}) & = & \mathbf{1} \\
 & \mathbf{X} & \in & \mathcal{S}^n \\
 & \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1}
\end{array}$$

Here, $[\mathbf{X}, \mathbf{Y}]$ denotes column binding of the two matrices $\mathbf{X}_{n\times p}$ and $\mathbf{Y}_{n\times m}$ to form a matrix of size $n \times (p+m)$. By minimizing e_0, we indirectly minimize $\mathbf{e} = svec(\mathbf{R}) - svec(\mathbf{X})$, since recall we have $e_0 \geq ||\mathbf{e}||$, which is the goal of the original objective function.

To see this as a conic linear optimization problem, notice that e_0 can be written as $\langle \mathbf{C}^q,\ \mathbf{X}^q \rangle$ by letting $\mathbf{C}^q = [1; \mathbf{0}_{n(n+1)/2}]$ and $\mathbf{X}^q = \mathbf{e}^*$. Since the matrix $\mathbf{X}$ (i.e., $\mathbf{X}^s$) does not appear in the objective function, the matrix $\mathbf{C}^s$ is an $n \times n$ matrix of zeros.

Re-writing the first constraint as

$$svec(\mathbf{X}) + [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\ \mathbf{e}^* = \ svec(\mathbf{R})$$

we can easily define the constraint matrices and right hand side of the first constraint as

$$\begin{array}{ll}
\mathbf{A}_1^s & = \mathbf{I}_{n(n+1)/2} \\
\mathbf{A}_1^q & = [\mathbf{0}, \mathbf{I}_{n(n+1)/2}] \\
\mathbf{b}_1 & = svec(\mathbf{R})
\end{array}$$

The second constraint is identical to the constraint from the Max-Cut problem, where each diagonal element of $\mathbf{X}$ is constrained to be equal to 1. Define $\mathbf{b}_2 = \mathbf{1}$, and for the k^{th} diagonal element of $\mathbf{X}$, define the matrix $\mathbf{A}_k$ as

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

yielding $\langle \mathbf{A}_k, \mathbf{X} \rangle = x_{kk}$. To write this as $(\mathbf{A}_2^s)^\mathsf{T}\mathbf{X}^s$, define

$$\mathbf{A}_2^s = [svec(\mathbf{A}_1), ..., svec(\mathbf{A}_n)]$$

Since $\mathbf{e}^*$ does not appear in the second constraint, $\mathbf{A}_2^q = \mathbf{0}_{n(n+1)/2+1}$.

The final step is to combine the individual constraint matrices from each constraint to form one constraint matrix for each variable, which is done by defining $\mathbf{A}^s = [\mathbf{A}_1^s,\ \mathbf{A}_2^s]$, $\mathbf{A}^q = [\mathbf{A}_1^q,\ \mathbf{A}_2^q]$. We also concatenate both right hand side vectors to form a single vector by defining $\mathbf{b} = [\mathbf{b}_1;\ \mathbf{b}_2]$. Here, the notation $[\mathbf{X}; \mathbf{Y}]$ is used to denote two matrices $\mathbf{X}_{p\times m}$ and $\mathbf{Y}_{q\times m}$ bound vertically to form a matrix of size $(p+q) \times m$. With this, the nearest correlation matrix problem is written as a conic linear optimization.

Semidefinite quadratic linear programming

While Equation 2 allows for additional variables to be present, it can be made more general still to allow even more problems to be solved. We will refer to this general form as a *semidefinite quadratic linear programming* (SQLP) problem.

The first generality afforded by an SQLP is the addition of an unconstrained variable $\mathbf{X}^u$, which, as the name suggests, is not bound to a cone, but instead, it is "constrained" to the reals in the appropriate dimension. The second generalization is to allow for what are known as *log-barrier* terms to exist in the objective function. In general, a barrier function in an optimization problem is a term that approaches infinity as the point approaches the boundary of the feasible region. As we will see, these log-barrier terms appear as log terms in the objective function.

Recall that for any linear optimization problem, there exists two formulations - the primal formulation and the dual formulation. For the purposes of a semidefinite quadratic linear programming problem, the primal problem will always be defined as a minimization, and the associated dual problem will therefore be a maximization

The primal problem

The primal formulation of the SQLP problem is

$$\begin{array}{ll} \underset{\mathbf{X}_j^s,\mathbf{X}_i^q,\mathbf{X}^l,\mathbf{X}^u}{\text{minimize}} & \sum_{j=1}^{n_s}[\langle \mathbf{C}_j^s,\, \mathbf{X}_j^s\rangle - \mathbf{v}_j^s\, log\; det\, \mathbf{X}_j^s] \;+\; \sum_{i=1}^{n_q}[\langle \mathbf{C}_i^q,\, \mathbf{X}_i^q\rangle - \mathbf{v}_i^q\, log\; \gamma(\mathbf{X}_i^q)] \\ & \quad +\, \langle \mathbf{C}^l,\, \mathbf{X}^l\rangle \;-\; \sum_{k=1}^{n_l} \mathbf{v}_k^l\, log\, \mathbf{X}_k^l \;+\; \langle \mathbf{C}^u,\, \mathbf{X}^u\rangle \\ \text{subject to} & \\ & \begin{array}{rcll} \sum_{j=1}^{n_s} \mathbf{A}_j^s(\mathbf{X}_j^s) + \sum_{i=1}^{n_q} \mathbf{A}_i^q \mathbf{X}_i^q + \mathbf{A}^l\mathbf{X}^l + \mathbf{A}^u\mathbf{X}^u & = & \mathbf{b} & \\ \mathbf{X}_j^s & \in & \mathcal{S}^{s_j} & \forall\, j \\ \mathbf{X}_i^q & \in & \mathcal{Q}^{q_i} & \forall\, i \\ \mathbf{X}^l & \in & \mathcal{L}^{n_l} & \\ \mathbf{X}^u & \in & \mathcal{R}^{n_u} & \end{array} \end{array} \tag{3}$$

For each j, $\mathbf{C}_j^s$ and $\mathbf{X}_j^s$ are symmetric matrices of dimension s_j, restricted to the cone of positive semidefinite matrices of the same dimension. Similarly, for all i, $\mathbf{C}_i^q$ and $\mathbf{X}_i^q$ are real vectors of dimension q_i, restricted to the quadratic cone of dimension q_i. For a vector $\mathbf{u} = [u_0; \tilde{\mathbf{u}}]$ in a second order cone, define $\gamma(u) = \sqrt{u_0^2 - \tilde{\mathbf{u}}^\mathsf{T}\tilde{\mathbf{u}}}$. Finally, $\mathbf{C}^l$ and $\mathbf{X}^l$ are vectors of dimension n_l, restricted to linear cone of the same dimension, and $\mathbf{C}^u$ and $\mathbf{X}^u$ are unrestricted real vectors of dimension n_u.

As before, the matrices $\mathbf{A}_i^q$, $\mathbf{A}^l$, and $\mathbf{A}^u$ are constraint matrices in q_i, n_l, and n_u dimensions respectively, each corresponding to their respective quadratic, linear, or unrestricted block. $\mathbf{A}_j^s$ is defined to be a linear map from $\mathcal{S}^{s_j}$ to $\mathcal{R}^m$ defined by

$$\mathbf{A}_j^{s_j}(\mathbf{X}_j^s) = [\langle \mathbf{A}_{j,1}^s, \mathbf{X}_j^s\rangle; \ldots; \langle \mathbf{A}_{j,m}^s, \mathbf{X}_j^s\rangle]$$

where $\mathbf{A}_{j,1}^s \ldots \mathbf{A}_{j,m}^s \in \mathcal{S}^{s_j}$ are constraint matrices associated with the j^{th} semidefinite variable $\mathbf{X}_j^s$.

The dual problem

The dual problem associated with the semidefinite quadratic linear programming formulation is

$$\begin{array}{lll} \underset{\mathbf{Z}_j^s,\mathbf{Z}_i^q,\mathbf{Z}^l,\mathbf{y}}{\text{maximize}} & \mathbf{b}^\mathsf{T}\mathbf{y} & +\; \sum_{j=1}^{n_s}[\mathbf{v}_j^s\, log\; det\, \mathbf{Z}_j^s \;+\; s_j\, \mathbf{v}_j^s\, (1 - log\, \mathbf{v}_j^s)] \\ & & +\; \sum_{i=1}^{n_q}[\mathbf{v}_i^q\, log\, \gamma(\mathbf{Z}_i^q) \;+\; \mathbf{v}_i^q\, (1 - log\, \mathbf{v}_i^q)] \\ & & +\; \sum_{k=1}^{n_l}[\mathbf{v}_k^l\, log\, \mathbf{Z}_k^l \;+\; \mathbf{v}_k^l\, (1 - log\, \mathbf{v}_k^l)] \\ \text{subject to} & & \\ & & \begin{array}{lllll} (\mathbf{A}_j^s)^\mathsf{T}\mathbf{y} & +\; \mathbf{Z}_j^s & = \mathbf{C}_j^s, & \mathbf{Z}_j^s & \in \mathcal{S}^{s_j}, \quad j = 1,\ldots,n_s \\ (\mathbf{A}_i^q)^\mathsf{T}\mathbf{y} & +\; \mathbf{Z}_i^q & = \mathbf{C}_i^q, & \mathbf{Z}_i^q & \in \mathcal{Q}^{q_i}, \quad i = 1,\ldots,n_q \\ (\mathbf{A}^l)^\mathsf{T}\mathbf{y} & +\; \mathbf{Z}^l & = \mathbf{C}^l, & \mathbf{Z}^l & \in \mathcal{L}^{n_l} \\ (\mathbf{A}^u)^\mathsf{T}\mathbf{y} & & = \mathbf{C}^u, & \mathbf{y} & \in \mathcal{R}^m \end{array} \end{array} \tag{4}$$

where $(\mathbf{A}_j^s)^T$ is defined to be the adjoint operator of $\mathbf{A}_j^s$, where $(\mathbf{A}_j^s)^\mathsf{T}\mathbf{y} = \sum_{k=1}^m \mathbf{y}_k \mathbf{A}_{j,k}^s$. Equations 3 and 4 represent the most general form of the linear conic optimization problem that can be solved using **sdpt3r**.

Optimal design of experiments

Consider the problem of estimating a vector $\mathbf{x}$ from measurements $\mathbf{y}$ given by the relationship

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0,1).$$

The variance-covariance matrix of such an estimator is proportional to $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$. A reasonable goal during the design phase of an experiment would therefore be to minimize $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ in some way.

There are many different ways in which $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ might be made minimal. For example, minimization of the trace of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (A-Optimality), minimization of the maximum eigenvalue of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (E-Optimality), minimization of the determinant of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (D-Optimality), and maximization of the trace of $\mathbf{A}^\mathsf{T}\mathbf{A}$ (T-Optimality) all have their merits.

Perhaps the most commonly used of these optimality criteria is D-Optimality, which is equivalent to maximizing the determinant of $\mathbf{A}^\mathsf{T}\mathbf{A}$. Typically, the rows of $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_q]^T$ are chosen from M possible test vectors $\mathbf{u}_i \in \mathcal{R}^p, \; i = 1, ...M$, which are known in advance. That is,

$$\mathbf{a}_i \in \{\mathbf{u}_1, ..., \mathbf{u}_M\}, \quad i = 1, ..., q$$

Given that the matrix $\mathbf{A}$ is made up of these test vectors $\mathbf{u}_i$, Vandenberghe et al. (1998) write the matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$ as

$$\mathbf{A}^\mathsf{T}\mathbf{A} = q\sum_{i=1}^{M}\lambda_i\mathbf{u}_i\mathbf{u}_i^\mathsf{T} \tag{5}$$

where λ_i is the fraction of rows in $\mathbf{A}$ that are equal to the vector $\mathbf{u}_i$. Then, Vandenberghe et al. (1998) write the D-optimal experimental design problem as a minimum determinant problem

$$\begin{array}{ll} \underset{\lambda}{\text{minimize}} & \log\det\,(\sum_{i=1}^{M}\lambda_i\mathbf{u}_i\mathbf{u}_i^\mathsf{T})^{-1} \\ \text{subject to} & \\ & \begin{array}{rcl} \lambda_i & \geq & 0, \quad i = 1, ..., m \\ \sum_{i=1}^{M}\lambda_i & = & 1 \end{array} \end{array}$$

Due to the inequality constraint, this primal formulation cannot be interpreted as an SQLP of the form of Equation 3. By defining $\mathbf{Z} = \mathbf{u}\; diag(\lambda)\; \mathbf{u}^\mathsf{T}$, the dual problem is (Toh et al., 1999)

$$\begin{array}{ll} \underset{\mathbf{Z},\,\mathbf{z}^l,\,\lambda}{\text{maximize}} & \log\det\,(\mathbf{Z}) \\ \text{subject to} & \\ & \begin{array}{rcll} -\sum_{i=1}^{p}\lambda_i(\mathbf{u}_i\mathbf{u}_i^\mathsf{T}) + \mathbf{Z} & = & 0, & \mathbf{Z} \in \mathcal{S}^n \\ -\boldsymbol{\lambda} + \mathbf{z}^l & = & 0, & \mathbf{z}^l \in \mathcal{R}_+^p \\ \mathbf{1}^T\boldsymbol{\lambda} & = & 1, & \boldsymbol{\lambda} \in \mathcal{R}^p \end{array} \end{array}$$

Keeping in mind that this is a dual configuration, and thus follows Equation 4, we proceed with writing the D-Optimal design problem as an SQLP by first considering the objective function. The objective function depends only on the determinant of the matrix variable $\mathbf{Z}$, which is the log-barrier. This indicates that the variable v^s in Equation 4 is equal to 1 in this formulation, while v^q and v^l are both zero. Since $\boldsymbol{\lambda}$ does not appear in the objective function, the vector $\mathbf{b}$ is equal to $\mathbf{0}$.

The constraint matrices $\mathbf{A}$ are easy to define in the case of the dual formulation, as they multiply the vector $\mathbf{y}$ in Equation 4, so therefore multiply $\boldsymbol{\lambda}$ in our case. In the first constraint, each λ_i is multiplied by the matrix formed by $-\mathbf{u}_i\mathbf{u}_i^\mathsf{T}$, so define $\mathbf{A}_i$ to be

$$\mathbf{A}_i = -\mathbf{u}_i\mathbf{u}_i^\mathsf{T}, \;\; i = 1, ..., p.$$

Then, the constraint matrix is $\mathbf{A}^s = [svec(\mathbf{A}_1), ..., svec(\mathbf{A}_p)]$. In the second constraint containing the linear variable $\mathbf{z}^l$, the constraint matrix is $\mathbf{A}^l = -\mathbf{I}_p$, and in the third constraint containing only the unconstrained variable $\boldsymbol{\lambda}$, the constraint matrix is $\mathbf{A}^u = \mathbf{1}^\mathsf{T}$. Since there is no quadratic variable, $A^q = \mathbf{0}$.

Finally, define the right hand side of each constraint

$$\begin{array}{ll} \mathbf{C}^s & = \mathbf{0}_{n\times n} \\ \mathbf{C}^l & = \mathbf{0}_{p\times 1} \\ \mathbf{C}^u & = 1 \end{array}$$

which fully specifies the D-Optimal design problem as an SQLP.

In the next section, we will demonstrate using R how these definitions can be translated for use in the main function of **sdpt3r** so an SQLP problem can be solved.

Solving a conic linear optimization problem with sdpt3r

Each of the problems presented in Section 52.2 can be solved using the **sdpt3r** package, an R implementation of the Matlab program SDPT3. The algorithm is an infeasible primal-dual predictor-corrector path-following method, utilizing either an HKM (Helmberg et al., 1996) or NT (Nesterov and Todd, 1997) search direction. The interested reader is directed to Tütüncü et al. (2003) for further details surrounding the implementation.

The main function available in **sdpt3r** is `sqlp`, which takes a number of inputs (or an `sqlp_input` object) specifying the problem to be solved, and executes the optimization, returning both the primal and dual solution to the problem. This function will be thoroughly discussed in Section 52.3.1, and examples will be provided. In addition to `sqlp`, a prospective user will also have access to a number of helper functions for well known problems that can be solved using **sdpt3r**. For example, the function `maxcut` takes as input an adjacency matrix $\mathbf{B}$, and produces an S3 object containing all the input variables necessary to solve the problem using `sqlp`. These functions will be discussed in Sections 52.3.3, 52.3.4, 52.3.4, and 52.4.

For **sdpt3r**, each optimization variable will be referred to as a *block* in the space in which it is restricted. For instance, if we have an optimization variable $\mathbf{X} \in \mathcal{S}^n$, we will refer to this as a semidefinite block of size n. It is important to note that it is possible to have multiple blocks from the same space, that is, it is possible to have both $\mathbf{X} \in \mathcal{S}^n$ as well as $\mathbf{Y} \in \mathcal{S}^m$ in the same problem.

Input variables

The main function call in **sdpt3r** is `sqlp`, which takes the following input variables

`blk`	A named-list object describing the block structure of the optimization variables.
`At`	A list object containing constraint matrices $\mathbf{A}^s$, $\mathbf{A}^q$, $\mathbf{A}^l$, and $\mathbf{A}^u$ for the primal-dual problem.
`b`	A vector containing the right hand side of the equality constraints, $\mathbf{b}$, in the primal problem, or equivalently the constant vector in the dual.
`C`	A list object containing the constant $\mathbf{C}$ matrices in the primal objective function or equivalently the corresponding right hand side of the equality constraints in the dual problem.
`X0, y0, Z0`	Matrix objects containing an initial iterate for the $\mathbf{X}$, $\mathbf{y}$, and $\mathbf{Z}$ variables for the SQLP problem. If not provided, an initial iterate is computed internally.
`control`	A list object providing additional parameters for use in `sqlp`. If not provided, default values are used.

The input variable `blk` describes the block structure of the problem. Letting `L` be the total number of semidefinite, quadratic, linear, and unrestricted blocks in the SQLP problem, define `blk` to be a named-vector object of length L, with names describing the type of block, and values denoting the size of the optimization variable, summarized in Table 1.

Block type	Name	Value
Semidefinite	s	s_j
Quadratic	q	q_i
Linear	l	n_l
Unrestricted	u	n_u

Table 1: *Structure of* `blk`.

The input variable `At` corresponds to the constraint matrices in Equation 3, and `C` the constant matrices in the objective function. The size of these input variables depends on the block they are representing, summarized in Table 2 for each block type.

	Block type			
	Semidefinite	Quadratic	Linear	Unrestricted
`At`	$\bar{s}_j \times m$	$q_j \times m$	$n_l \times m$	$n_u \times m$
`C`	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$

Table 2: *Size of* `At` *and* `C` *for each block type.*

Note that in Table 2, $\bar{s}_j = s_j(s_j + 1)/2$. The size of `At` in the semidefinite block reflects the upper-triangular input format that has been discussed previously. In a semidefinite block, the optimization variable $\mathbf{X}$ is necessarily symmetric and positive semidefinite, it is therefore more efficient to consider only the upper-triangular portion of the corresponding constraint matrix.

It is important to note that both input variables `At` and `C` are lists containing constraint and constant matrices for each optimization variable. In general, the user need not supply initial iterates `X0`, `y0`, and

`Z0` for a solution to be found using `sqlp`. The infeasible starting point generated internally by `sqlp` tends to be sufficient to find a solution. If the user wishes to provide a starting point however, the size parameters in Table 3 must be met for each block.

	Block type			
	Semidefinite	Quadratic	Linear	Unrestricted
`X0`	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$
`y0`	$s_j \times 1$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$
`Z0`	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$

Table 3: *Required size for initial iterates* `X0`, `y0`, *and* `Z0`.

The user may choose to depart from the default values of several parameters which could affect the optimization by specifying alternative values in the `control` list. A complete list of all parameters that can be altered can be found in Appendix 52.6.

An important example is the specification of the `parbarrier` parameter in `control`, which specifies the presence of a log-barrier in the objective function. The default case in `control` assumes that the parameters $\mathbf{v}_j^s$, $\mathbf{v}_i^q$, $\mathbf{v}_k^l$ in Equation 3 are all $\mathbf{0}$. If this, however, is not the case, then the user must specify an $L \times 1$ matrix object in `control$parbarrier` to store the values of these parameters (including zeros). If the j^{th} block is a semidefinite block containing p variables, $parbarrier_j = [v_{j1}^s, ..., v_{jn}^s]$. If the j^{th} block is a quadratic block containing p variables, $parbarrier_j = [v_{j1}^q, ..., v_{jn}^q]$. If the j^{th} block is a linear block $parbarrier_j = [v_1^l, ..., v_{n_l}^l]$. Finally, if the j^{th} block is the unrestricted block, then $parbarrier_j = [0, ..., 0]$, where 0 is repeated n_u times.

When executed, `sqlp` simultaneously solves both the primal and dual problems, meaning solutions for both problems are returned. The relevance of each output therefore depends on the problem being solved. The following object of class `sqlp_output` is returned upon completion

`pobj`	the value of the primary objective function
`dobj`	the value of the dual objective function
`X`	A list object containing the optimal matrix **X** for the primary problem
`y`	A vector object containing the optimal vector **y** for the dual problem
`Z`	A list object containing the optimal matrix **Z** for the dual problem

The examples in subsequent subsubsections will demonstrate the output provided by `sqlp`.

Toy Examples

Before moving on to more complex problems, consider first some very simple example to illustrate the functionality of the **sdpt3r** package. First, consider the following simple linear programming problem:

$$\begin{array}{ll} \text{Minimize} & x_1 + x_2 \\ \text{subject to} & \\ & \begin{array}{rcl} x_1 + 4x_2 & = & 12 \\ 3x_1 - x_2 & = & 10 \end{array} \end{array}$$

This problem can be solved using **sdpt3r** in very straightforward fashion. First, this is a linear programming problem with two variables, x_1 and x_2. This implies that `blk = c("l" = 2)`. Next the objective function can be written as $1 * x_1 + 1 * x_2$, so `C = matrix(c(1,1),nrow=1)`. The constraints can be summarized in matrix form as:

$$A = \begin{bmatrix} 1 & 4 \\ 3 & -1 \end{bmatrix}$$

so `A = matrix(c(1,3,4,-1),nrow=2))` and `At = t(A)`. Finally the right hand side can be written in vector form as $[12, 10]$, so `b = c(12,10)`. Pulling these all together, the problem is solved using `sqlp`:

```
blk = c("l" = 2)
C = matrix(c(1,1),nrow=1)
A = matrix(c(1,3,4,-1), nrow=2)
At = t(A)
b = c(12,10)
```

```
out = sqlp(blk,list(At),list(C),b)
out

$X
$X[[1]]
2 x 1 Matrix of class "dgeMatrix"
     [,1]
[1,]    4
[2,]    2


$y
          [,1]
[1,] 0.3076923
[2,] 0.2307692

$Z
$Z[[1]]
2 x 1 Matrix of class "dgeMatrix"
             [,1]
[1,] 6.494441e-10
[2,] 1.234448e-09


$pobj
[1] 6

$dobj
[1] 6
```

which returns the solution $x_1 = 4$ and $x_2 = 2$, and the optimal primal solution of 6. Second, consider the following simple quadratic programming problem:

$$\begin{array}{lrcl} \text{Minimize} & \frac{1}{2}x_1^2 - x_2^2 & & \\ \text{subject to} & & & \\ & 2x_1 - x_2 & = & 5 \\ & x_1 + x_2 & = & 4 \end{array}$$

This problem can be solved using **sdpt3r** by formulating the input variables in a similar fashion as the linear programming problem:

```
blk = c("q" = 2)
C = matrix(c(0.5,-1),nrow=1)
A = matrix(c(2,1,-1,1), nrow=2)
At = t(A)
b = c(5,4)

out = sqlp(blk,list(At),list(C),b)
out

$X
$X[[1]]
2 x 1 Matrix of class "dgeMatrix"
     [,1]
[1,]    3
[2,]    1


$y
     [,1]
[1,]  0.5
[2,] -0.5

$Z
$Z[[1]]
```

```
2 x 1 Matrix of class "dgeMatrix"
              [,1]
[1,]  2.186180e-09
[2,] -3.522956e-10


$pobj
[1] 0.5

$dobj
[1] 0.5
```

which returns the solution $x_1 = 3$ and $x_2 = 1$, with optimal primal solution of 0.5. Finally, consider the following simple semidefinite programming problem (taken from Freund (2004)):

$$\begin{array}{ll} \text{Minimize} & \begin{bmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \\ \text{subject to} & \\ & \begin{bmatrix} 1 & 0 & 1 \\ 0 & 3 & 7 \\ 1 & 7 & 5 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} = 11 \\ & \begin{bmatrix} 0 & 2 & 8 \\ 2 & 6 & 0 \\ 8 & 0 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} = 9 \end{array}$$

This problem is written almost exactly in the language used by **sdpt3**, and so can be easily solved by taking:

```
blk = c("s" = 3)
C = list(matrix(c(1,2,3,2,9,0,3,0,7), nrow=3))
A1 = matrix(c(1,0,1,0,3,7,1,7,5), nrow=3)
A2 = matrix(c(0,2,8,2,6,0,8,0,4), nrow=3)
At = svec(blk,list(A1,A2))
b = c(11,9)

out = sqlp(blk,At,C,b)
out

$X
$X[[1]]
3 x 3 Matrix of class "dgeMatrix"
           [,1]      [,2]      [,3]
[1,] 0.08928297 0.1606827 0.2453417
[2,] 0.16068265 0.2891815 0.4415426
[3,] 0.24534167 0.4415426 0.6741785



$y
          [,1]
[1,] 0.5172462
[2,] 0.4262486

$Z
$Z[[1]]
3 x 3 Matrix of class "dsyMatrix"
           [,1]      [,2]       [,3]
[1,]  0.4827538  1.147503 -0.9272352
[2,]  1.1475028  4.890770 -3.6207235
[3,] -0.9272352 -3.620723  2.7087744


$pobj
[1] 9.525946
```

```
$dobj
[1] 9.525946
```

which provides the optimal matrix solution X, and the optimal value of the objective function 9.53. Note that the function svec is used since the problem is a semidefinite programming problem, and thus each A matrix is necessarily symmetric.

The Max-Cut problem

Recall that the maximum cut of a graph **G** with adjacency matrix **B** can be found as the solution to

$$\begin{array}{lrcl} \text{Minimize} & \langle \mathbf{C}, \mathbf{X} \rangle & & \\ \text{subject to} & & & \\ & diag(\mathbf{X}) & = & \mathbf{1} \\ & \mathbf{X} & \in & \mathcal{S}^n \end{array}$$

where $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. In Section 52.2, we wrote this in the form of an SQLP

$$\begin{array}{lrcl} \text{Minimize} & \langle \mathbf{C}, \mathbf{X} \rangle & & \\ \text{subject to} & & & \\ & \langle \mathbf{A}_k, \mathbf{X} \rangle & = & 1, \quad k = 1, \ldots, n \\ & \mathbf{X} & \in & \mathcal{S}^n \end{array}$$

where we defined $\mathbf{A}_k$ as

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

To convert this to a form usable by sqlp, we begin by noting that we have one optimization variable, **X**, and therefore $L = 1$. For an adjacency matrix B of dimension n for which we would like to determine the Max-Cut, **X** is constrained to the space of semidefinite matrices of size n. Therefore, for a 10×10 matrix B (as in Figure 1), blk is specified as

```
B <- rbind(c(0, 0, 0, 1, 0, 0, 1, 1, 0, 0),
           c(0, 0, 0, 1, 0, 0, 1, 0, 1, 1),
           c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
           c(1, 1, 0, 0, 0, 0, 0, 1, 0, 1),
           c(0, 0, 0, 0, 0, 0, 1, 1, 1, 1),
           c(0, 0, 0, 0, 0, 0, 0, 0, 1, 0),
           c(1, 1, 0, 0, 1, 0, 0, 1, 1, 1),
           c(1, 0, 1, 1, 1, 0, 1, 0, 0, 0),
           c(0, 1, 0, 0, 1, 1, 1, 0, 0, 1),
           c(0, 1, 0, 1, 1, 0, 1, 0, 1, 0))

n <- max(dim(B))

blk <- c("s" = n)
```

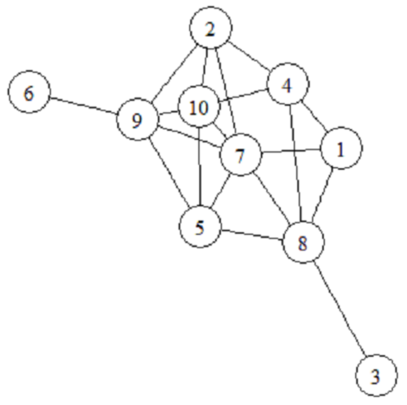
With the objective function in the form $\langle \mathbf{C}, \mathbf{X} \rangle$, we define the input C as

```
 one <- matrix(1, nrow = n, ncol = 1)
 C <- -(diag(c(B %*% one)) - B) / 4
```

where, again, **B** is the adjacency matrix for a graph on which we would like to find the maximum cut, such as the one in Figure 1.

The matrix At is constructed using the upper triangular portion of the $\mathbf{A}_k$ matrices. To do this in R, the function svec is made available in **sdpt3r**.

```
  A <- list()
  for(k in 1:n){
    A[[k]] <- Matrix(0,n,n)
    A[[k]][k,k] <- 1
  }

 At <- svec(blk[1],A,1)
```



$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 1: *A graph object and associated adjacency matrix for which we would like to find the maximum cut.*

Having each of the diagonal elements of **X** constrained to be 1, b is a $n \times 1$ matrix of ones

```
b <- matrix(1, nrow = n, ncol = 1)
```

With all the input variables now defined, we can now call sqlp to solve the Max-Cut problem

```
sqlp(blk, At, list(C), b)
```

A numerical example and the maxcut function

The built-in function maxcut takes as input a (weighted) adjacency matrix B and returns the optimal solution directly. If we wish to find to the maximum cut of the graph in Figure 1, given the adjacency matrix **B** we can compute using maxcut as

```
out <- maxcut(B)
out

$pobj

[1] -14.67622

$X

      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]  [,10]
V1   1.000  0.987 -0.136 -0.858  0.480  0.857 -0.879  0.136 -0.857  0.597
V2   0.987  1.000  0.026 -0.763  0.616  0.929 -0.791 -0.026 -0.929  0.459
V3  -0.136  0.026  1.000  0.626  0.804  0.394  0.592 -1.000 -0.394 -0.876
V4  -0.858 -0.763  0.626  1.000  0.039 -0.469  0.999 -0.626  0.470 -0.925
V5   0.480  0.616  0.804  0.039  1.000  0.864 -0.004 -0.804 -0.864 -0.417
V6   0.857  0.929  0.394 -0.469  0.864  1.000 -0.508 -0.394 -1.000  0.098
V7  -0.879 -0.791  0.592  0.999 -0.004 -0.508  1.000 -0.592  0.508 -0.907
V8   0.136 -0.026 -1.000 -0.626 -0.804 -0.394 -0.592  1.000  0.394  0.876
V9  -0.857 -0.929 -0.394  0.470 -0.864 -1.000  0.508  0.394  1.000 -0.098
V10  0.597  0.459 -0.876 -0.925 -0.417  0.098 -0.907  0.876 -0.098  1.000
```

Note that the value of the primary objective function is negative as we have defined $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$ since we require the primal formulation to be a minimization problem. The original formulation given in Goemans and Williamson (1995) frames the Max-Cut problem as a maximization problem with $\mathbf{C} = (diag(\mathbf{B1}) - \mathbf{B})/4$. Therefore, the approximate value of the maximum cut for the graph in Figure 1 is 14.68 (recall we are solving a relaxation).

As an interesting aside, we can show that the matrix **X** is actually a correlation matrix by considering its eigenvalues - we can see it clearly is symmetric, with unit diagonal and all elements in [-1,1].

```
eigen(out$X[[1]])

$values
```

```
[1] 5.59e+00 4.41e+00 2.07e-07 1.08e-07 4.92e-08 3.62e-08 3.22e-08
[8] 1.90e-08 1.66e-08 9.38e-09
```

The fact that **X** is indeed a correlation matrix comes as no surprise. Goemans and Williamson (1995) show that the set of feasible solutions for the Max-Cut problem is in fact the set of correlation matrices. So while we may not be interested in **X** as an output for solving the Max-Cut problem, it is nonetheless interesting to see that it is in fact in the set of feasible solutions.

Nearest correlation matrix

Recall that the nearest correlation matrix is found as the solution to

$$\begin{array}{ll} \underset{\mathbf{e}^*,\,\mathbf{X}}{\text{minimize}} & e_0 \\ \text{subject to} & \\ & \begin{array}{rcl} svec(\mathbf{R}) - svec(\mathbf{X}) & = & [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\ \mathbf{e}^* \\ diag(\mathbf{X}) & = & \mathbf{1} \\ \mathbf{X} & \in & \mathcal{S}^n \\ \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1} \end{array} \end{array}$$

In Section 52.2.1 we wrote this as the following SQLP

$$\begin{array}{ll} \underset{\mathbf{e}^*,\,\mathbf{X}}{\text{minimize}} & \langle \mathbf{C}, \mathbf{e}^* \rangle \\ \text{subject to} & \\ & \begin{array}{rcl} (\mathbf{A}^s)^{\mathsf{T}} svec(\mathbf{X}) + (\mathbf{A}^q)^{\mathsf{T}} \mathbf{e}^* & = & \mathbf{b} \\ \mathbf{X} & \in & \mathcal{S}^n \\ \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1} \end{array} \end{array}$$

for $\mathbf{C} = [1, \mathbf{0}_{n(n+1)/2}]$, and

$$\begin{array}{rcl} \mathbf{A}^s & = & [\mathbf{A}^s_1,\ \mathbf{A}^s_2] \\ \mathbf{A}^q & = & [\mathbf{A}^q_1,\ \mathbf{A}^q_2] \end{array} \qquad \begin{array}{rcl} \mathbf{b} & = & [\mathbf{b}_1;\ \mathbf{b}_2] \end{array}$$

where

$$\begin{array}{rcl} \mathbf{A}^s_1 & = & \mathbf{I}_{n_2} \\ \mathbf{A}^q_1 & = & [\mathbf{0}, \mathbf{I}_{n_2}] \\ & & \\ \mathbf{A}^s_2 & = & [svec(\mathbf{A}_1), \ldots, svec(\mathbf{A}_n)] \\ \mathbf{A}^q_2 & = & \mathbf{0}_{n_2} \end{array} \qquad \begin{array}{rcl} \mathbf{b}_1 & = & svec(\mathbf{R}) \\ \mathbf{b}_2 & = & \mathbf{1}^{\mathsf{T}} \end{array}$$

and $\mathbf{A}_1, \ldots, \mathbf{A}_n$ are given by

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

To be solved using `sqlp`, we first define `blk`. There are two optimization variables in the formulation of the nearest correlation matrix - **X** is an $n \times n$ matrix constrained to be in a semidefinite cone, and **y** is an $n(n+1)/2 + 1$ length vector constrained to be in a quadratic cone, so

```
data(Hnearcorr)

X = Hnearcorr
n = max(dim(X))
n2 = n * (n + 1) / 2

blk <- c("s" = n, "q" = n2+1)
```

Note that **X** does not appear in the objective function, so the `C` entry corresponding to the block variable **X** is an $n \times n$ matrix of zeros, which defines `C` as

```
C1 <- matrix(0, nrow = n, ncol = n)
C2 <- rbind(1, matrix(0, nrow = n2, ncol = 1))
C <- list(C1,C2)
```

Next comes the constraint matrix for **X**

```
Aks <- matrix(list(), nrow = 1, ncol = n)
for(k in 1:n){
  Aks[[k]] <- matrix(0, nrow = n, ncol = n)
  diag(Aks[[k]])[k] <- 1
}

A1s <- svec(blk[1], Aks)[[1]]
A2s <- diag(1, nrow = n2, ncol = n2)

At1 <- cbind(A1s,A2s)
```

then the constraint matrix for $\mathbf{e}^*$.

```
A1q <- matrix(0, nrow = n, ncol = n2 + 1)

A2q1 <- matrix(0, nrow = n2, ncol = 1)
A2q2 <- diag(1, nrow = n2, ncol = n2)
A2q <- cbind(A2q1, A2q2)

At2 <- rbind(A1q, A2q)
```

and the right hand side vector `b`

```
b <- rbind(matrix(1, n, 1),svec(blk[1], X))
```

The nearest correlation matrix problem is now solved by

```
sqlp(blk, list(At1,At2), C, b)
```

A numerical example and the nearcorr function

To demonstrate the nearest correlation matrix problem, we will modify an existing correlation matrix by exploring the effect of changing the sign of just one of the pairwise correlations. In the context of stock correlations, we make use of tools available in the R package **quantmod** (Ryan and Ulrich, 2017) to access stock data from five tech firms (Microsoft, Apple, Amazon, Alphabet/Google, and IBM) beginning in 2007.

```
library("quantmod")

getSymbols(c("MSFT", "AAPL", "AMZN", "GOOGL", "IBM"))
stock.close <- as.xts(merge(MSFT, AAPL, AMZN,
   GOOGL, IBM))[, c(4, 10, 16, 22, 28)]
```

The correlation matrix for these five stocks is

```
stock.corr <- cor(stock.close)
stock.corr
```

```
           MSFT.Close AAPL.Close AMZN.Close GOOGL.Close IBM.Close
MSFT.Close  1.0000000 -0.2990463  0.9301085   0.5480033 0.2825698
AAPL.Close -0.2990463  1.0000000 -0.1514348   0.3908624 0.6887127
AMZN.Close  0.9301085 -0.1514348  1.0000000   0.6228299 0.3870390
GOOGL.Close 0.5480033  0.3908624  0.6228299   1.0000000 0.5885146
IBM.Close   0.2825698  0.6887127  0.3870390   0.5885146 1.0000000
```

Next, consider the effect of having a positive correlation between Microsoft and Apple

```
stock.corr[1, 2] <- -1 * stock.corr[1, 2]
stock.corr[2, 1] <- stock.corr[1, 2]
stock.corr
```

```
           MSFT.Close AAPL.Close AMZN.Close GOOGL.Close IBM.Close
MSFT.Close  1.0000000  0.2990463  0.9301085   0.5480033 0.2825698
AAPL.Close  0.2990463  1.0000000 -0.1514348   0.3908624 0.6887127
```

```
AMZN.Close   0.9301085 -0.1514348  1.0000000   0.6228299 0.3870390
GOOGL.Close  0.5480033  0.3908624  0.6228299   1.0000000 0.5885146
IBM.Close    0.2825698  0.6887127  0.3870390   0.5885146 1.0000000
```

Unfortunately, this correlation matrix is not positive semidefinite

```
 eigen(stock.corr)$values

[1]  2.8850790  1.4306393  0.4902211  0.3294150 -0.1353544
```

Given the approximate correlation matrix `stock.corr`, the built-in function `nearcorr` solves the nearest correlation matrix problem using `sqlp`

```
 out <- nearcorr(stock.corr)
```

Since this is a minimization problem, and thus a primal formulation of the SQLP, the output `X` from `sqlp` will provide the optimal solution to the problem - that is, `X` will be the nearest correlation matrix to `stock.corr`.

```
out$X

          [,1]        [,2]        [,3]      [,4]      [,5]
[1,] 1.0000000  0.25388359  0.86150833 0.5600734 0.3126420
[2,] 0.2538836  1.00000000 -0.09611382 0.3808981 0.6643566
[3,] 0.8615083 -0.09611382  1.00000000 0.6115212 0.3480430
[4,] 0.5600734  0.38089811  0.61152116 1.0000000 0.5935021
[5,] 0.3126420  0.66435657  0.34804303 0.5935021 1.0000000
```

The matrix above is symmetric with unit diagonal and all entries in $[-1,1]$. By checking the eigenvalues,

```
 eigen(out$X)

 $values

 [1] 2.846016e+00 1.384062e+00 4.570408e-01 3.128807e-01 9.680507e-11
```

we can see that `X` is indeed a correlation matrix.

D-optimal experimental design

Recall from Section 52.2.2 that the D-Optimal experimental design problem was stated as the following dual SQLP

$$\begin{array}{ll} \underset{\mathbf{Z},\,\mathbf{z}^l,\,\lambda}{\text{maximize}} & \log\det(\mathbf{Z}) \\ \text{subject to} & \\ & \begin{array}{rcll} -\sum_{i=1}^{p}\lambda_i(\mathbf{u}_i\mathbf{u}_i^{\mathsf{T}})+\mathbf{Z} & = & 0, & \mathbf{Z}\in\mathcal{S}^n \\ -\boldsymbol{\lambda}+\mathbf{z}^l & = & 0, & \mathbf{z}^l\in\mathcal{R}^p_+ \\ \mathbf{1}^T\boldsymbol{\lambda} & = & 1, & \boldsymbol{\lambda}\in\mathcal{R}^p \end{array} \end{array}$$

which we wrote as

$$\begin{array}{ll} \underset{\mathbf{Z},\,\mathbf{z}^l,\,\lambda}{\text{maximize}} & \log\det(\mathbf{Z}) \\ \text{subject to} & \\ & \begin{array}{rcrcll} (\mathbf{A}^s)^{\mathsf{T}}\boldsymbol{\lambda} & + & \mathbf{Z} & = & \mathbf{C}^s, & \mathbf{Z}\in\mathcal{S}^n \\ (\mathbf{A}^l)^{\mathsf{T}}\boldsymbol{\lambda} & + & \mathbf{z}^l & = & \mathbf{C}^q, & \mathbf{z}^l\in\mathcal{R}^p_+ \\ (\mathbf{A}^u)^{\mathsf{T}}\boldsymbol{\lambda} & & & = & \mathbf{C}^u, & \boldsymbol{\lambda}\in\mathcal{R}^p \end{array} \end{array}$$

where $\mathbf{b}=\mathbf{0}$, and

$$\begin{array}{llcll} \mathbf{A}^s & = -[svec(\mathbf{A}_1),\ldots,svec(\mathbf{A}_p)] & \qquad & \mathbf{C}^s & = \mathbf{0}_{n\times n} \\ \mathbf{A}^l & = -\mathbf{I}_p & & \mathbf{C}^l & = \mathbf{0}_{p\times 1} \\ \mathbf{A}^u & = \mathbf{1}^{\mathsf{T}} & & \mathbf{C}^u & = 1 \end{array}$$

Here, $\mathbf{A}_1,\ldots,\mathbf{A}_p$ are given by

$$\mathbf{A}_i = \mathbf{u}_i \mathbf{u}_i^\mathsf{T}, \quad i = 1, \ldots, p$$

To convert this to a form usable by **sdpt3r**, we first declare the three blocks in `blk`. For a matrix The first block is semidefinite containing the matrix $\mathbf{Z}$, the second a linear block containing $\mathbf{z}^l$, and the third an unrestricted block containing λ

```
data(DoptDesign)
V = DoptDesign
n = nrow(V)
p = ncol(V)

blk = c("s" = n, "l" = p, "u" = 1)
```

Next, by noting the variable λ does not appear in the objective function, we specify `b` as a vector of zeros

```
b <- matrix(0, nrow = p, ncol = 1)
```

Next, looking at the right-hand side of the constraints, we define the matrices `C`

```
C1 <- matrix(0, nrow = n, ncol = n)
C2 <- matrix(0, nrow = p, ncol = 1)
C3 <- 1

C = list(C1,C2,C3)
```

Finally, we construct `At` for each variable

```
A <- matrix(list(), nrow = p, ncol = 1)

for(k in 1:p){
  A[[k]] <- -V[,k] %*% t(V[,k])
}

At1 <- svec(blk[1], A)[[1]]
At2 <- diag(-1, nrow = p, ncol = p)
At3 <- matrix(1, nrow = 1, ncol = p)

At = list(At1,At2,At3)
```

The final hurdle necessary to address in this problem is the existence of the log-barrier. Recall that it is assumed that v^s, v^q, and v^l in Equation 4 are all zero in `control`. In this case, we can see that is not true, as we have a log term containing $\mathbf{Z}$ in the objective function, meaning v^s is equal to one. To pass this to `sqlp`, we define the `control$parbarrier` variable as

```
control <- list(parbarrier = matrix(list(),3,1))
control$parbarrier[[1]] <- 1
control$parbarrier[[2]] <- 0
control$parbarrier[[3]] <- 0
```

The D-Optimal experimental design problem can now be solved using `sqlp`

```
sqlp(blk, At, C, b, control)
```

A numerical example and the doptimal function

To demonstrate the output generated from a D-optimal experimental design problem, we consider a simple 3×25 matrix containing the known test vectors $\mathbf{u}_1, ..., \mathbf{u}_{25}$ (the data is available in the `sqlp` package). To solve the problem using`sqlp`, we use the function `doptimal`, which takes as input an $n \times p$ matrix $\mathbf{U}$ containing the known test vectors, and returns the optimal solution. The output we are interested in is `y`, corresponding to λ in our formulation, the percentage of each $\mathbf{u}_i$ necessary to achieve maximum information in the experiment.

```
data("DoptDesign")

out <- doptimal(DoptDesign)
```

```
out$y
       [,1]
 [1,] 0.000
 [2,] 0.000
 [3,] 0.000
 [4,] 0.000
 [5,] 0.000
 [6,] 0.000
 [7,] 0.154
 [8,] 0.000
 [9,] 0.000
[10,] 0.000
[11,] 0.000
[12,] 0.000
[13,] 0.319
[14,] 0.000
[15,] 0.000
[16,] 0.240
[17,] 0.000
[18,] 0.000
[19,] 0.000
[20,] 0.000
[21,] 0.000
[22,] 0.000
[23,] 0.287
[24,] 0.000
[25,] 0.000
```

The information matrix $\mathbf{A}^\top\mathbf{A}$ is a linear combination of the test vectors $\mathbf{u}_i$, weighted by the optimal vector y above.

Additional problems

The **sdpt3r** package considerably broadens the set of optimization problems that can be solved in R. In addition to those problems presented in detail in Section 52.3, there are a large number of well known problems that can also be formulated as an SQLP.

Each problem presented will be described briefly, with appropriate references for the interested reader, and presented mathematically in its classical form, not as an SQLP as in Equation 3 or 4. Accompanying each problem will be an R helper function, which will solve the corresponding problem using `sqlp`. Each helper function in **sdpt3r** (including those for the max-cut, D-optimal experimental design, and nearest correlation matrix) is an R implementation of the helper functions that are available to the user in the Matlab **SDPT3** package (Toh et al., 1999).

Minimum volume ellipsoids

The problem of finding the ellipsoid of minimum volume containing a set of points $\mathbf{v}_1, ..., \mathbf{v}_n$ is stated as the following optimization problem (Vandenberghe et al., 1998)

$$\begin{array}{ll} \underset{\mathbf{B},\,\mathbf{d}}{\text{maximize}} & log\; det(\mathbf{B}) \\ \text{subject to} & \\ & ||\mathbf{Bx}+\mathbf{d}|| \quad \leq \quad 1, \quad \forall\,]vex \;\in\; [\mathbf{v}_1, ..., \mathbf{v}_n] \end{array}$$

The function `minelips` takes as input an $n \times p$ matrix $\mathbf{V}$ containing the points around which we would like to find the minimum volume ellipsoid, and returns the optimal solution using `sqlp`.

```
data(Vminelips)
out <- minelips(Vminelips)
```

Distance weighted discrimination

Given two sets of points in a matrix $\mathbf{X} \in \mathcal{R}^n$ with associated class variables [-1,1] in $\mathbf{Y} = diag(\mathbf{y})$, distance weighted discrimination (Marron et al., 2007) seeks to classify the points into two distinct subsets by finding a hyperplane between the two sets of points. Mathematically, the distance weighted discrimination problem seeks a hyperplane defined by a normal vector, $\boldsymbol{\omega}$, and position, β, such that each element in the residual vector $\bar{\mathbf{r}} = \mathbf{YX}^{\mathsf{T}}\omega + \beta\mathbf{y}$ is positive and large. Since the class labels are either 1 or -1, having the residuals be positive is equivalent to having the points on the proper side of the hyperplane.

Of course, it may be impossible to have a perfect separation of points using a linear hyperplane, so an error term ξ is introduced. Thus, the perturbed residuals are defined to be

$$\mathbf{r} = \mathbf{YX}^{\mathsf{T}}\omega + \beta\mathbf{y} + \xi$$

Distance weighted discrimination (Marron et al., 2007) solves the following optimization problem to find the optimal hyperplane.

$$\begin{array}{ll} \underset{\mathbf{r},\,\omega,\,\beta,\,\xi}{\text{minimize}} & \sum_{i=1}^{n}(1/r_i) + C\mathbf{1}^{\mathsf{T}}\xi \\ \text{subject to} & \\ & \begin{array}{rcl} \mathbf{r} & = & \mathbf{YX}^{\mathsf{T}}\omega + \beta\mathbf{y} + \xi \\ \omega^{\mathsf{T}}\omega & \leq & 1 \\ \mathbf{r} & \geq & \mathbf{0} \\ \xi & \geq & \mathbf{0} \end{array} \end{array}$$

where $C > 0$ is a penalty parameter to be chosen.

The function `dwd` takes as input two $n \times p$ matrices `X1` and `X2` containing the points to be separated, as well as a penalty term `C` ≥ 0 penalizing the movement of a point on the wrong side of the hyperplane to the proper side, and returns the optimal solution using `sqlp`.

```
data(Andwd)
data(Apdwd)
C <- 0.5

out <- dwd(Apdwd,Andwd,penalty)
```

Max-kCut

Similar to the Max-Cut problem, the Max-kCut problem asks, given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and an integer k, does a cut exist of at least size k. For a given (weighted) adjacency matrix $\mathbf{B}$ and integer k, the Max-kCut problem is formulated as the following primal problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C},\, \mathbf{X} \rangle \\ \text{subject to} & \\ & \begin{array}{rcll} diag(\mathbf{X}) & = & \mathbf{1} & \\ X_{ij} & \geq & 1/(k-1) & \forall\, i \neq j \\ \mathbf{X} & \in & \mathcal{S}_n & \end{array} \end{array}$$

Here, $\mathbf{C} = -(1 - 1/k)/2 * (diag(\mathbf{B1}) - \mathbf{B})$. The Max-kCut problem is slightly more complex than the Max-Cut problem due to the inequality constraint. In order to turn this into a standard SQLP, we must replace the inequality constraints with equality constraints, which we do by introducing a slack variable $\mathbf{x}^l$, allowing the problem to be restated as

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C},\, \mathbf{X} \rangle \\ \text{subject to} & \\ & \begin{array}{rcll} diag(\mathbf{X}) & = & \mathbf{1} & \\ X_{ij} - x^l & = & 1/(k-1) & \forall\, i \neq j \\ \mathbf{X} & \in & \mathcal{S}^n & \\ \mathbf{x}^l & \in & \mathcal{L}^{n(n+1)/2} & \end{array} \end{array}$$

The function `maxkcut` takes as input an adjacency matrix `B` and an integer `k`, and returns the optimal solution using `sqlp`.

```
data(Bmaxkcut)
k = 2

out <- maxkcut(Bmaxkcut,k)
```

Graph partitioning problem

The graph partitioning problem can be formulated as the following primal optimization problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{CX}) \\ \text{subject to} & \\ & \begin{array}{rcl} tr(\mathbf{1}\mathbf{1}^{\mathsf{T}}\mathbf{X}) & = & \alpha \\ diag(\mathbf{X}) & = & \mathbf{1} \end{array} \end{array}$$

Here, $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})$, for an adjacency matrix $\mathbf{B}$, and α is any real number.

The function gpp, takes as input a weighted adjacency matrix B and a real number alpha and returns the optimal solution using sqlp.

```
data(Bgpp)
alpha <- nrow(Bgpp)

out <- gpp(Bgpp, alpha)
```

The Lovasz number

The Lovasz Number of a graph $\mathbf{G}$, denoted $\vartheta(\mathbf{G})$, is the upper bound on the Shannon capacity of the graph. For an adjacency matrix $\mathbf{B} = [B_{ij}]$ the problem of finding the Lovasz number is given by the following primal SQLP problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{CX}) \\ \text{subject to} & \\ & \begin{array}{rcll} tr(\mathbf{X}) & = & 1 & \\ X_{ij} & = & 0 & \text{if } B_{ij} = 1 \\ \mathbf{X} & \in & \mathcal{S}^n & \end{array} \end{array}$$

The function lovasz takes as input an adjacency matrix $\mathbf{B}$, and returns the optimal solution using sqlp.

```
data(Glovasz)

out <- lovasz(Glovasz)
```

Toeplitz approximation

Given a symmetric matrix $\mathbf{F}$, the Toeplitz approximation problem seeks to find the nearest symmetric positive definite Toeplitz matrix. In general, a Toeplitz matrix is one with constant descending diagonals, i.e.,

$$\mathbf{T} = \begin{bmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{bmatrix}$$

is a general Toeplitz matrix. The problem is formulated as the following optimization problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{maximize}} & -y_{n+1} \\ \text{subject to} & \\ & \begin{array}{rcl} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & -\beta \end{bmatrix} + \sum_{k=1}^{n} y_k \begin{bmatrix} \mathbf{0} & \gamma_k \mathbf{e}_k \\ \gamma_k \mathbf{e}_k^T & -2q_k \end{bmatrix} + y_{n+1}\mathbf{B} & \geq & \mathbf{0} \\ [y_1, ..., y_n]^{\mathsf{T}} + y_{n+1}\mathbf{B} & \geq & \mathbf{0} \end{array} \end{array}$$

where $\mathbf{B}$ is an $(n+1) \times (n+1)$ matrix of zeros, and $\mathbf{B}_{(n+1)(n+1)} = 1$, $q_1 = -tr(\mathbf{F})$, q_k = sum of k^{th} diagonal upper and lower triangular matrix, $\gamma_1 = \sqrt{n}$, $\gamma_k = \sqrt{2 * (n-k+1)}$, $k = 2, ..., n$, and $\beta = ||\mathbf{F}||_F^2$.

The function toep takes as input a symmetric matrix F for which we would like to find the nearest Toeplitz matrix, and returns the optimal solution using sqlp.

```
data(Ftoep)

out <- toep(Ftoep)
```

The educational testing problem

The educational testing problem arises in measuring the reliability of a student's total score in an examination consisting of a number of sub-tests (Fletcher, 1981). In terms of formulation as an optimization problem, the problem is to determine how much can be subtracted from the diagonal of a given symmetric positive definite matrix **S** such that the resulting matrix remains positive semidefinite (Chu and Wright, 1995).

The Educational Testing Problem (ETP) is formulated as the following dual problem

$$\begin{array}{ll} \underset{\mathbf{d}}{\text{maximize}} & \mathbf{1}^\mathsf{T}\mathbf{d} \\ \text{subject to} & \\ & \begin{array}{rcl} \mathbf{A} - diag(\mathbf{d}) & \succeq & \mathbf{0} \\ \mathbf{d} & \geq & \mathbf{0} \end{array} \end{array}$$

where $\mathbf{d} = [d_1, d_2, ..., d_n]$ is a vector of size n and $diag(\mathbf{d})$ denotes the corresponding $n \times n$ diagonal matrix. In the second constraint, having each element in **d** be greater than or equal to 0 is equivalent to having $diag(\mathbf{d}) \succeq 0$.

The corresponding primal problem is

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{AX}) \\ \text{subject to} & \\ & \begin{array}{rcl} diag(\mathbf{X}) & \geq & \mathbf{1} \\ \mathbf{X} & \succeq & 0 \end{array} \end{array}$$

The function `etp` takes as input an $n \times n$ positive definite matrix `A`, and returns the optimal solution using `sqlp`.

```
data(Betp)

out <- etp(Betp)
```

Logarithmic Chebyshev approximation

For a $p \times n$ $(p > n)$ matrix **B** and $p \times 1$ vector **f**, the Logarithmic Chebyshev Approximation problem is stated as the following optimization problem (Vandenberghe et al., 1998)

$$\begin{array}{ll} \underset{\mathbf{x},\, t}{\text{minimize}} & t \\ \text{subject to} & \\ & 1/t \;\leq\; (\mathbf{x}^\mathsf{T}\mathbf{B}_{i\cdot})/\mathbf{f}_i \;\leq\; t, \quad i = 1, ..., p \end{array}$$

where $\mathbf{B}_{i\cdot}$ denotes the i^{th} row of the matrix **B**. Note that we require each element of $\mathbf{B}_{\cdot j}/\mathbf{f}$ to be greater than or equal to 0 for all j.

The function `logcheby` takes as input a matrix `B` and vector `f`, and returns the optimal solution to the Logarithmic Chebyshev Approximation problem using `sqlp`.

```
data(Blogcheby)
data(flogcheby)

out <- logcheby(Blogcheby, flogcheby)
```

Linear matrix inequality problems

We consider three distinct linear matrix inequality problems, all written in the form of a dual optimization problem. The first linear matrix inequality problem we will consider is defined by the following optimization equation for some $n \times p$ matrix **B** known in advance

$$\begin{array}{lrcl}
\underset{\eta,\,\mathbf{Y}}{\text{maximize}} & -\eta & & \\
\text{subject to} & & & \\
& \mathbf{BY}+\mathbf{YB}^{\mathsf{T}} & \preceq & 0 \\
& -\mathbf{Y} & \preceq & -\mathbf{I} \\
& \mathbf{Y}-\eta\mathbf{I} & \preceq & 0 \\
& Y_{11} & = & 1, \quad \mathbf{Y}\in\mathcal{S}^n
\end{array}$$

The function lmi1 takes as input a matrix **B**, and returns the optimal solution using sqlp.

```
B <- matrix(c(-1,5,1,0,-2,1,0,0,-1), nrow=3)

out <- lmi1(B)
```

The second linear matrix inequality problem is

$$\begin{array}{lrcl}
\underset{\mathbf{P},\,\mathbf{d}}{\text{maximize}} & -tr(\mathbf{P}) & & \\
\text{subject to} & & & \\
& \mathbf{A}_1\mathbf{P}+\mathbf{P}\mathbf{A}_1{}^{\mathsf{T}}+\mathbf{B}*diag(\mathbf{d})*\mathbf{B}^{\mathsf{T}} & \preceq & 0 \\
& \mathbf{A}_2\mathbf{P}+\mathbf{P}\mathbf{A}_2{}^{\mathsf{T}}+\mathbf{B}*diag(\mathbf{d})*\mathbf{B}^{\mathsf{T}} & \preceq & 0 \\
& -\mathbf{d} & \preceq & 0 \\
& \sum_i^p d_i & = & 1
\end{array}$$

Here, the matrices **B**, $\mathbf{A}_1$, and $\mathbf{A}_2$ are known in advance.

The function lmi2 takes the matrices A1, A2, and B as input, and returns the optimal solution using sqlp.

```
A1 <- matrix(c(-1,0,1,0,-2,1,0,0,-1),3,3)
A2 <- A1 + 0.1*t(A1)
B  <- matrix(c(1,3,5,2,4,6),3,2)

out <- lmi2(A1,A2,B)
```

The final linear matrix inequality problem originates from a problem in control theory (Boyd et al., 1994) and requires three matrices be known in advance, **A**, **B**, and **G**

$$\begin{array}{ll}
\underset{\eta,\,\mathbf{P}}{\text{maximize}} & \eta \\
\text{subject to} & \\
& \begin{bmatrix} \mathbf{AP}+\mathbf{PA}^{\mathsf{T}} & \mathbf{0} \\ \mathbf{BP} & \mathbf{0} \end{bmatrix} + \eta \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \preceq \begin{bmatrix} -\mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}
\end{array}$$

The function lmi3 takes as input the matrices A, B, and G, and returns the optimal solution using sqlp.

```
A <- matrix(c(-1,0,1,0,-2,1,0,0,-1),3,3)
B <- matrix(c(1,2,3,4,5,6), 2, 3)
G <- matrix(1,3,3)

out <- lmi3(A,B,G)
```

Summary

In Section 52.2, we introduced the problem of conic linear optimization. Using the Max-Cut, Nearest Correlation Matrix, and D-Optimal Experimental Design problems as examples, we demonstrated the increasing generality of the problem, culminating in a general form of the conic linear optimization problem, known as the semidefinite quadratic linear program, in Section 52.2.2.

In Section 52.3, we introduced the R package **sdpt3r**, and the main function call available in the package, sqlp. The specifics of the necessary input variables, the optional input variables, and the output variables provided by sqlp were presented. Using the examples from Section 52.2, we showed how a problem written as a semidefinite quadratic linear program could be solved in R using **sdpt3r**.

Finally, in Section 52.4, we presented a number of additional problems that can be solved using the **sdpt3r** package, and presented the helper functions available so these problems could be easily solved using sqlp.

The **sdpt3r** package broadens the range of problems that can be solved using R. Here, we discussed a number of problems that can be solved using **sdpt3r**, including problems in the statistical sciences, graph theory, classification, control theory, and general matrix theory. The `sqlp` function in **sdpt3r** is in fact even more general, and users may apply it to any other conic linear optimization problem that can be written in the form of Equation 3 or 4 by specifying the input variables `blk`, `At`, `C`, and `b` for their particular problem.

control

`vers`	specifies the search direction
	0, HKM if semidefinite blocks present, NT otherwise (default)
	1, HKM direction
	2, NT direction
`predcorr`	TRUE, use Mehrotra prediction-correction (default)
	FALSE, otherwise
`gam`	step-length (default 0)
`expon`	exponent used to decrease sigma (default 1)
`gaptol`	tolerance for duality gap as a fraction of the objective function (default $1e-8$)
`inftol`	tolerance for stopping due to infeasibility (default 1e-8)
`steptol`	tolerance for stopping due to small steps (default 1e-6)
`maxit`	maximum number of iterations (default 100)
`stoplevel`	0, continue until successful completion, maximum iteration, or numerical failure
	1, automatically detect termination, restart if small steps is cause (default)
	2, automatically detect termination
`scale_data`	TRUE, scale data prior to solving
	FALSE, otherwise (default)
`rmdepconstr`	TRUE, remove nearly dependent constraints
	FALSE, otherwise (default)
`parbarrier`	declare the existence of a log barrier term
	default value is 0 (i.e., no log barrier)

Bibliography

S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994. [p391]

H. C. Bravo. *Rcsdp: R Interface to the CSDP Semidefinite Programming Library*, 2016. URL http://CRAN.R-project.org/package=Rcsdp. R package version 0.1.55. [p371]

M. T. Chu and J. W. Wright. The educational testing problem revisited. *IMA journal of numerical analysis*, 15(1):141–160, 1995. [p371, 390]

R. Fletcher. A nonlinear programming problem in statistics (educational testing). *SIAM Journal on Scientific and Statistical Computing*, 2(3):257–267, 1981. [p390]

R. M. Freund. Introduction to semidefinite programming (sdp). *Massachusetts Institute of Technology*, 2004. [p380]

H. Friberg. Rmosek: The r-to-mosek optimization interface. *R package version*, 1(3), 2012. [p371]

A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R Package for Disciplined Convex Optimization. *arXiv*, 2017. URL https://arxiv.org/abs/1711.07582v1. [p371]

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995. [p372, 382, 383]

C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996. [p376]

N. J. Higham. Computing the nearest correlation matrix-a problem from finance. *IMA Journal of Numerical Analysis*, 22(3):329–343, 2002. [p371, 373]

F. John. Extremum problems with inequalities as subsidiary conditions. In *Traces and Emergence of Nonlinear Programming*, pages 197–215. Springer-Verlag, 2014. [p371]

R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer-Verlag, 1972. [p372]

J. S. Marron, M. J. Todd, and J. Ahn. Distance-weighted discrimination. *Journal of the American Statistical Association*, 102(480):1267–1271, 2007. [p371, 387, 388]

Y. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations research*, 22(1):1–42, 1997. [p376]

B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 2.0.2. https://github.com/cvxgrp/scs, 2017. [p371]

C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991. [p372]

B. Pfaff. *Cccp: Cone Constrained Convex Problems*, 2015. URL http://CRAN.R-project.org/package=cccp. R package version 0.2-4. [p371]

J. A. Ryan and J. M. Ulrich. *Quantmod: Quantitative Financial Modelling Framework*, 2017. URL http://CRAN.R-project.org/package=quantmod. R package version 0.4-9. [p384]

K. Smith. On the standard deviations of adjusted and interpolated values of an observed polynomial function and its constants and the guidance they give towards a proper choice of the distribution of observations. *Biometrika*, 12(1-2):1–85, 1918. [p371]

S. Theußl, F. Schwendinger, and K. Hornik. Roi: The r optimization infrastructure package. Research Report Series / Department of Statistics and Mathematics 133, WU Vienna University of Economics and Business, Vienna, 2017. URL http://epub.wu.ac.at/5858/. [p371]

K.-C. Toh, M. J. Todd, and R. H. Tütüncü. Sdpt3 - a matlab software package for semidefinite programming, version 1.3. *Optimization methods and software*, 11(1-4):545–581, 1999. [p371, 373, 376, 387]

R. H. Tütüncü, K.-C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using sdpt3. *Mathematical programming*, 95(2):189–217, 2003. [p372, 376]

L. Vandenberghe, S. Boyd, and S.-P. Wu. Determinant maximization with linear matrix inequality constraints. *SIAM journal on matrix analysis and applications*, 19(2):499–533, 1998. [p371, 376, 387, 390]

Z. Zhu and Y. Ye. *Rdsdp: R Interface to DSDP Semidefinite Programming Library*, 2016. URL http://CRAN.R-project.org/package=Rdsdp. R package version 1.0.4-2. [p371]

Adam Rahman
University of Waterloo
200 University Ave W, Waterloo, Ontario
Canada
a45rahma@uwaterloo.ca

Explanations of Model Predictions with live and breakDown Packages

by Mateusz Staniak and Przemysław Biecek

Abstract Complex models are commonly used in predictive modeling. In this paper we present R packages that can be used for explaining predictions from complex black box models and attributing parts of these predictions to input features. We introduce two new approaches and corresponding packages for such attribution, namely **live** and **breakDown**. We also compare their results with existing implementations of state-of-the-art solutions, namely, **lime** (Pedersen and Benesty, 2018) which implements *Locally Interpretable Model-agnostic Explanations* and **iml** (Molnar et al., 2018) which implements *Shapley values*.

Introduction

Predictive modeling is a very exciting field with a wide variety of applications. Lots of algorithms have been developed in this area. As proven in many Kaggle competitions (Fogg, 2016), winning solutions are often obtained with the use of elastic tools like random forest, gradient boosting or neural networks. Many of them are implemented by R packages found in the CRAN Task View *MachineLearning*.

These algorithms have many strengths but they also share a major weakness, which is a deficiency in interpretability of a model structure. A single random forest, an xgboost model or a neural network may be parameterized with thousands of parameters which make these models hard to understand. Lack of interpretability results in the lack of trust in model predictions. Lack of trust is a major obstacle when one thinks about applications in regulated areas such as personalized medicine or similar fields. An interesting example of a situation in which trust issues are fully justified is presented in Ribeiro et al. (2016). Authors compare two classifiers that were trained to recognize whether a text describes Christianity or Atheism. After explanations were provided, it turned out that the model with superior performance in the test set often based its prediction on irrelevant words, for example, prepositions. To overcome this problem, the interpretability of complex machine learning models has been a subject of much research, devoted partially also to model visualization (Štrumbelj and Kononenko, 2011; Tzeng and Ma, 2005; Zeiler and Fergus, 2014).

The general approach to interpretability is to identify important variables (features) in the model and then learn the expected model response for a single variable. A description of a general framework of permutation-based variable importance rankings may be found in Altmann et al. (2010). An interesting and widely adopted tool for estimation of marginal model response is *Partial Dependency Plot* (Friedman, 2001), that presents the marginal relationship between the variable of interest and a single variable from the model. An effective and very elastic implementation of this method is available in the **pdp** package (Greenwell, 2017). This method has many extensions such as for example *Individual Conditional Expectations* (Goldstein et al., 2015). The ICE method allows for tracing predictions for individual variables and it is very useful for the identification of interactions. On the other hand, ALE plots (Apley, 2016) were proposed as a superior tool for handling strongly correlated predictors by describing the conditional distribution of predicted values. This method can be used to assess both main effects and interactions between predictors. All these methods are focused on the effect of a single variable or small set of variables within the black box model.

A different approach is presented by Ribeiro et al. (2016). While methods such as PDP and ALE plot aim to describe model behavior globally, it is also possible to explain individual predictions as in ICE. We will focus on such methods. The authors propose LIME (*Locally Interpretable Model agnostic Explanations*) as a method for explaining black box predictions by fitting an interpretable model locally around a prediction of interest. This methodology was illustrated with examples from image and text classification areas. Later, it was extended by Puri et al. (2017) to MAGIX methodology (*Model Agnostic Globally Interpretable Explanations*) and modified by the authors of the original article to aLIME (*anchor-LIME*) by Tulio Ribeiro et al. (2016).

So far, two implementations of the method have been found. Python library was developed by the authors of the original article and it is available on GitHub at `https://github.com/marcotcr/lime`. It works for any text or image classifier as well as for tabular data. Regression models can be explained using simple linear regression. The R package **lime** is a port to the original Python package. This package works with tabular and text data and handles all models supported by either **caret** (Kuhn, 2018) or **mlr** (Bischl et al., 2016) package and it can be easily extended to work with other models. An implementation of the *sp-LIME* algorithm proposed in the original article to choose representative observations that would explain the behavior of the model globally is available for Python.

In addition to packages that implement single methods, in R we have also two packages with a consistent collection of explainers: **DALEX** (Biecek, 2018) and **iml**.

In this article, we give a short overview of methods for explaining predictions made by complex models. We also introduce two new methods implemented in R packages **live** and **breakDown**. Examples presented in this paper were recorded with the **archivist** package (Biecek and Kosinski, 2017). They can be accessed and restored with a single R instruction listed in footnotes.

Related work

In this section we present two of the most recognized methods for explanations of a single prediction from a complex black box model (so-called instance-level explanations).

Locally Interpretable Model-agnostic Explanations (LIME)

Ribeiro et al. (2016) proposed **LIME** method for explaining prediction for a single observation. The algorithm is designed to explain predictions of any classifier and it works primarily for image and text data. First, original observation of interest is transformed into simplified input space of binary vectors (for example presence or absence of words). Then a dataset of similar observations is created by sampling features that are present in the representation of the explained instance. The closeness of these observations to the original observations is measured via a specified similarity kernel. This distance is taken into account while the explanation model is fitted to the new dataset. The interpretable model can be penalized to assure that it does not become too complex itself. In mathematical terms, LIME explanation for observation x is a model g which approximates complex model f by solving the following optimization problem

$$g(\cdot) = \arg\min_{h \in G} \left[\mathcal{L}(f, h, \pi_x(z)) + \Omega(h) \right],$$

where z is the interpretable (binary) representation of x, $\pi_x(z)$ is a measure of closeness of z and x (the kernel), $\mathcal{L}$ is a loss function that measures local faithfulness of the explanation and Ω is a model complexity measure, that serves as a regularization term.

Shapley values (SHAP)

In 2017 Lundberg and Lee (2017) introduced a general framework for explaining machine learning models that encompasses LIME among other methods. The method is associated with some specific visualization techniques that present how predictors contribute to the predicted values. In this framework, observations are transformed into the space of binary variables called simplified inputs. Explanation models are restricted to the so-called **additive feature attributions methods**, what means that values predicted by the explanation model are linear combinations of these binary input vectors. Formally, if $z = (z_1, \dots, z_p)$ is a binary vector in simplified inputs space and g is the explanation model, then

$$g(z) = \phi_0 + \sum_{j=1}^{M} \phi_j z_j,$$

where $\phi_j, j = 0, \dots, M$ are weights. These weights measure how each feature contributes to the prediction. Authors prove that in this class of explanation models **Shapley values** provide unique solutions to the problem of finding optimal weights ϕ_j that assure that the model has desirable properties of local accuracy and consistency. For formal treatment and examples, please refer to the original article of Lundberg and Lee (2017). In particular, you will find there a proof that for certain choices of parameters in the LIME method, coefficients of the fitted local linear model are Shapley values. The Python implementation of this method is available at `https://github.com/slundberg/shap`. In R *Shapley values* can be found in few packages. For tree based models they are implemented in **xgboost** package (Chen et al., 2018). An independent model agnostic implementation is available in the **iml** package. A development version of a package dedicated entirely to Shapley values can be found at `https://github.com/redichh/ShapleyR`.

Local Interpretable Visual Explanations (LIVE)

The next two sections introduce two new approaches to explaining model prediction, implemented in R packages **live** and **breakDown**, respectively. Both of these methods describe locally (at an instance level) how features contribute to a model prediction.

Motivation

live is an alternative implementation of LIME for regression problems, which emphasizes the role of model visualization in the understanding of complex models. In comparison with the original LIME, both the method of local exploration and handling of interpretable inputs are changed. A dataset for local exploration is simulated by perturbing the explained instance one feature at a time. The process is described in section 54.3.2. By default, the Gaussian kernel is used to measure distances between simulated observation and the observation of interest, but other kernels can also be used, too. Original variables are used as interpretable inputs, so numerical features are used in the explanation model. Interpretability of the local explanation comes from a tractable relationship between inputs and the predicted response. Variable selection is optional for linear regression when sparsity is required.

One of the main purposes of **live** is to provide tools for model visualization, which is why in this package emphasis is put on models that are easy to visualize. For linear models, waterfall plots can be drawn to present how predictors contribute to the overall model score for a given prediction, while forest plots (Kennedy, 2017) can be drawn to summarize the structure of local linear approximation. Examples clarifying both techniques are given in section 54.5. Other interpretable models that are equipped with generic `plot` function can be visualized, too. In particular, decision trees which can be plotted using **party** package (Strobl et al., 2008) are well suited for this task, as they can help discover interactions. An example is given in section 54.5. The package uses the **mlr** interface to handle machine learning algorithms, hence any classifier or regression method supported by **mlr** can be used as an interpretable model, though in practice simple models will be preferred. The most common choice is a sparse linear model.

Methodology

live package uses a two-step procedure to explain prediction of a selected black box model in the point x. First, an artificial dataset X' is created around point x. Then, the white box model is fitted to the model predictions for points in X'.

The first step is described by the Algorithm 1.

```
1: $p \leftarrow$ number of predictors
2: $m \leftarrow$ number of observations to generate
3: Duplicate the given observation $m$ times
4: for i in $\{1, \ldots, m\}$ do
5:     Draw number $k \in \{1, \ldots, p\}$ uniformly. Replace the value of $k$-th variable in i-th
       duplicate with a random draw from the empirical distribution of this variable in the
       original dataset
6: end for
```

Algorithm 2: Simulating X' - surroundings around the selected x.

In other words, the procedure amounts to iterating over the set of m observations identical to a given instance and changing the value of one random variable at each step. Alternatively, in **live** package new dataset can be sampled from a multivariate normal distribution and using permutations of each column. Details can be found in the manual. Current implementation of this algorithm relies on **data.table** package for performance (Dowle and Srinivasan, 2017). The choice of the number of instances to sample is an open problem both in LIME and LIVE methods.

Model agnostic greedy explanations of model predictions (breakDown)

Motivation

live package approximates the local structure of the black box model around a single point in the feature space. The idea behind the **breakDown** is different. In the case of that package, the main goal is to decompose model predictions into parts that can be attributed to particular variables. It is straightforward for linear (and more generally: additive) models. Below we present a model agnostic approach that works also for nonlinear models.

Let us use the following notation: $x = (x_1, x_2, ..., x_p) \in X \subset \mathcal{R}^p$ is a vector in feature space X. $f : X \to R$ is a scoring function for the model under consideration, that may be used for regression of classification problems. X^{train} is a training dataset with n observations.

For a single observation x^{new} the model prediction is equal to $f(x^{new})$. Our goal is to attribute parts of this score to variables (dimensions) in the X space.

The lm-break: version for additive models

For linear models (and also generalized linear models) the scoring function (e.g. link function) may be expressed as a linear combination of feature vectors.

$$f(x^{new}) = (1, x^{new})(\mu, \beta)^T = \mu + x_1^{new}\beta_1 + \ldots + x_p^{new}\beta_p. \tag{1}$$

In this case it is easy to attribute the impact of feature x_i to prediction $f(x^{new})$. The most straightforward approach would be to use the $x_i^{new}\beta_i$ as the attribution. However, it is easier to interpret variable attributions if they are invariant to scale-location transformations of x_i, such as change of the units or origin. Centering addresses location changes and scaling also changes the scales of β parameters. This is why for linear models the **lm-break** variable attributions are defined as $(x_i^{new} - \bar{x}_i)\beta_i$. The equation 1 may be rewritten as follows:

$$f(x^{new}) = (1, x^{new})(\mu, \beta)^T = baseline + (x_1^{new} - \bar{x}_1)\beta_1 + ... + (x_p^{new} - \bar{x}_p)\beta_p \tag{2}$$

where

$$baseline = \mu + \bar{x}_1\beta_1 + ... + \bar{x}_p\beta_p.$$

Components $(x_i^{new} - \bar{x}_i)\beta_i$ are all expressed in the same units. For `lm` and `glm` models these values are calculated and plotted by the generic `broken()` function from the **breakDown** package.

The ag-break: model agnostic approach

Interpretation of **lm-break** attributions is straightforward but limited only to additive models. In this section, we present an extension for non-additive models. This extension uses additive attributions to explain predictions from non-additive models thus some information about the model structure will be lost. Still, for many models, such attribution may be useful. For additive models the **ag-break** approach gives the same results as **lm-break** approach.

The intuition behind **ag-break** approach is to identify components of x^{new} that cannot be changed without a significant change in the prediction $f(x^{new})$. In order to present this approach in a more formal way, we first need to introduce some definitions.

Definition 54.4.1 (Relaxed model prediction)**.** Let $f^{IndSet}(x^{new})$ denote an expected model prediction for x^{new} relaxed on the set of indexes $IndSet \subset \{1, \ldots, p\}$.

$$f^{IndSet}(x^{new}) = E[f(x)|x_{IndSet} = x_{IndSet}^{new}]. \tag{3}$$

Thus $f^{IndSet}(x^{new})$ is an expected value for model response conditioned on variables from set $IndSet$ in such a a way, that $\forall_{i \in IndSet} x_i = x_i^{new}$.

The intuition behind relaxed prediction is that we are interested in an average model response for observations that are equal to x^{new} for features from $IndSet^C$ set and follow the population distribution for features from $IndSet$ set. Clearly, two extreme cases are

$$f^{\{1,\ldots,p\}}(x^{new}) = f(x^{new}), \tag{4}$$

which is the case of no relaxation, and

$$f^{\varnothing}(x^{new}) = E[f(x)]. \tag{5}$$

which corresponds to full relaxation. We will say that a variable was relaxed when we do not fix its value and we let it follow the population distribution. This will play a crucial part in the algorithm presented in this section.

Since we do not know the joint distribution of x, we will use its estimate instead.

$$\widehat{f^{IndSet}(x^{new})} = \frac{1}{n}\sum_{i=1}^{n} f(x_{-IndSet}^{i}, x_{IndSet}^{new}). \tag{6}$$

We will omit the dashes to simplify the notation.

Definition 54.4.2 (Distance to relaxed model prediction)**.** Let us define the distance between model prediction and relaxed model prediction for a set of indexes *IndSet*.

$$d(x^{new}, IndSet) := |f^{IndSet}(x^{new}) - f(x^{new})|. \tag{7}$$

It is the difference between model prediction for observation x^{new} and observation relaxed on features *indSet*. The smaller the difference, the less important are variables in the *indSet* set.

Definition 54.4.3 (Added feature contribution)**.** For j-th feature we define its contribution relative to a set of indexes *IndSet* (*added contribution*) as

$$\text{contribution}^{IndSet}(j) = f^{IndSet\cup\{j\}}(x^{new}) - f^{IndSet}(x^{new}). \tag{8}$$

It is the change in model prediction for x^{new} after relaxation on j.

The model agnostic feature contribution is based on distances to relaxed model predictions. In this approach, we look for a series of variables that can be relaxed in such a way so as to move model prediction from $f(x^{new})$ to a fully relaxed prediction $E[f(x)]$ (expected value for all model predictions). The order of features in this series is important. But here we use a greedy strategy in which we add features to the *indSet* iteratively (one feature per iteration) and minimize locally the distance to relaxed model prediction.

This approach can be seen as an approximation of Shapley values where feature contribution is linked with the average effect of a feature across all possible relaxations. These approaches are identical for additive models. For non-additive models, the additive attribution is just an approximation in both cases, yet the greedy strategy produces explanations that are easier to interpret. It is worth noting that similar decomposition of predictions and measures of contribution for classifiers have been examined in Robnik-Šikonja and Kononenko (2008).

The greedy search can start from a null set of indexes (then in each step a single feature is being relaxed) or it can start from a full set of relaxed features (then in each step a single feature is removed from the set). The above approaches are called *step-up* and *step-down*, respectively. They are presented in algorithms 3 and 4.

The algorithm 3 presents the procedure that generates a sequence of variables with increasing contributions. This sequence corresponds to variables that can be relaxed in such a way so as to minimize the distance to the original prediction. The resulting sequence of *Contributions* and *Variables* may be plotted with Break Down Plots, see an example in Figure 2. Figure 1 summarizes the idea behind algorithm 3. By relaxing consecutive variables one finds a path between single prediction and average model prediction.

One can also consider an opposite strategy - instead of starting from $IndSet = \{1, \ldots, p\}$ one can start with $IndSet = \varnothing$. That strategy is called *step-up* approach and it is presented in Algorithm 4.

Note that the Break Down method is also available for Python in the `pyBreakDown` library.[1]

```
 1: $p \leftarrow$ number of variables
 2: $IndSet \leftarrow \{1, \ldots, p\}$ set of indexes of all variables
 3: for i in $\{1, \ldots, p\}$ do
 4:    Find new variable that can be relaxed with small loss in relaxed distance to $f(x^{new})$
 5:    for j in $IndSet$ do
 6:       Calculate relaxed distance with $j$ removed
 7:       $dist(j) \leftarrow d(x^{new}, IndSet \setminus \{j\})$
 8:    end for
 9:    Find and remove $j$ that minimizes loss
10:    $j_{min} \leftarrow \arg\min_j dist(j)$
11:    $Contribution^{IndSet}(i) \leftarrow f^{IndSet}(x^{new}) - f^{IndSet\setminus\{j_{min}\}}(x^{new})$
12:    $Variables(i) \leftarrow j_{min}$
13:    $IndSet \leftarrow IndSet \setminus \{j_{min}\}$
14: end for
```

Algorithm 3: Model agnostic break down of model predictions. The *step-down* approach.

[1]https://github.com/MI2DataLab/pyBreakDown/

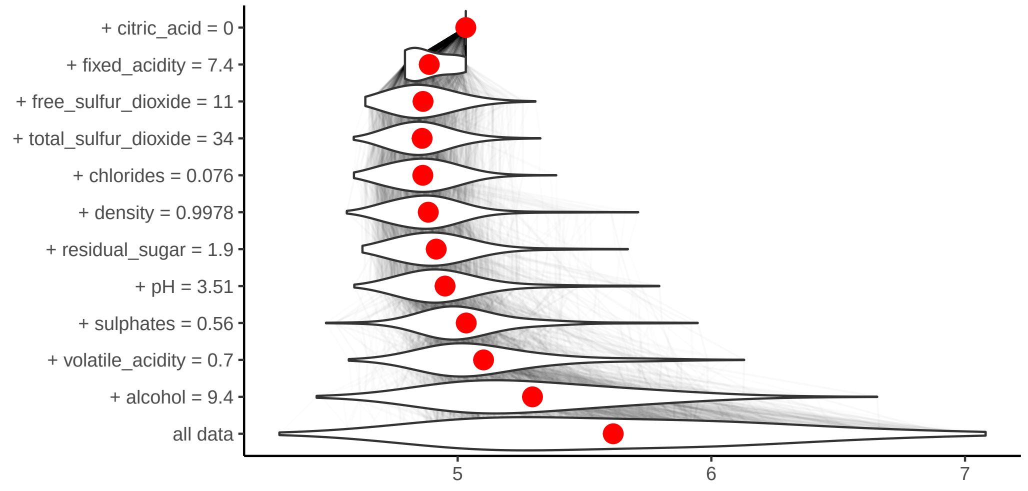


Figure 1: An illustration of algorithm 3. Each row in this plot correspond to a distribution of model scores $f(x)|x_{IndSet} = x^{new}_{IndSet}$ for different sets of $IndSet$ indexes. Initially $IndSet = \{1, \ldots, p\}$ and in each step single variable is removed from this set. Labels on the left-hand side of the plots show which variable is removed in a given step. Red dots stand for conditional average - an estimate of relaxed predictions $f^{IndSet}(x^{new})$. Violin plots summarize conditional distributions of scores while gray lines show how model predictions change for particular observations between consecutive relaxations. This plot is based on wine dataset described in section 54.5.

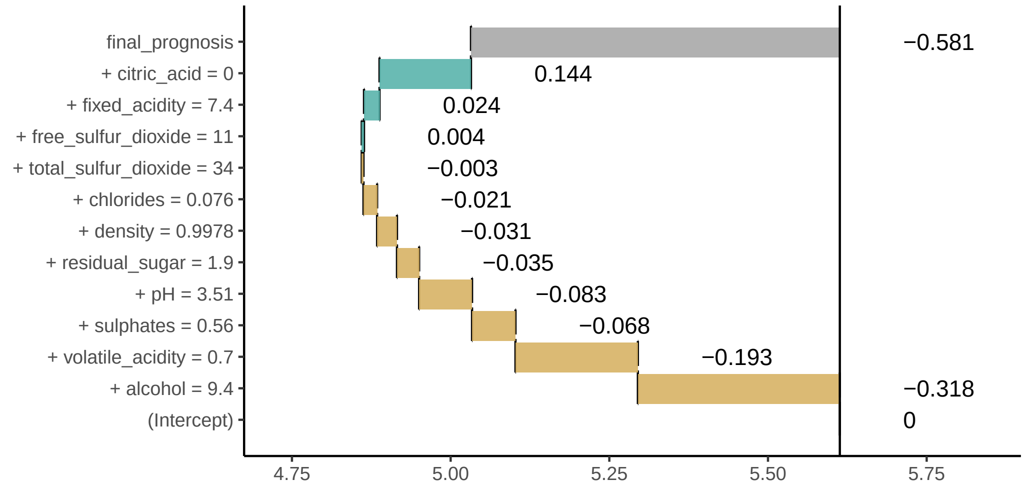


Figure 2: Break Down Plot for decomposition identified in Figure 1. Beginning and end of each rectangle corresponds to relaxed prediction (red dot in Figure 1) with and without a particular feature. This plot is based on wine dataset described in section 54.5.

```
 1: $p \leftarrow$ number of variables
 2: $IndSet \leftarrow \varnothing$ empty set
 3: for i in $\{1, \ldots, p\}$ do
 4:   Find new variable that can be relaxed with large distance to $f^{\varnothing}(x^{new})$
 5:   for j in $\{1, \ldots, p\} \setminus IndSet$ do
 6:     Calculate relaxed distance with $j$ added
 7:     $dist(j) \leftarrow d(x^{new}, IndSet \cup \{j\})$
 8:   end for
 9:   Find and add $j$ that maximize distance
10:   $j_{max} \leftarrow \arg\max_j dist(j)$
11:   $Contribution^{IndSet}(i) \leftarrow f^{IndSet \cup \{j_{max}\}}(x^{new}) - f^{IndSet}(x^{new})$
12:   $Variables(i) \leftarrow j_{max}$
13:   $IndSet \leftarrow IndSet \cup \{j_{max}\}$
14: end for
```

Algorithm 4: Model agnostic break down of model predictions. The *step-up* approach.

Case study: How good is this red wine?

Wine quality data (Cortez et al., 2009) is a well-known dataset which is commonly used as an example in predictive modeling. The main objective associated with this dataset is to predict the quality of some variants of Portuguese *Vinho Verde* on the basis of 11 chemical properties. A single observation from the dataset can be found in Table 1. According to the results from the original article, the Support Vector Machine (SVM) model performs better than other models including linear regression, neural networks and others.

In this section we will show how **live** package can be used to fit linear regression model locally and generate a visual explanation for the black box model as well as how **breakDown** package can be used to attribute parts of the final prediction to particular features. This example will focus on a regression problem, but since both methods are supposed to be model-agnostic, classification problems are treated in the same way by using scores (probabilities) rather than predicted classes.

fixed acidity	volatile acidity	citric acid	res. sugar	Cl^-	free SO_2	total SO_2	D	pH	SO_4^{2-}	alcohol
7.40	0.70	0.00	1.90	0.08	11.00	34.00	1.00	3.51	0.56	9.40

Table 1: The fifth observation in wine quality dataset. D denotes density, Cl^- stands for chlorides, "res." for residual and SO_4^{2-} for sulphates.

The SVM model[2] used in this example is trained with the use of **e1071** package (Meyer et al., 2018). We do not perform hyperparameter's tuning as the model performance is not relevant in this use-case.

```
library("e1071")
data(wine, package = "live")
wine_svm_model <- svm(quality~., data = wine)
```

Different approaches for explaining single predictions are illustrated on the basis of the prediction for the fifth wine from this dataset, the one presented in table 1. The actual quality of this wine is 5, while the quality predicted by the SVM model is 5.03.

```
predict(wine_svm_model, wine[5, ])
##        1
## 5.032032
```

The live package

The **live** package approximates black box model (here SVM model) with a simpler white box model (here linear regression model) to explain the local structure of a black box model and in consequence to assess how features contribute to a single prediction.

To do this, we first need to generate artificial observations around the selected observation x^{new} for local exploration. We use `sample_locally` function from **live** package.

[2]Access this model with `archivist::aread("MI2DataLab/live/arepo/1025d")`

```
library("live")
library("mlr")
wine_sim <- sample_locally(data = wine,
                           explained_instance = wine[5, ],
                           explained_var = "quality",
                           size = 2000,
                           seed = 17)
wine_sim_svm <- add_predictions(wine_sim, wine_svm)
nc <- which(colnames(wine) == "quality")
```

If multiple models are to be explained, there is no need to generate multiple *artificial* datasets. Predictions of each model on a single simulated dataset can be added with the use of `add_predictions` function. A different object should be created for each model, but the same result of a call to `sample_locally` function should be used as the `to_explain` argument. Black box model can be passed as a model object or as a name of **mlr** learner. While the object created by `sample_locally` function stores the dataset, name of the response variable and other metadata, the object returned by `add_predictions` function also stores the fitted black box model. The result of applying `sample_locally` functions does not contain the response but the result of `add_predictions` contains a column with model predictions, which has the same name as the response in the original dataset. The `seed` argument is passed to `set.seed` function to ensure reproducibility, though our experience with this approach shows that the results are stable.

Once the artificial data points around x^{new} are generated, we may fit the white box model to them. In this example we fit a linear regression model using `fit_explanation` function[3].

```
wine_expl_live <- fit_explanation(wine_sim_svm)
```

This function returns a native **mlr** object. The model object (for example, `lm` object) can be extracted with the use of `getLearnerModel` function.

The white box model `wine_expl` approximates the black box model `wine_svm_model` around x^{new}. Coefficients of this model can be presented graphically with the generic `plot` function for class `live_explainer`. See the corresponding Forest Plot in Figure 3 and the corresponding Waterfall Plot in Figure 4.

```
plot(wine_expl, type = "forest")
plot(wine_expl, type = "waterfall")
```

In case of datasets with larger number of variables, we could obtain sparse results by setting `selection = TRUE` in the `fit_explanation` function. This option allows for performance of variable selection based on LASSO implemented in **glmnet** (Friedman et al., 2010; Simon et al., 2011). When using Generalized Linear Model or Elastic Net as a white box model it is possible to set `family` argument to one of the distribution families available in `glm` and `glmnet` functions via `response_family` argument to `fit_explanation`.

As mentioned, other interpretable models are suitable as local explanations, too. In particular, decision trees can be used to visualization interactions between variables. It is enough to select a different white box model while calling `fit_explanation` function.

```
wine_expl_tree <- fit_explanation(wine_sim_svm, "regr.ctree", kernel = identity_kernel,
                                  hyperpars = list(maxdepth = 2))
```

Kernel is set to identity, because trees built by **party** package cannot handle non-integer weights. The result is the following decision tree in Figure 5.

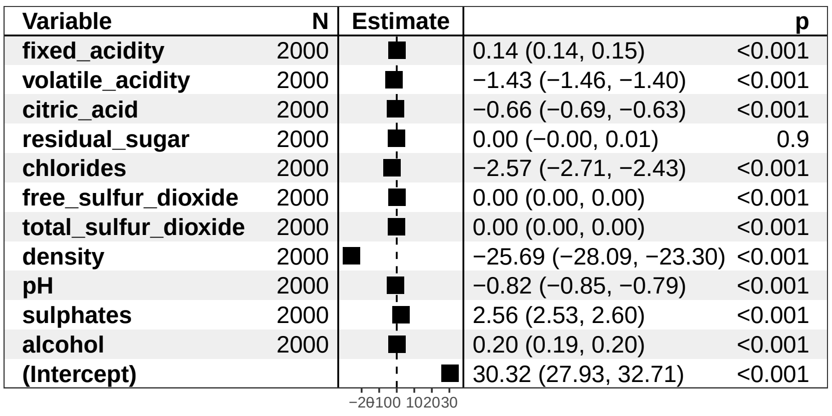
The lime package

The LIME method is implemented in the R package **lime**. It produces sparse explanations by default. In the first step a `lime` object is created for a specified dataset and a fitted black box model.

```
library("lime")
set.seed(17)
wine_expl_lime <- lime(wine[5, ], wine_svm_model)
```

Then we use the `explain` function, which in case of regression takes the observation of interest, `lime` object and the number of top features to be used for explanation[4]. In this case data is low dimensional, hence we can use all predictors. Alternatively, we could set `feature_select` to `none` to skip the selection part.

[3]Access this model with `archivist::aread("MI2DataLab/live/arepo/eebe6")`
[4]Access this object with `archivist::aread("MI2DataLab/live/arepo/878d5")`



Variable	N	Estimate		p
fixed_acidity	2000	■	0.14 (0.14, 0.15)	<0.001
volatile_acidity	2000	■	−1.43 (−1.46, −1.40)	<0.001
citric_acid	2000	■	−0.66 (−0.69, −0.63)	<0.001
residual_sugar	2000	■	0.00 (−0.00, 0.01)	0.9
chlorides	2000	■	−2.57 (−2.71, −2.43)	<0.001
free_sulfur_dioxide	2000	■	0.00 (0.00, 0.00)	<0.001
total_sulfur_dioxide	2000	■	0.00 (0.00, 0.00)	<0.001
density	2000	■	−25.69 (−28.09, −23.30)	<0.001
pH	2000	■	−0.82 (−0.85, −0.79)	<0.001
sulphates	2000	■	2.56 (2.53, 2.60)	<0.001
alcohol	2000	■	0.20 (0.19, 0.20)	<0.001
(Intercept)		■	30.32 (27.93, 32.71)	<0.001

Figure 3: Forest plot for a linear white box model that approximates the black box model.

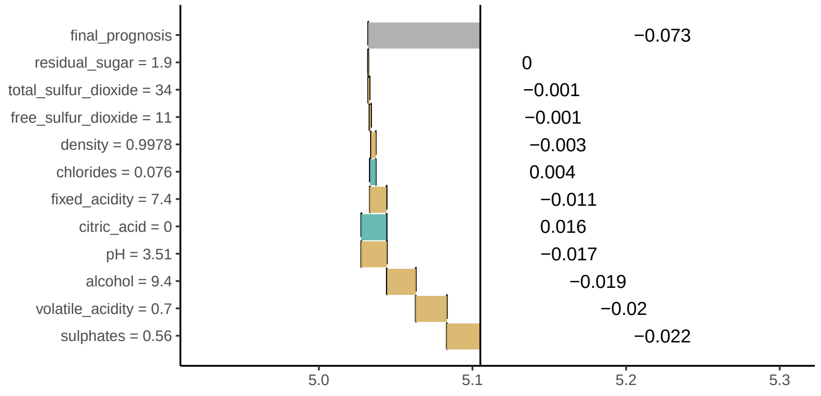


Figure 4: Waterfall plot for additive components of linear model that approximates the black box model around x^{new}.

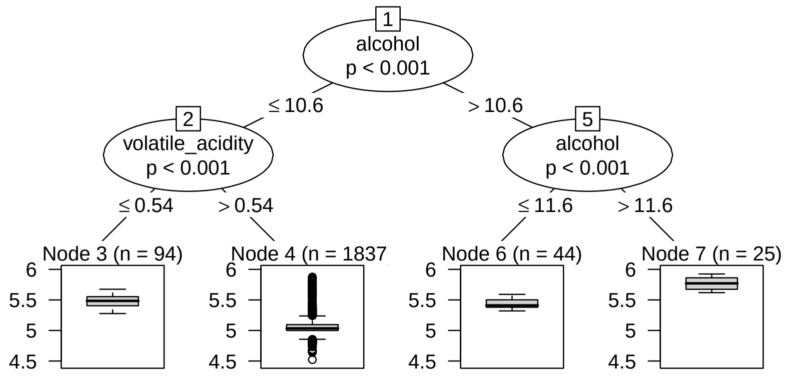


Figure 5: Decision tree that approximates the black box model around x^{new}.

```
model_type.svm <- function(x, ...) "regression"
svm_explained <- explain(wine[5, ], wine_expl_lime, n_features = 11)
plot_explanation(svm_explained)
```

Results produced by the `plot_explanation` function are presented in Figure 6.

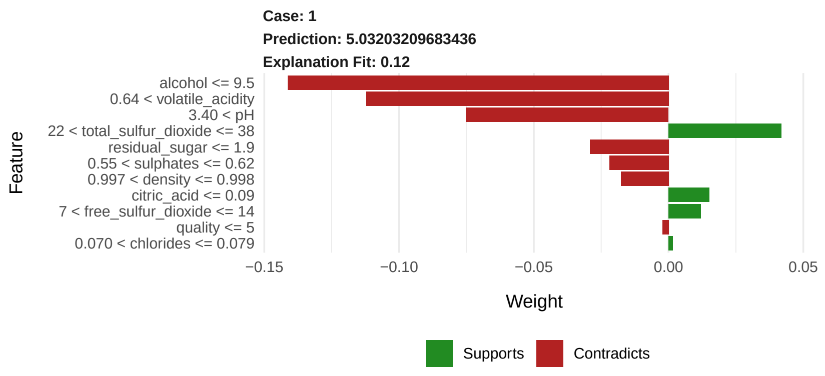


Figure 6: Contributions of particular features to the prediction calculated with SVM model assessed with LIME method.

Note that here, the explanation is presented in terms of discretized variables rather than original continuous predictors.

The breakDown package

The **breakDown** package directly calculates variable attributions for a selected observation. It does not use any surrogate model.

The `broken()` function is used to calculate feature attributions[5]. Generic functions `print()` and `plot()` show feature attributions as texts or waterfall plots. The `baseline` argument specifies the origin of a waterfall plot. By default, it is 0. Use `baseline = "intercept"` to set the origin to average model prediction.

```
library("breakDown")
explain_bd <- broken(wine_svm, new_observation = wine[5, -nc],
                     data = wine[, -nc],
                     baseline = "Intercept",
                     keep_distributions = TRUE)
explain_bd
##                             contribution
## baseline                           5.613
## + alcohol = 9.4                   -0.318
## + volatile_acidity = 0.7          -0.193
## + sulphates = 0.56                -0.068
## + pH = 3.51                       -0.083
## + residual_sugar = 1.9            -0.035
## + density = 0.9978                -0.031
## + chlorides = 0.076               -0.021
## + total_sulfur_dioxide = 34       -0.003
## + quality = 5                      0.000
## + free_sulfur_dioxide = 11         0.004
## + fixed_acidity = 7.4              0.024
## + citric_acid = 0                  0.144
## final_prognosis                    5.032

plot(explain_bd)
```

Figure 7 shows variable contributions for step-up and step-down strategy. Variable ordering is different but the contributions are consistent across both strategies.

[5]Access example attributions with `archivist::aread("MI2DataLab/live/arepo/1f320")`

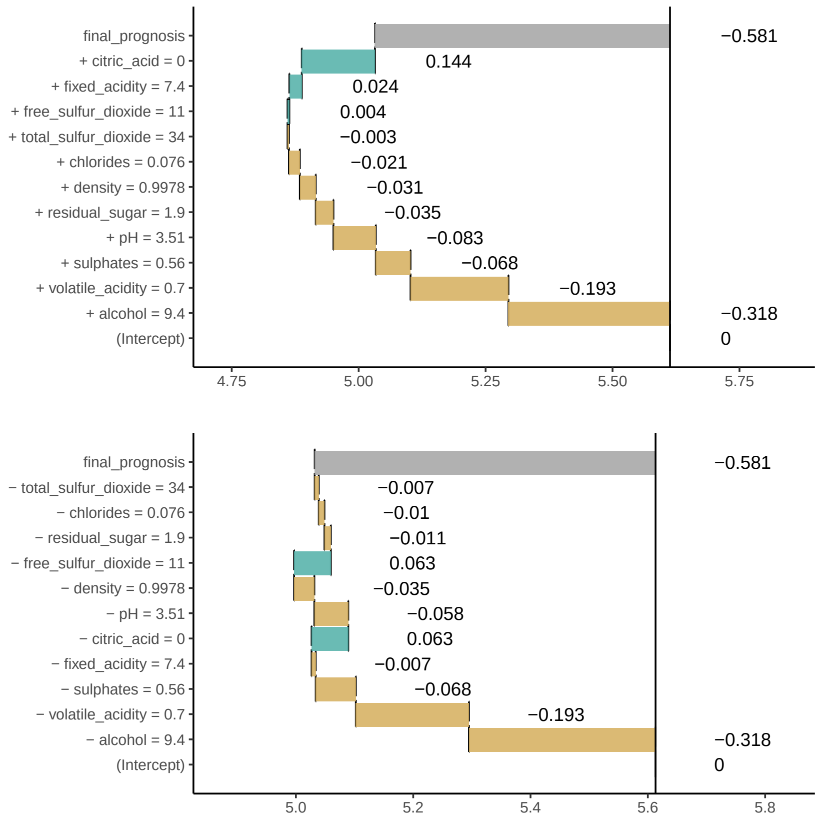


Figure 7: ag-break feature attributions for SVM model calculated for the 5th wine. The upper plot presents feature attributions for the step-up strategy, while the bottom plot presents results for the step-down strategy. Attributions are very similar even if the ordering is different. Vertical black line shows the average prediction for the SVM model. The 5th wine gets final prediction of 5.032 which is below the average for this model by 0.581 point.

Find more examples for classification and regression models created with **caret**, **mlr**, **randomForest** (Liaw and Wiener, 2002) and other frameworks in package vignettes[6].

[6]https://pbiecek.github.io/breakDown/

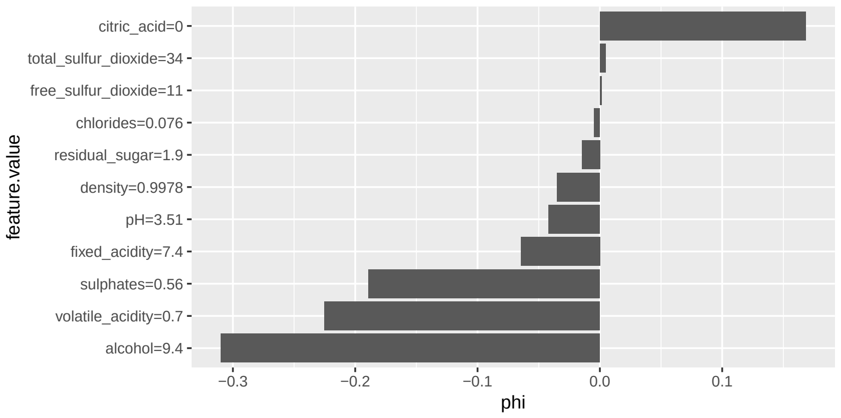


Figure 8: Visualization of Shapley values.

Shapley values (SHAP)

Authors of the original article about Shapley values maintain a Python package which implements several methods of computing these values. Also, it provides visual diagnostic tools that help understand global behavior of the black box model such as plotting variable contributions for all instances in the dataset.

The **iml** package can be used to compute Shapley values for any model. Other tools are more restricted in this regard, for example, the **shapleyR** package only works with **mlr** models. First, we need to create a `Predictor` object for the model, then it is enough to apply the `Shapley` function for a specified instance. This implementation samples permutation of variables to take into account different orders in which conditioning can be done and the number of permutations to be used is a parameter of this function. Details can be found in the documentation for the package.

```
model_svm = Predictor$new(model = wine_svm, data = wine[, -nc], y = wine$quality)
set.seed(17)
shapley_iml = Shapley$new(model_svm, x.interest = wine[5, -nc])
plot(shapley_iml)
```

Calculated Shapley values[7] are presented in Table 2 and compared to results of Break Down method. Shapley values can be also visualized on a plot similar to waterfall plot, which can be created using generic `plot` function. An example is shown in Figure 8.

	feature	Shapley value	breakDown score
1	alcohol	-0.31	-0.32
2	volatile_acidity	-0.23	-0.19
3	sulphates	-0.19	-0.07
4	citric_acid	0.17	0.14
5	fixed_acidity	-0.06	0.02
6	pH	-0.04	-0.08
7	density	-0.03	-0.03
8	residual_sugar	-0.01	-0.03
9	total_sulfur_dioxide	0.00	-0.00
10	chlorides	-0.00	-0.02
11	free_sulfur_dioxide	0.00	0.00

Table 2: Comparison of feature attributions calculated with **iml** and **breakDown** packages.

Discussion

In this paper, we presented four approaches and four R packages that can be used for explanations of predictions from complex black box models. Two of them have already been introduced in literature, while **live** and **breakDown** were introduced in this article for the first time.

[7]Access this object with `archivist::aread("MI2DataLab/live/arepo/50f5f")`

All four approaches are model agnostic in a sense that, the method does not depend on any particular structure of black box model. Yet there are also some differences between these approaches.

1. Surrogate models vs. conditional expected responses. **live** and **lime** packages use surrogate models (the so-called white box models) that approximate local structure of the complex black box model. Coefficients of these surrogate models are used for explanations. Unlike them, **breakDown** and **shapleyR** construct feature attributions on the basis of conditional responses of a black box model.
2. **live** and **lime** packages differ in terms of the manner in which the surrounding of x^{new} is defined. This task is highly non-trivial especially for mixed data with continuous and categorical features. **live** does not use *interpretable input space* (and so does not fall under the *additive feature attribution methods* category), but approximates the black box model directly in the data space, what can be considered a more effective use of data. It comes with no theoretical guarantees that are provided for Shapley values, but apart from being very intuitive, it offers several tools for visual inspection of the model.
3. Computations required to obtain LIME and LIVE explanations are relatively simple. For large datasets, sampling can be easily parallelized. Shapley values and Break Down Plots, on the other hand, are more computationally demanding, but the current implementation of **breakDown** computes additive explanations in linear time.
4. **shapleyR** and **breakDown** take conditional expectation of the predictor function with respect to explanatory features. They differ in terms of the manner in which conditioning is applied to calculating feature attributions. **shapleyR** is based on results from game theory; in this package contribution of a single feature is averaged across all possible conditionings. **breakDown** uses a greedy approach in which only a single series of nested conditionings is considered. The greedy approach is easier to interpret and faster to compute. Moreover, exact methods of computing Shapley values exist only for linear regression and tree ensemble models. Approximate computations are also problematic, as they require the choice of the number of samples of subsets of predictors which will be used. These two methods produce nearly identical results for linear models (see table Table 2), but for more complex models the estimated contributions can be very different, even pointing in opposite directions. An advantage of Shapley values are proven theoretical properties, though they are restricted to explanation models that belong to the *additive feature attribution methods* class.
5. When parameters (kernel and regularization term) are chosen as in Lundberg and Lee (2017), **lime** produces estimates of Shapley values, while other choices of kernel and penalty term lead to inconsistent results. The fact that the suggested penalty term is equal to 0 can be considered a huge limitation of LIME and SHAP, because in this setting they will not produce sparse explanations.
6. All presented methods decompose final prediction into additive components attributed to particular features. This approach will not work well for models with heavy components related to interactions between features.
7. Evaluation of human-readability of discussed methods will be a subject of future research. It will help in making comparisons between different approaches to individual prediction explanations.
8. Authors of SHAP and LIME proposed methods of combining explanations calculated for different observations into global explanations. Such aggregation methods for Break Down and LIVE will also be a subject of future research.

Comparison of the methods presented in the previous section is far from being comprehensive. More research is needed to better understand the differences between these approaches and new approaches are needed to overcome the constraints listed above. Nevertheless, the availability of the mentioned packages creates an opportunity for further studies on model exploration.

Acknowledgements

Work on this article is financially supported by the *NCN Opus grant 2017/27/B/ST6/01307*.

Bibliography

A. Altmann, L. Toloşi, O. Sander, and T. Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010. ISSN 1460-2059, 1367-4803. [p395]

D. W. Apley. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *ArXiv e-prints*, 2016. [p395]

P. Biecek. DALEX: Explainers for complex predictive models. *ArXiv e-prints*, 2018. [p396]

P. Biecek and M. Kosinski. archivist: An R Package for Managing, Recording and Restoring Data Analysis Results. *Journal of Statistical Software*, 82(11):1–28, 2017. URL `https://doi.org/10.18637/jss.v082.i11`. [p396]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016. [p395, 574]

T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, and Y. Li. *Xgboost: Extreme Gradient Boosting*, 2018. URL `https://CRAN.R-project.org/package=xgboost`. R package version 0.71.2. [p396]

P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decis. Support Syst.*, 47(4):547–553, 2009. ISSN 0167-9236. URL `https://doi.org/10.1016/j.dss.2009.05.016`. [p401]

M. Dowle and A. Srinivasan. *Data.table: Extension of 'data.frame'*, 2017. URL `https://CRAN.R-project.org/package=data.table`. R package version 1.10.4-3. [p397]

A. Fogg. *Anthony Goldbloom Gives You the Secret to Winning Kaggle Competitions*, 2016. [p395]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL `https://doi.org/10.18637/jss.v033.i01`. [p402]

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5): 1189–1232, 2001. URL `https://doi.org/10.1214/aos/1013203451`. [p395]

A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015. ISSN 1061-8600, 1537-2715. URL `https://doi.org/10.1080/10618600.2014.907095`. [p395]

B. M. Greenwell. Pdp: An r package for constructing partial dependence plots. *The R Journal*, 9(1): 421–436, 2017. URL `https://doi.org/10.32614/rj-2017-016`. [p395]

N. Kennedy. *Forestmodel: Forest Plots from Regression Models*, 2017. URL `https://CRAN.R-project.org/package=forestmodel`. R package version 0.4.3. [p397]

M. Kuhn. *Caret: Classification and Regression Training*, 2018. URL `https://CRAN.R-project.org/package=caret`. R package version 6.0-80. [p395]

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. [p405]

S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. [p396, 407]

D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *E1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2018. URL `https://CRAN.R-project.org/package=e1071`. R package version 1.7-0. [p401]

C. Molnar, B. Bischl, and G. Casalicchio. Iml: An r package for interpretable machine learning. *JOSS*, 3 (26):786, 2018. URL `https://doi.org/10.21105/joss.00786`. [p395]

T. L. Pedersen and M. Benesty. *Lime: Local Interpretable Model-Agnostic Explanations*, 2018. URL `https://CRAN.R-project.org/package=lime`. R package version 0.4.1. [p395]

N. Puri, P. Gupta, P. Agarwal, S. Verma, and B. Krishnamurthy. MAGIX: Model Agnostic Globally Interpretable Explanations. *ArXiv e-prints*, 2017. [p395]

M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. URL `https://doi.org/10.1145/2939672.2939778`. [p395, 396]

M. Robnik-Šikonja and I. Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):589–600, 2008. ISSN 1041-4347. URL https://doi.org/10.1109/tkde.2007.190734. [p399]

N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, 39(5):1–13, 2011. URL https://doi.org/10.18637/jss.v039.i05. [p402]

C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis. Conditional variable importance for random forests. *BMC Bioinformatics*, 9(307), 2008. [p397]

M. Tulio Ribeiro, S. Singh, and C. Guestrin. Nothing Else Matters: Model-Agnostic Explanations By Identifying Prediction Invariance. *ArXiv e-prints*, 2016. [p395]

F. Y. Tzeng and K. L. Ma. Opening the black box - data driven visualization of neural networks. In *VIS 05. IEEE Visualization, 2005.*, pages 383–390, 2005. URL https://doi.org/10.1109/visual.2005.1532820. [p395]

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014*, pages 818–833. Springer-Verlag, 2014. ISBN 978-3-319-10590-1. URL https://doi.org/10.1007/978-3-319-10590-1_53. [p395]

E. Štrumbelj and I. Kononenko. A general method for visualizing and explaining black-box regression models. In *Proceedings of the 10th International Conference on Adaptive and Natural Computing Algorithms - Volume Part II*, ICANNGA'11, pages 21–30, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-20266-7. URL https://doi.org/10.1007/978-3-642-20267-4_3. [p395]

Mateusz Staniak
Faculty of Mathematics and Information Science
Warsaw University of Technology
Poland
m.staniak@mini.pw.edu.pl

Przemysław Biecek
Faculty of Mathematics and Information Science
Warsaw University of Technology
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
Poland
ORCiD: 0000-0001-8423-1823
przemyslaw.biecek@gmail.com

Downside Risk Evaluation with the R Package GAS

by David Ardia, Kris Boudt, and Leopoldo Catania

Abstract Financial risk managers routinely use non–linear time series models to predict the downside risk of the capital under management. They also need to evaluate the adequacy of their model using so–called backtesting procedures. The latter involve hypothesis testing and evaluation of loss functions. This paper shows how the R package **GAS** can be used for both the dynamic prediction and the evaluation of downside risk. Emphasis is given to the two key financial downside risk measures: Value-at-Risk (VaR) and Expected Shortfall (ES). High-level functions for: (i) prediction, (ii) backtesting, and (iii) model comparison are discussed, and code examples are provided. An illustration using the series of log–returns of the Dow Jones Industrial Average constituents is reported.

Introduction

The **GAS** package of Catania et al. (2016) provides a complete framework for modeling, estimating and predicting time series processes for which the time variation in the parameters is driven by the score of the conditional density function. This increasingly popular class of score-driven models has been introduced by Creal et al. (2013) and Harvey (2013). Ardia et al. (2019) describe the general functionality implemented in the **GAS** package, but do not cover the functionality useful for the estimation and backtesting of Value-at-Risk (VaR) and Expected Shortfall (ES), which are the two leading risk measures used in finance. The aim of this paper is to show how the functions available in the **GAS** package can be used for VaR and ES evaluation, prediction, and backtesting.

The economic relevance of this topic follows partly from the Basel Accords (currently the Basel III Accords), which impose that banks and financial institutions have to meet capital requirements, and must rely on state-of-the-art risk systems. In particular, they must assess the uncertainty about the future values of their portfolios and estimate the extent and the likelihood of potential losses using a risk measure. Nowadays, VaR and ES risk measures are the standards (Jorion, 1997). For an asset (or portfolio) return, the VaR at a given time horizon equals the return such that lower returns only occur with a given probability level α (referred to as the *risk level*, and which is typically set to one or five percent, that is $\alpha \in \{0.01, 0.05\}$). The ES risk measure is the expectation of the asset (or portfolio) return when the return is below the VaR level.

The estimation of the VaR and ES thus requires first to accurately estimate the conditional distribution of the future portfolios' or assets' returns. Formally, assuming a continuous cumulative density function (*cdf*) with time-varying parameters $\boldsymbol{\theta}_t \in \mathbb{R}^d$ and additional static parameters $\boldsymbol{\psi} \in \mathbb{R}^q$, $F(\cdot; \boldsymbol{\theta}_t, \boldsymbol{\psi})$, for the log return at time t, $r_t \in \mathbb{R}$, the $VaR_t(\alpha)$ is given by:

$$VaR_t(\alpha) \equiv F^{-1}(\alpha; \boldsymbol{\theta}_t, \boldsymbol{\psi}) \,,$$

where $F^{-1}(\cdot)$ denotes the inverse of the *cdf*, that is, the quantile function. It follows that $VaR_t(\alpha)$ is nothing more than the α–quantile of the return distribution at time t.[1] The ES metric measures the expected loss after a violation of the VaR level and it is defined as:

$$ES_t(\alpha) \equiv \frac{1}{\alpha} \int_{-\infty}^{VaR_t} z \, dF(z, \boldsymbol{\theta}_t, \boldsymbol{\psi}) \,.$$

It follows that a crucial point for correct VaR and ES assessment is the determination of $F(\cdot)$ and its parameters $\boldsymbol{\theta}_t$ and $\boldsymbol{\psi}$. For a general overview of existing methods, we refer the reader to Nieto and Ruiz (2016). In this paper, we illustrate how this can be achieved using the framework of Generalized Autoregressive Score (GAS) models introduced by Creal et al. (2013) and Harvey (2013). GAS models are also referred to as Score Driven (SD) models and Dynamic Conditional Score (DCS) models and have been used extensively for financial risk management purposes. For a comparison between the accuracy of VaR and ES estimates obtained by the GAS approach against alternative volatility models, we refer the reader to Bernardi and Catania (2016), Gao and Zhou (2016), Ardia et al. (2018) and Ardia et al. (2019).

Formally, in GAS models the vector of time-varying parameters, $\boldsymbol{\theta}_t$, is updated through a dynamic

[1]Sometimes VaR is defined with respect to the loss variable $l_t = -r_t$. All the arguments of this paper can be easily adapted to this case.

equation based on the score of the conditional[2] probability density function of r_t, $f(\cdot;\boldsymbol{\theta}_t,\boldsymbol{\psi})$, that is:

$$\boldsymbol{\theta}_{t+1} \equiv \boldsymbol{\kappa} + \mathbf{A}\mathbf{s}_t + \mathbf{B}\boldsymbol{\theta}_t\,, \tag{1}$$

where $\mathbf{s}_t$ is the (possibly) scaled score of $f(\cdot;\boldsymbol{\theta}_t,\boldsymbol{\psi})$ with respect to $\boldsymbol{\theta}_t$, evaluated in r_t; see Creal et al. (2013) and also the Appendix for examples. The coefficients $\boldsymbol{\kappa}$, $\mathbf{A}$, and $\mathbf{B}$ control for the evolution of $\boldsymbol{\theta}_t$ and need to be estimated along with $\boldsymbol{\psi}$ from the data, usually by maximum likelihood.

We focus on the three major steps practitioners involved in risk management face during their job: (i) prediction of future downside risk, (ii) backtesting, and (iii) comparison with alternative models. The empirical part of the article deals with these three points from an applied perspective while the computational part details the **GAS** functionalities devoted to downside risk.

A flexible GAS specification for modeling financial returns

Financial returns exhibit several stylized facts that need to be taken into consideration to produce reliable risk forecasts. Empirically, the distribution of returns is (left) skewed and fat tailed, and its variance is time varying (*i.e.*, returns exhibit the so-called *volatility clustering*); see, for example, McNeil et al. (2015).

To account for these features, we consider a very flexible specification, in which we assume that the log return at time t, r_t, is distributed conditionally on past observations as follows:

$$r_t|\mathcal{I}_{t-1} \sim \mathcal{SKST}(r_t;\mu,\sigma_t,\xi,\nu)\,,$$

where $\mathcal{I}_{t-1}$ is the information set up to time $t-1$, and $\mathcal{SKST}(r_t;\cdot)$ denotes the skew Student-t distribution of Fernández and Steel (1998) with location $\mu \in \mathbb{R}$, time-varying scale (volatility) $\sigma_t > 0$, and skewness and shape parameters $\xi > 0$ and $\nu > 2$, respectively. We parametrize the $\mathcal{SKST}$ distribution as in Bauwens and Laurent (2005) such that $\mathsf{E}[r_t\,|\,\mathcal{I}_{t-1}] = \mu$ and $\mathsf{V}[r_t\,|\,\mathcal{I}_{t-1}] = \sigma_t^2$. To ensure positivity of the volatility parameter, we set the time-varying GAS parameter $\boldsymbol{\theta}_t$ in (1) to $\boldsymbol{\theta}_t \equiv \theta_t \equiv \log\sigma_t$, and we define $\boldsymbol{\psi} \equiv (\mu,\xi,\nu)$. In the Appendix, we show that the corresponding score s_t which enters linearly in the updating equation (1) is given by:

$$s_t \equiv \left(\frac{z_t\,(\nu+1)\,(z_t - m)}{(\xi_t^*)^2\,(\nu-2) + z_t^2} - 1\right)\,,$$

with $\xi_t^* \equiv \xi^{I\{z_t\geq 0\} - I\{z_t \leq 0\}}$, where $I\{\cdot\}$ is the indicator function equal to one if the condition holds, and zero otherwise, and with $z_t \equiv \left(\frac{r_t-\mu}{\sigma_t}\right)k + m$, where expressions for k and m are provided in the Appendix. For a fixed value of the asymmetry parameter ξ, we see that the effect of shock on future values is dampened when ν decreases. Starting from the general $\mathcal{SKST}$ distribution, we recover as special cases:

(i) the Student-t distribution, $\mathcal{ST}$, imposing $\xi = 1$;

(ii) the Normal distribution, $\mathcal{N}$, imposing $\nu = \infty$ and $\xi = 1$.

(See the Appendix for the score in these cases.) Given a series of T log returns, $r_1,\ldots,r_T$, the model parameters are estimated by maximizing the log-likelihood function; see Blasques et al. (2014). Prediction with GAS models is straightforward thanks to the recursive nature of the updating equation (1). Specifically, the one-step ahead predictive distribution $F(\cdot;\widehat{\boldsymbol{\theta}}_{T+1},\widehat{\boldsymbol{\psi}})$ is available in closed form whereas the h-step ahead distribution ($h>1$) needs to be simulated; see Blasques et al. (2016). VaR and ES forecasts are easily obtained from the predictive distribution.

Evaluating downside risk forecasts

The recursive method of forecasting is usually employed to backtest the adequacy of a statistical model, as well as to perform models comparisons in terms of VaR and ES predictions (Marcellino et al., 2006). The objective of a backtesting analysis is to verify the precision of the prediction by separating the estimation window and the evaluation period. The objective of a model comparison analysis is usually to order models according to a loss function.

To this end, the full sample of T returns is divided into an in-sample period of length S, and an out-of-sample period of length H. Model parameters are first estimated over the in-sample period,

[2]The conditioning is intended with respect to the past observations r_{t-s} ($s > 0$), however, for notational purposes, this is not always reported.

subsequently the h-step ahead prediction of the return distribution at time $S+h$ is generated along with the corresponding VaR and ES measures. These steps are repeated augmenting the in-sample period with new observations in a recursive way until we reach the end of the series, T. If during the data augmentation step, past observations are eliminated, we are considering a rolling window, otherwise we have an expanding window. In this paper, we follow the standard approach in daily risk management of a typical trading desk and use the rolling window configuration with $h = 1$.

VaR backtesting

Once a series of VaR predictions is available, forecasts adequacy is assessed through backtesting procedures. VaR backtesting procedures usually check the correct coverage of the unconditional and conditional left-tail of the log-returns distribution. Correct unconditional coverage (UC) was first considered by Kupiec (1995), while correct conditional coverage (CC) by Christoffersen (1998). The main difference between UC and CC concerns the distribution we are focusing on. For instance, UC considers correct coverage of the left-tail of the unconditional log-return distribution, $f(r_t)$, while CC deals with the conditional density $f(r_t|\mathcal{I}_{t-1})$. From an inferential perspective, UC looks at the ratio between the number of realized VaR violations observed from the data and the expected number of VaR violations implied by the chosen risk level, α, during the forecast period, that is, αH. In order to investigate CC, Christoffersen (1998) proposed a test on the series of VaR exceedance $\{d_t, t = S, \dots, S+H\}$, where $d_t \equiv I\{r_t < VaR_t(\alpha)\}$, usually referred to as the *hitting series*. Specifically, if correct conditional coverage is achieved by the model, VaR exceedances should be independently distributed over time.

The DQ test by Engle and Manganelli (2004) assesses the joint hypothesis that $\mathsf{E}[d_t] = \alpha$ and the hit variables are independently distributed. The implementation of the test involves the de-meaned process $Hit_t^\alpha \equiv d_t - \alpha$. Under correct model specification, unconditionally and conditionally, Hit_t^α has zero mean and is serially uncorrelated. The DQ test is then the traditional Wald test of the joint nullity of all coefficients in the following linear regression:

$$Hit_t^\alpha = \delta_0 + \sum_{l=1}^{L} \delta_l Hit_{t-l}^\alpha + \delta_{L+1} VaR_{t-1}(\alpha) + \epsilon_t .$$

Under the null hypothesis of correct unconditional and conditional coverage, we have that the Wald test statistic is asymptotically chi-square distributed with $L+2$ degrees of freedom. Engle and Manganelli (2004) set $L = 4$ lags, which has become the standard choice.

VaR model comparison

Real world applications consider several models for VaR prediction. If correct unconditional/conditional coverage is achieved by more than one model, the practitioner faces the problem of not being able to choose between different alternatives. In this situation, model comparison techniques are used to choose the best performing model. Model ranking is achieved thanks to the definition of a loss function. Among several available loss functions for quantile prediction (McAleer and Da Veiga, 2008), the Quantile Loss (QL) used for quantile regressions (Koenker and Bassett, 1978) is one of the most frequent choices in the VaR context; see González-Rivera et al. (2004). Formally, given a VaR prediction at risk level α for time t, the associated quantile loss, $QL_t(\alpha)$, is defined as:

$$QL_t(\alpha) \equiv (\alpha - d_t)(r_t - VaR_t(\alpha)) . \tag{2}$$

QL is an asymmetric loss function that penalizes more heavily with weight $(1-\alpha)$ the observations for which we observe returns showing VaR exceedance. Quantile losses are then averaged over the forecasting period, and models with lower averages are preferred. The outperformance of model $\mathcal{A}$ versus model $\mathcal{B}$ is finally assessed looking at the ratio between the average QLs, associated with the two models, that is, if $QL_{\mathcal{A}}/QL_{\mathcal{B}} < 1$ then model $\mathcal{A}$ outperforms model $\mathcal{B}$ and vice versa.

Joint VaR and ES model comparison

The quantile loss function in (2) for VaR assessment is appropriate since quantiles are elicited by it, that is, when the conditional distribution is static over the sample, the VaR can be estimated by minimizing the average quantile loss function. Unfortunately, there is no loss function available for which the ES risk measure is elicitable; see, for instance, Bellini and Bignozzi (2015) and Ziegel (2016). However, it has been recently shown by Fissler and Ziegel (2016) (FZ) that the couple (VaR, ES) is jointly elicitable, as the values of v_t and e_t that minimize the sample average of the following loss function:

$$FZ(r_t, v_t, e_t, \alpha, G_1, G_2) \equiv (d_t - \alpha)\left(G_1(v_t) - G_1(r_t) + \frac{1}{\alpha}G_2(e_t)v_t\right) - G_2(e_t)\left(\frac{1}{\alpha}d_t r_t - e_t\right) - \mathcal{G}_2(e_t)\,,$$

where G_1 is weakly increasing, G_2 is strictly positive and strictly increasing, and $\mathcal{G}_2' = G_2$. The **GAS** package implements the FZ loss function for a specific choice of G_1, G_2 and $\mathcal{G}_2$. We set $G_1(x) = 0$ and $G_2(x) = -1/x$ and assume the values of VaR and ES to be strictly negative; see Patton et al. (2017) and Ardia et al. (2018) for a similar approach. For VaR and ES predictions at risk level α for time t, the associated joint loss function (FZL) is then given by:

$$FZL_t^\alpha \equiv \frac{1}{\alpha\, ES_t^\alpha}\, d_t\,(r_t - VaR_t^\alpha) + \frac{VaR_t^\alpha}{ES_t^\alpha} + \log(-ES_t^\alpha) - 1\,, \tag{3}$$

for $ES_t^\alpha \leq VaR_t^\alpha < 0$. As for QL, FZ losses are averaged over the forecasting period and models with lower averages are preferred.

Summarizing, given a set of available models, a typical downside risk forecasting exercise consists of the following three major steps:

(i) perform rolling forecast during the out-of-sample period;

(ii) perform statistical backtest of VaR predictions using UC, CC, and DQ tests;

(iii) perform VaR and joint VaR and ES model comparison looking at the average QL and FZL of each model.

The next section is devoted to detailing the implementation of each of these steps with the **GAS** package.

Empirical illustration

We first briefly review how to make predictions with GAS models using the **GAS** package. The main illustration is for the last $T = 2{,}500$ observations of the daily log-returns of the General Electric stock in the dataset `dji30ret`, which is a dataframe consisting of the thirty Dow Jones Industrial Average constituents:

```
> library("GAS")
> data("dji30ret", package = "GAS")
> dji30ret <- tail(dji30ret, 2500)
```

The corresponding returns are shown as gray points in Figure 1. We see the time-variation in the volatility of the daily return series, which we model next using the GAS specification with skewed Student-t innovations, as previously described. Thanks to the **GAS** package, it is straightforward to estimate the GAS model on rolling windows of the available data and to make one-step ahead rolling forecasts.

Specifically, the user needs to specify the model through the `UniGASSpec()` function, and then perform rolling predictions with the `UniGASRoll()` function. In the **GAS** package, models are specified through the definition of the conditional distribution assumed for the data, `Dist`, and the list of time-varying parameters, `GASpar`.

`Dist` is a character equal to the label of the distribution. For instance, $\mathcal{SKST}$ is identified as `"sstd"`, $\mathcal{ST}$ as `"std"`, and $\mathcal{N}$ as `"norm"`; see Table 1 of Ardia et al. (2019) for the list of distributions and associated labels available in the **GAS** package.

`GASPar` is a `list` with named `boolean` elements. Entries name are: `location`, `scale`, `skewness`, and `shape`. These indicate whether the associated distribution parameters are time varying or not. By default we have `GASPar = list(location = FALSE,scale = TRUE,skewness = FALSE,shape = FALSE)`, that is, only volatility is time varying. For instance, in order to specify the three GAS models: GAS–$\mathcal{N}$, GAS–$\mathcal{ST}$, and GAS–$\mathcal{SKST}$, we need to execute the following lines:

```
> GASSpec_N <- UniGASSpec(Dist = "norm", GASPar = list(scale = TRUE))
> GASSpec_ST <- UniGASSpec(Dist = "std", GASPar = list(scale = TRUE))
> GASSpec_SKST <- UniGASSpec(Dist = "sstd", GASPar = list(scale = TRUE))
```

`UniGASSpec()` delivers an object of the class `"uGASSpec"` which comes with several methods; see `help("UniGASSpec")`.

The `UniGASRoll()` function accepts an object of the class `"uGASSpec"`, `"GASSpec"`, a `numeric` vector for the series of returns, `data`, and other arguments, such as:

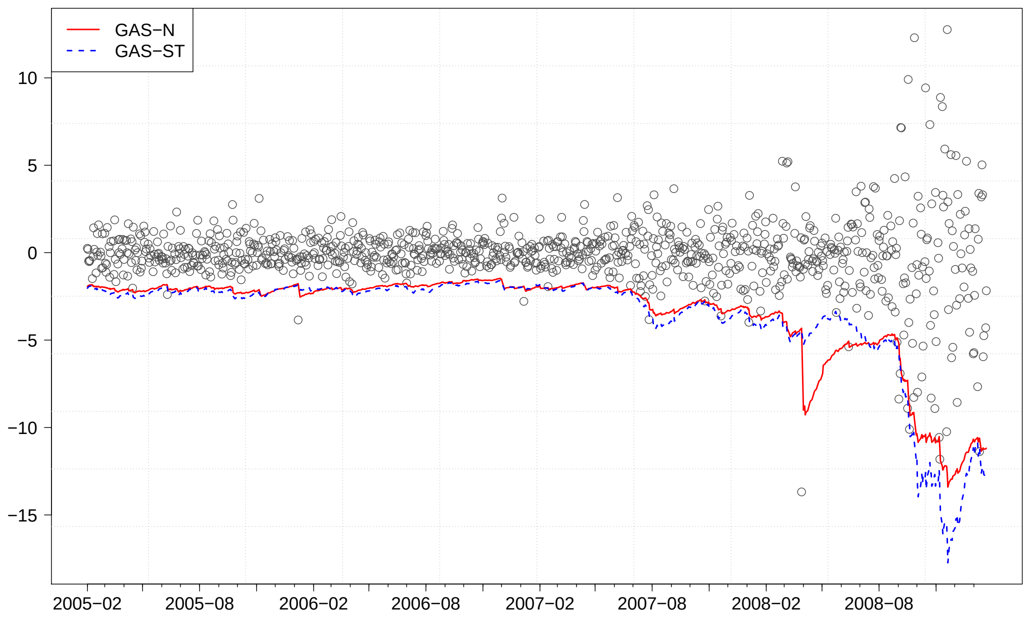


Figure 1: One-step ahead VaR forecasts for General Electric (GE) at the $\alpha = 1\%$ risk level for the GAS–$\mathcal{N}$ (*solid*) and GAS–$\mathcal{ST}$ (*dotted*) models. Gray points indicate realized log returns calculated as the differences between the natural logarithm of two consecutive prices. The forecasting period ranges from February 14, 2005, to February 3, 2009, for a total of $H = 1{,}000$ out-of-sample observations.

- the length of the out-of-sample period: `ForecastLength`;
- the type of the rolling window used to update the data: `RefitWindow`;
- the number of observations within each model re-estimation: `RefitEvery`,

among others; see `help("UniGASRoll")`. `ForecastLength` and `RefitEvery` are `numeric` elements while `RefitWindow` is a `character` equal to `"moving"` (the default) for a rolling window scheme or `"recursive"` for an expanding window. As previously mentioned, in this paper we consider the case `RefitWindow = "moving"`. We fix the length of the out-of-sample period to $H = 1000$, and re-estimate the model parameters at the weekly frequency (i.e., every 5 new observations). One-step ahead rolling predictions for the first series of returns using the GAS–$\mathcal{N}$ model are then computed as:

```
> library("parallel")
> cluster <- makeCluster(2)
> H <- 1000
> Roll_N <- UniGASRoll(dji30ret[, "GE"], GASSpec_N, RefitEvery = 5,
  cluster = cluster, ForecastLength = H)
```

We have also made use of parallel processing (with 2 cores) through the definition of a `cluster` object exploiting the **parallel** package included in R since version 2.14.0.

The output of `UniGASRoll()` is an object of the class `"uGASRoll"` which comes with several methods; see `help("UniGASRoll")`.

VaR and ES one-step ahead rolling forecasts at the risk level $\alpha = 0.01$ can be computed from `Roll_N` using the `quantile` and `ES` methods, respectively:

```
> alpha <- 0.01
> VaR_N <- quantile(Roll_N, probs = alpha)
> ES_N <- ES(Roll_N, probs = alpha)
```

`VaR_N` and `ES_N` are matrices of dimension $1{,}000 \times 1$ containing the VaR and ES forecasts at the 1% risk level.[3] While VaR predictions are obtaining by numerical inversion of the predicted cumulative density function, ES predictions are computed by numerical adaptive integration of the predicted

[3]The `probs` argument in `quantile` and `ES` can also be a `numeric` vector of p VaR levels. In this case, `VaR_N` and `ES_N` would be a $1000 \times p$ `matrix`.

density relying on the **cubature** package (Narasimhan and Johnson, 2017). Multi-step ahead prediction of VaR and ES are obtained by Monte Carlo simulation and are still computed with the `quantile` and `ES` methods. To compute multi-step ahead predictions an object of class `"uGASFor"` delivered by the function `UniGASFor` has to be provided; see `help("UniGASFor")`.

Predictions for the GAS–$\mathcal{ST}$ and GAS–$\mathcal{SST}$ specifications using the `GASSpec_ST` and `GASSpec_SKST` model definitions are computed analogously. Figure 1 reports 1% VaR predictions delivered by the GAS–$\mathcal{N}$ (*solid* line) and the GAS–$\mathcal{ST}$ (*dotted* line). We see the impact of the recent Global Financial Crisis on the volatility of the series. Indeed, the 2007-2008 returns of GE present much more variability than over the period 2005-2006, which is translated into lower values for VaR. What is also clearly evident from Figure 1, is the robustness of the GAS–$\mathcal{ST}$ model to extreme observations compared with GAS–$\mathcal{N}$. Indeed, on April 11, 2008, General Electric reported an unexpected net income drop of 6%, which in turn translated to a fall of about 12% of its market value. The signal captured by GAS–$\mathcal{N}$ was that of an abrupt increase in volatility, with the consequence of large VaR level predictions. In contrast, the GAS–$\mathcal{ST}$ model slightly increased the volatility level and continued to predict reasonable VaR levels. What happened is that, GAS–$\mathcal{ST}$ treated the 12% negative return as a realization from the fat-tailed Student-t distribution, hence tapering its impact on the conditional volatility level. On the contrary, GAS–$\mathcal{N}$ treated the negative return as a realization from the Normal distribution, which is a clear signal of increase of volatility. The same behavior of the GAS–$\mathcal{N}$ specification is shown in the ES predictions (not reported to save space).

VaR backtesting

Let us now show how the accuracy of the VaR forecasts can be evaluated using the `BacktestVaR()` function in the **GAS** package. This function accepts the following arguments:

- `data`, `numeric` containing the out-of-sample data;
- `VaR`, `numeric` containing the series of VaR forecasts;
- `alpha`, the VaR risk level α;
- `Lags`, the number of lags used in the DQ test, by default `Lags = 4`; see Engle and Manganelli (2004).

The function returns a `list` with named entries:

- `LRuc`, the test statistic and associated p-value for the UC test of Kupiec (1995);
- `LRcc`, the test statistic and associated p-value for the CC test of Christoffersen (1998);
- `DQ`, the test statistic and associated p-value for the DQ test of Engle and Manganelli (2004);
- `Loss`, the quantile loss (QL), as defined in (2), together with the average QL used by González-Rivera et al. (2004);
- `AD`, the mean and max VaR Absolute Deviation (AD) used by McAleer and Da Veiga (2008);
- `AE`, the Actual over Expected ratio.

For instance, in order to compute the VaR backtest measures defined above on the forecast series `VaR_N`, we use:

```
> VaRBacktest_N <- BacktestVaR(data = tail(dji30ret[, "GE"], H), VaR = VaR_N, alpha = alpha)
```

Then, the DQ test statistic and its associated p-value can be extracted as:

```
> VaRBacktest_N$DQ
$stat
         [,1]
[1,] 52.47578

$pvalue
           [,1]
[1,] 4.7043e-09
```

which, in this case, is against the null of correct model specification for the 1% VaR level.

Now, if we evaluate VaR forecasts using the GAS–$\mathcal{ST}$ model, and save the results as the `VaRBacktest_ST` object, then the DQ test reports:

```
> VaRBacktest_ST$DQ
$stat
         [,1]
[1,] 8.763418

$pvalue
         [,1]
[1,] 0.270091
```

	$\alpha = 1\%$			$\alpha = 5\%$		
Asset	GAS–$\mathcal{N}$	GAS–$\mathcal{ST}$	GAS–$\mathcal{SKST}$	GAS–$\mathcal{N}$	GAS–$\mathcal{ST}$	GAS–$\mathcal{SKST}$
AA	0.00	0.17	0.16	0.00	0.00	0.00
AIG	0.00	0.00	0.00	0.00	0.00	0.00
AXP	0.09	0.99	0.25	0.19	0.13	0.15
BA	0.29	0.98	0.30	0.13	0.55	0.54
BAC	0.00	0.00	0.06	0.00	0.03	0.06
C	0.00	0.00	0.00	0.01	0.00	0.00
CAT	0.00	0.22	0.22	0.00	0.38	0.51
CVX	0.00	0.02	0.00	0.02	0.03	0.01
DD	0.03	0.41	0.34	0.35	0.31	0.13
DIS	0.00	0.05	0.08	0.00	0.35	0.15
GE	0.00	0.04	0.00	0.16	0.30	0.03
GM	0.00	0.06	0.02	0.02	0.29	0.17
HD	0.01	0.01	0.10	0.09	0.63	0.57
HPQ	0.00	0.01	0.03	0.00	0.11	0.35
IBM	0.06	0.03	0.03	0.00	0.04	0.05
INTC	0.04	0.07	0.33	0.06	0.10	0.06
JNJ	0.95	1.00	0.35	0.00	0.00	0.00
JPM	0.18	0.95	0.21	0.27	0.42	0.17
KO	0.00	0.00	0.00	0.00	0.00	0.00
MCD	0.11	0.68	0.73	0.10	0.17	0.40
MMM	0.00	0.34	0.00	0.00	0.18	0.33
MRK	0.00	0.05	0.06	0.00	0.11	0.05
MSFT	0.11	0.28	0.42	0.05	0.43	0.52
PFE	0.15	0.37	0.38	0.13	0.81	0.75
PG	0.00	0.18	0.07	0.13	0.11	0.06
T	0.35	0.37	0.42	0.02	0.01	0.06
UTX	0.38	0.41	0.07	0.09	0.90	0.93
VZ	0.04	0.99	0.96	0.88	0.70	0.83
WMT	0.00	0.00	0.01	0.93	0.54	0.26
XOM	0.00	0.00	0.00	0.07	0.04	0.17

Table 1: DQ test statistic p-values for the Dow Jones Industrial Average constituents one-step ahead VaR forecasts at the two downside risk levels $\alpha = 1\%$ and $\alpha = 5\%$. Under the null hypothesis, we have correct model specification for the chosen risk level. Light gray cells indicate p-values lower than 1%. The forecasting period ranges from February 14, 2005, to February 3, 2009, for a total of $H = 1{,}000$ out-of-sample observations. The model parameters are re-estimated at the monthly frequency.

The large p-value indicates that the null of the correct model specification for the 1% VaR cannot be rejected at the usual levels of significance for the GAS–$\mathcal{ST}$ model.

We reproduce this analysis for each of the thirty daily stock return series in the `dji30ret` data set previously detailed. The out-of-sample period starts on February 14, 2005, and includes the recent Global Financial Crisis of 2007-2008. We consider two VaR risk levels: $\alpha = 1\%$ and $\alpha = 5\%$. The code used for this application is available in the GitHub **GAS** repository: `https://github.com/LeopoldoCatania/GAS/wiki`.

Table 1 reports the p-values of the DQ test for the three model specifications and the two VaR risk levels. Under the null hypothesis, we have correct model specification for the α-quantile level. Clearly, our results indicate that GAS–$\mathcal{N}$ is suboptimal with respect to GAS–$\mathcal{ST}$ and GAS–$\mathcal{SKST}$ in terms of correct unconditional and conditional coverage. This result is somehow expected since GAS–$\mathcal{ST}$ and GAS–$\mathcal{SKST}$ exhibit excess kurtosis and deliver more robust updates for the volatility parameter than GAS–$\mathcal{N}$. Moreover, we see that GAS–$\mathcal{ST}$ and GAS–$\mathcal{SKST}$ perform similarly in terms of correct unconditional and conditional coverage. Hence, the inclusion of skewness does not seem to increase the performance of VaR predictions for the considered series. Indeed, sometimes results are even worse for GAS–$\mathcal{SKST}$ compared with GAS–$\mathcal{ST}$, indicating that the estimation error for the additional skewness parameter could worsen VaR predictions. This is the case for example for 3M Company (MMM) when GAS–$\mathcal{SKST}$ rejects the null for $\alpha = 1\%$, whereas GAS–$\mathcal{ST}$ does not.

VaR model comparison

The goal of VaR model comparison is to rank the models in terms of accuracy of their VaR forecasts. This can be done using the average value of the quantile loss in (2), as available in the output from the `BacktestVaR()` function.

In our application on the daily stock returns of General Electric, we find that the model comparison in terms of average QL also favors GAS–$\mathcal{ST}$ compared with GAS–$\mathcal{N}$. Indeed the ratio between the QL of GAS–$\mathcal{ST}$ and GAS–$\mathcal{N}$ is below unity indicating a lower average quantile loss when using the GAS–$\mathcal{ST}$:

```
> round(VaRBacktest_ST$Loss$Loss / VaRBacktest_N$Loss$Loss, 2)
[1] 0.94
```

This indicates that GAS–$\mathcal{ST}$ outperforms GAS–$\mathcal{N}$ by 6% in terms of average QL.

The left part of Table 2 (QL ratios) reports the results of repeating this model comparison analysis for all thirty daily stock return series in the `dji30ret` dataset.[4] It shows, for both $\alpha = 1\%$ and $\alpha = 5\%$, the ratios between the average QL of the considered models over the one delivered by GAS–$\mathcal{N}$. Values greater than one indicate outperformance of GAS–$\mathcal{N}$, and vice versa. Consistently with the DQ test results, we find that GAS–$\mathcal{ST}$ and GAS–$\mathcal{SKST}$ perform similarly and are preferred to GAS–$\mathcal{N}$.

	QL ratios				FZL ratios			
	$\alpha = 1\%$		$\alpha = 5\%$		$\alpha = 1\%$		$\alpha = 5\%$	
Asset	GAS–$\mathcal{ST}$	GAS–$\mathcal{SKST}$	GAS–$\mathcal{ST}$	GAS–$\mathcal{SKST}$	GAS–$\mathcal{ST}$	GAS–$\mathcal{SKST}$	GAS–$\mathcal{ST}$	GAS–$\mathcal{SKST}$
AA	0.93	0.93	0.98	0.98	0.95	0.95	0.98	0.98
AIG	1.00	1.01	1.02	1.03	0.86	0.87	0.93	0.94
AXP	0.93	0.94	0.98	0.99	0.94	0.96	0.98	1.00
BA	0.99	1.00	0.99	0.99	0.98	0.99	0.99	0.99
BAC	0.83	0.85	1.01	1.01	0.88	0.88	0.98	0.98
C	0.97	1.00	1.02	1.03	0.94	0.95	0.98	0.99
CAT	0.82	0.83	0.91	0.91	0.83	0.84	0.91	0.91
CVX	1.00	1.18	1.01	1.06	0.99	1.20	1.01	1.09
DD	0.94	0.94	0.98	0.98	0.95	0.97	0.98	0.98
DIS	0.98	0.99	1.00	1.00	0.95	0.96	0.99	1.00
GE	0.95	0.96	0.97	0.98	0.96	0.99	0.98	0.98
GM	0.87	0.88	0.93	0.94	0.91	0.93	0.96	0.97
HD	1.02	1.00	1.00	1.00	1.01	1.00	1.00	1.00
HPQ	0.95	0.95	0.94	0.94	0.95	0.95	0.95	0.95
IBM	1.00	1.00	0.94	0.94	0.92	0.93	0.93	0.93
INTC	0.97	0.97	0.96	0.96	0.94	0.94	0.96	0.96
JNJ	1.03	1.01	1.02	1.00	1.00	0.99	1.01	0.99
JPM	0.92	0.92	0.98	0.99	0.94	0.95	0.99	0.99
KO	0.97	0.96	0.96	0.96	0.97	0.96	0.93	0.92
MCD	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.99
MMM	0.82	0.83	0.90	0.90	0.80	0.82	0.89	0.89
MRK	0.83	0.84	0.89	0.89	0.81	0.82	0.88	0.89
MSFT	0.86	0.87	0.90	0.90	0.87	0.90	0.91	0.91
PFE	0.83	0.84	0.91	0.91	0.83	0.83	0.90	0.91
PG	0.86	0.87	0.95	0.95	0.82	0.85	0.92	0.93
T	0.95	0.97	1.00	1.00	0.97	0.98	0.99	0.99
UTX	0.93	0.94	0.94	0.94	0.93	0.94	0.92	0.92
VZ	0.91	0.91	0.98	0.98	0.93	0.94	0.97	0.97
WMT	0.97	0.97	0.99	1.00	0.96	0.97	0.99	1.00
XOM	1.02	0.96	1.02	1.01	0.97	0.92	1.00	0.99

Table 2: QL ratios (left part) and FZL ratios (right part) for GAS–$\mathcal{ST}$ and GAS–$\mathcal{SKST}$ with respect to GAS–$\mathcal{N}$ for the two risk levels $\alpha = 1\%$ and $\alpha = 5\%$. Values greater than one indicate outperformance of GAS–$\mathcal{N}$ and vice versa. Light gray cells indicate ratios higher than one (note that values are rounded). The forecasting period ranges from February 14, 2005, to February 3, 2009, for a total of $H = 1{,}000$ out-of-sample observations. The model parameters are re-estimated at the monthly frequency.

Joint VaR and ES model comparison

The `FZLoss` function implements the FZ loss and accepts the following arguments:

[4]Here we re-estimate the model parameters at the monthly frequency (*i.e.*, every 21 new observations).

- `data`: vector of observations;
- `VaR`: vector of VaR predictions;
- `ES`: vector of ES predictions;
- `alpha`: the α risk level.

This function returns a `numeric` vector of the same size of `data` with the FZ losses computed at each point in time. For instance, using the `VaR_N` and `ES_N` vectors previously computed we can evaluate the associated FZ loss as:

```
> FZL <- FZLoss(data = tail(dji30ret[, "GE"], H), VaR = VaR_N, ES = ES_N,
                alpha = alpha)
```

where `FZL` is a `numeric` vector of length `H`.

The right part of Table 2 (FZL ratios) reports the results of performing model comparison analysis for all thirty daily stock return series in the `dji30ret` dataset according to the FZ loss reported in (3). It shows, for both $\alpha = 1\%$ and $\alpha = 5\%$, the ratios between the average FZ loss of the considered models over the one delivered by GAS–$\mathcal{N}$. Values greater than one indicate outperformance of GAS–$\mathcal{N}$, and vice versa. Consistently with the comparison in terms of only VaR predictions, we find that GAS–$\mathcal{ST}$ and GAS–$\mathcal{SKST}$ perform similarly and are preferred to GAS–$\mathcal{N}$.

Conclusion

Under the regulation of the Basel Accords, risk managers of financial institutions need to rely on state-of-the-art methodologies for predicting and evaluating their downside risk (Board of Governors of the Federal Reserve Systems, 2012). This article illustrates the usefulness of the R package **GAS** in putting the theory of modern downside risk management into practice. The recommended strategy consists of four steps: (i) model specification, (ii) downside risk predictions, (iii) backtesting, and (iv) model comparison. We illustrate this proposed implementation in R using the package **GAS** applied to the Value-at-Risk and Expected Shortfall estimation for the daily returns of the thirty Dow Jones Industrial Average constituents.

Computational details

The results in this paper were obtained using R 3.5.0 with the package **GAS** version 0.2.8 available on CRAN at `https://cran.r-project.org/package=GAS`. Computations were performed on Windows 7 x64 (build 7601) Service Pack 1, x86_64-w64-mingw32/x64 (64-bit) with Intel(R) Xeon(R) CPU E5-2560 v3 2.30 GHz.

Acknowledgments

The authors thank the Editor-in-Chief, John Verzani, two Executive Editors, Roger Bivand and Norman Matloff, the anonymous reviewer, Alexios Ghalanos, participants at the R/Finance conference 2017 in Chicago, and participants at the session "Financial econometrics with R" at the 11th International Conference on Computational and Financial Econometrics 2017 in London. The authors acknowledge Google for financial support via the Google Summer of Code 2016 and 2017 project "GAS".

Appendix

To obtain the score of the $\mathcal{SKST}$ distribution, we first write its log density evaluated in r_t:

$$\log f_{\mathcal{SKST}}\left(r_t; \mu, \sigma_t, \xi, \nu\right) = \log g + \log k + c - \log \sigma_t - \frac{\nu+1}{2} \log \left[1 + \frac{\left[\left(\frac{r_t-\mu}{\sigma_t}\right) k + m\right]^2}{(\nu-2)\left(\xi_t^*\right)^2}\right],$$

where:

$$\begin{aligned}
m &\equiv \mu_1 \left(\xi - \frac{1}{\xi} \right) \\
k &\equiv \sqrt{(1-\mu_1^2)\left(\xi^2 + \frac{1}{\xi^2}\right) + 2\mu_1^2 - 1} \\
g &\equiv \frac{2}{\xi + \frac{1}{\xi}} \\
c &\equiv \frac{1}{2}\left[-\log(\nu-2) - \log\pi\right] + \log\Gamma\left(\frac{\nu+1}{2}\right) - \log\Gamma\left(\frac{\nu}{2}\right),
\end{aligned}$$

with:

$$\mu_1 \equiv \frac{2\sqrt{\nu-2}}{(\nu-1)} \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\Gamma(\frac{1}{2})},$$

and $\xi_t^* \equiv \xi^{I\{z_t \geq 0\} - I\{z_t \leq 0\}}$, where $I\{\cdot\}$ is the indicator function, and $z_t \equiv \left(\frac{r_t - \mu}{\sigma_t}\right) k + m$. Taking the partial derivative of the log density with respect to θ_t, yields the score s_t, given by:

$$s_t \equiv \frac{\partial \log f_{\mathcal{SKST}}}{\partial \theta_t} = \left(\frac{z_t\,(\nu+1)\,(z_t - m)}{(\xi_t^*)^2\,(\nu-2) + z_t^2} - 1 \right).$$

In the case where $\xi = 1$, we have $m = 0$, and the score becomes:

$$s_t = \left(\frac{z_t^2\,(\nu+1)}{(\nu-2) + z_t^2} - 1 \right),$$

which is the score of a Student-t distribution. Finally, when $\nu \to \infty$, we obtain:

$$s_t = z_t^2 - 1,$$

which is the score of the Normal distribution.

Bibliography

D. Ardia, K. Bluteau, K. Boudt, and L. Catania. Forecasting performance of Markov–switching GARCH models: A large–scale empirical study. *International Journal of Forecasting*, 34(4):733–747, 2018. URL https://doi.org/10.1016/j.ijforecast.2018.05.004. [p410, 413]

D. Ardia, K. Boudt, and L. Catania. Generalized autoregressive score models in R: The GAS package. *Journal of Statistical Software*, 88(6):1–28, 2019. URL https://doi.org/10.18637/jss.v088.i06. [p410, 413]

L. Bauwens and S. Laurent. A new class of multivariate skew densities, with application to generalized autoregressive conditional heteroscedasticity models. *Journal of Business & Economic Statistics*, 2005. URL https://doi.org/10.1198/073500104000000523. [p411]

F. Bellini and V. Bignozzi. On elicitable risk measures. *Quantitative Finance*, 15(5):725–733, 2015. URL https://doi.org/10.1080/14697688.2014.946955. [p412]

M. Bernardi and L. Catania. Comparison of Value–at–Risk models using the MCS approach. *Computational Statistics*, 31(2):579–608, 2016. URL https://doi.org/10.1007/s00180-016-0646-6. [p410]

F. Blasques, S. J. Koopman, and A. Lucas. Maximum likelihood estimation for correctly specified generalized autoregressive score models: Feedback effects, contraction conditions and asymptotic properties. Techreport TI 14-074/III, Tinbergen Institute, 2014. URL http://www.tinbergen.nl/discussionpaper/?paper=2332. [p411]

F. Blasques, S. J. Koopman, K. Łasak, and A. Lucas. In–sample confidence bands and out–of–sample forecast bands for time–varying parameters in observation–driven models. *International Journal of Forecasting*, 32(3):875–887, 2016. URL https://doi.org/10.1016/j.ijforecast.2015.11.018. [p411]

Board of Governors of the Federal Reserve Systems. 99th annual report. Technical report, Board of Governors of the Federal Reserve Systems, 2012. URL https://www.federalreserve.gov/publications/annual-report/files/2012-annual-report.pdf. [p418]

L. Catania, K. Boudt, and D. Ardia. *GAS: Generalized Autoregressive Score Models*, 2016. URL https://github.com/LeopoldoCatania/GAS. R package version 0.2.7. [p410]

P. F. Christoffersen. Evaluating interval forecasts. *International Economic Review*, 39(4):841–862, 1998. URL http://www.jstor.org/stable/2527341. [p412, 415]

D. Creal, S. J. Koopman, and A. Lucas. Generalized autoregressive score models with applications. *Journal of Applied Econometrics*, 28(5):777–795, 2013. URL https://doi.org/10.1002/jae.1279. [p410, 411]

R. F. Engle and S. Manganelli. CAViaR: Conditional autoregressive Value at Risk by regression quantiles. *Journal of Business & Economic Statistics*, 22(4):367–381, 2004. URL https://doi.org/10.1198/073500104000000370. [p412, 415]

C. Fernández and M. F. Steel. On Bayesian modeling of fat tails and skewness. *Journal of the American Statistical Association*, 93(441):359–371, 1998. URL https://doi.org/10.1080/01621459.1998.10474117. [p411]

T. Fissler and J. F. Ziegel. Higher order elicitability and Osband's principle. *The Annals of Statistics*, 44 (4):1680–1707, 2016. URL https://doi.org/10.1214/16-AOS1439. [p412]

C.-T. Gao and X.-H. Zhou. Forecasting VaR and ES using dynamic conditional score models and skew Student distribution. *Economic Modelling*, 53:216–223, 2016. URL https://doi.org/10.1016/j.econmod.2015.12.004. [p410]

G. González-Rivera, T.-H. Lee, and S. Mishra. Forecasting volatility: A reality check based on option pricing, utility function, Value–at–Risk, and predictive likelihood. *International Journal of Forecasting*, 20(4):629–645, 2004. URL https://doi.org/10.1016/j.ijforecast.2003.10.003. [p412, 415]

A. C. Harvey. *Dynamic Models for Volatility and Heavy Tails: With Applications to Financial and Economic Time Series*. Cambridge University Press, 2013. URL https://doi.org/10.1017/CBO9781139540933. [p410]

P. Jorion. *Value at Risk*. McGraw–Hill, New York, 1997. URL https://doi.org/10.1036/0071464956. [p410]

R. Koenker and G. Bassett. Regression quantiles. *Econometrica*, 46(1):33–50, 1978. URL https://doi.org/10.2307/1913643. [p412]

P. H. Kupiec. Techniques for verifying the accuracy of risk measurement models. *Journal of Derivatives*, 3(2):73–84, 1995. URL https://doi.org/10.3905/jod.1995.407942. [p412, 415]

M. Marcellino, J. H. Stock, and M. W. Watson. A comparison of direct and iterated multistep AR methods for forecasting macroeconomic time series. *Journal of Econometrics*, 135(1):499–526, 2006. URL https://doi.org/10.1016/j.jeconom.2005.07.020. [p411]

M. McAleer and B. Da Veiga. Single–index and portfolio models for forecasting Value–at–Risk thresholds. *Journal of Forecasting*, 27(3):217–235, 2008. URL https://doi.org/10.1002/for.1054. [p412, 415]

A. J. McNeil, R. Frey, and P. Embrechts. *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton university press, Princeton, 2015. URL https://doi.org/10.1111/jtsa.12177. [p411]

B. Narasimhan and S. G. Johnson. *cubature: Adaptive Multivariate Integration over Hypercubes*, 2017. URL https://CRAN.R-project.org/package=cubature. R package version 1.3-11. [p415]

M. R. Nieto and E. Ruiz. Frontiers in VaR forecasting and backtesting. *International Journal of Forecasting*, 32(2):475–501, 2016. URL https://doi.org/10.1016/j.ijforecast.2015.08.003. [p410]

A. J. Patton, J. F. Ziegel, and R. Chen. Dynamic semiparametric models for expected shortfall, 2017. URL https://doi.org/10.2139/ssrn.3000465. Working paper. [p413]

J. F. Ziegel. Coherence and elicitability. *Mathematical Finance*, 26(4):901–918, 2016. URL https://doi.org/10.1111/mafi.12080. [p412]

David Ardia
Institute of Financial Analysis
University of Neuchâtel, Switzerland
and
Department of Decision Sciences
HEC Montréal, Canada
ORCiD 0000-0003-2823-782X
david.ardia@unine.ch

Kris Boudt
Department of Economics
Ghent University, Belgium
and
Vrije Universiteit Brussel, Belgium
and
Vrije Universiteit Amsterdam, The Netherlands
ORCiD 0000-0002-1000-5142
kris.boudt@vub.be

Leopoldo Catania (corresponding author)
Department of Economics and Business Economics and CREATES
School of Business and Social Sciences
Aarhus University, Denmark
ORCiD 0000-0002-0981-1921
leopoldo.catania@econ.au.dk

NetworkToolbox: Methods and Measures for Brain, Cognitive, and Psychometric Network Analysis in R

by Alexander P. Christensen

Abstract This article introduces the **NetworkToolbox** package for R. Network analysis offers an intuitive perspective on complex phenomena via models depicted by nodes (variables) and edges (correlations). The ability of networks to model complexity has made them the standard approach for modeling the intricate interactions in the brain. Similarly, networks have become an increasingly attractive model for studying the complexity of psychological and psychopathological phenomena. **NetworkToolbox** aims to provide researchers with state-of-the-art methods and measures for estimating and analyzing brain, cognitive, and psychometric networks. In this article, I introduce **NetworkToolbox** and provide a tutorial for applying some the package's functions to personality data.

Introduction

Open science is ushering in a new era of psychology where multi-site collaborations are common and big data are readily available. Often, in these data, noise is mixed in with relevant information. Thus, researchers are faced with a challenge: deciphering information from the noise and maintaining the inherent structure of the data while reducing its complexity. Other areas of science have tackled this challenge with the use of network science and graph theory methods. Psychology is beginning to follow suit, changing the way we conceptualize psychological (Cramer et al., 2012; Schmittmann et al., 2013) and psychopathological phenomena (Borsboom and Cramer, 2013; Borsboom, 2017).

Psychological and psychopathological constructs are intrinsically complex. Over the years, researchers have focused on reducing phenomena to only the most significant variables. This reductionist approach has given us a greater understanding of what variables matter and what variables don't. Nonetheless, the more we reduce, the less we know about the phenomena as a whole (Barabási, 2011). With network science, we can begin to put psychological phenomena back together again, embracing the complexity.

Here, I introduce **NetworkToolbox**, a package for R (Team, 2018), available on the Comprehensive R Archive Network (`https://cran.r-project.org/web/packages/NetworkToolbox/index.html`). **NetworkToolbox** offers various network science methods and measures, which can be applied to brain, cognitive, psychometric, and other data suitable for network analysis. Networks are made up of nodes (e.g., circles) that are connected by edges (e.g., lines) to other nodes. Nodes represent variables (e.g., brain regions, words, questionnaire items) and edges represent a relation between two nodes (e.g., correlations, partial correlations), which can be undirected (i.e., bidirectional) or directed, and weighted or unweighted (all weights are equivalent). Thus, networks are conceptually simple yet capable of considerable complexity. In addition, their graphical depictions offer representations that have intuitive interpretations (Epskamp et al., 2012).

There are several network analysis packages that are already implemented in R, such as **statnet** (Goodreau et al., 2008), **igraph** (Csardi and Nepusz, 2006), and **sna** (Butts and others, 2008). Indeed, there are R packages that specifically focus on brain (e.g., **brainGraph**; Watson, 2017) and psychometric (e.g., **qgraph**, Epskamp et al., 2012; **IsingFit**, van Borkulo et al., 2014) networks. **NetworkToolbox** differentiates itself by including state-of-the-art network methods and measures that are not currently available in these other packages. Notably, network visualization aspects are not considered in this package. Where necessary, I refer readers to visualization sources that are compatible with **NetworkToolbox**'s outputs. In general, the R package **qgraph** (Epskamp et al., 2012) is recommended for network visualizations.

Psychometric network analysis

In this section, I provide explanations for many methods and measures in **NetworkToolbox** and include associated code implemented in R. The main body of **NetworkToolbox** is devoted to psychometric network analysis; however, several functions, such as network construction methods and network measures, could be applied more generally. To provide examples of functions in **NetworkToolbox**, I will use psychometric data but I will provide basic interpretations, so that measures can be generalized to other domains. All analyses conducted in this article were performed using R version 3.4.4 (2018-03-15) and **NetworkToolbox** version 1.2.1. Before exploring the methods and measures in **NetworkToolbox**, I will introduce the psychometric data that are used throughout the paper.

Data

The data included in **NetworkToolbox** are centered around the personality trait Openness to Experience scale of the NEO personality inventory (NEO-PI-3, McCrae et al., 2005; NEO-FFI-3, McCrae and Costa Jr, 2007). People high in Openness to Experience are described as *creative*, *curious*, *imaginative*, *unconventional*, and *intellectual*. The NEO Openness to Experience inventories have 6 facets: *actions*, *aesthetics*, *fantasy*, *feelings*, *ideas*, and *values*, representing distinct characterizations of the global trait. **NetworkToolbox** includes psychometric and brain data, which are openly available for researchers to use.

For the psychometric data, I will use data that includes four Openness to Experience inventories (Big Five Aspects Scale, DeYoung et al., 2007; HEXACO-100, Lee and Ashton, 2016; NEO-PI-3, McCrae et al., 2005; Woo et al.'s Openness to Experience Inventory, Woo et al., 2014), which was previously analyzed in Christensen et al. (2018a). For this article, the analyses of these data will be focused on the NEO-PI-3 (48 items). Data for all inventories can be found on the Open Science Framework (`https://osf.io/954a7/`). These data contain a relatively large sample ($N = 802$), which provides a foundation for some of the more nuanced analyses in **NetworkToolbox**. Below is code to load **NetworkToolbox** and the NEO-PI-3 data in the package:

```
# Load NetworkToolbox
library(NetworkToolbox)

# Load NEO-PI-3 data
data("neoOpen")
```

Network construction

A popular network construction method in network analysis is the *least absolute selection and shrinkage operator* (LASSO; Tibshirani, 1996). In short, the LASSO approach penalizes the inverse covariance matrix to produce sparse networks whose connections reflect conditional independence (i.e., nodes that are connected are uniquely associated, given all other nodes in the network). Because there are already a number of other R packages that compute LASSO networks (see **bootnet**, Epskamp et al., 2018; **glasso**, Friedman et al., 2014; **IsingFit**, van Borkulo et al., 2014), they are not included in **NetworkToolbox**. **NetworkToolbox** does include several other options for constructing a network.

The main network construction methods within the package are from the *Information Filtering Networks* approach (IFN; Barfuss et al., 2016). This approach constructs networks based on the zero-order correlations between variables and induces parsimony via a topological (structural) constraint. Currently, this family of networks has four methods: the *Triangulated Maximally Filtered Graph* (TMFG; Massara et al., 2016), the *Planar Maximally Filtered Graph* (PMFG; Tumminello et al., 2005), the *Maximum Spanning Tree* (MaST; Chu and Liu, 1965; Edmonds, 1968; Mantegna, 1999), and most recently, the *Pólya filter* (Marcaccioli and Livan, 2018). Additionally, the TMFG method can be associated with the inverse covariance matrix via the Local/Global inversion method (LoGo; Aste and Di Matteo, 2017; Barfuss et al., 2016) for probabilistic graphical modeling. For brevity, I will only introduce the TMFG and LoGo methods.

The `TMFG()` method constructs the network by first sorting the edge weights (i.e., correlations) in descending order. Next, the TMFG algorithm finds the four nodes that have the largest sum of edge weights with all other nodes in the network and connects them, forming a tetrahedron. Then, the algorithm iteratively identifies and adds the node that maximizes the sum of its connections to three of the nodes already included in the network. This process is completed once the network reaches $3n$ - 6 edges (n = nodes). The resulting network is composed of 3-node and 4-node cliques (i.e., triangles and tetrahedrons, respectively), which imposes a nested hierarchy and automatically generates a *chordal* network (for more details see Barfuss et al., 2016 and Song et al., 2012). Therefore, the TMFG method of network construction is a suitable choice for modeling psychological constructs like personality traits (Christensen et al., 2018a; McCrae, 2015) and psychopathological disorders (Kotov et al., 2017; Markon et al., 2005).

The chordal property of the TMFG network means that each 4-node clique (i.e., a diamond; a set of 4 connected nodes) in the network possesses a *chord*, which connects two of the nodes not already connected, forming two triangles. Chordal networks are thus composed of `separators`, which are the 3-node cliques that *separate* one 4-node clique from another 4-node clique. Chordal networks are a special class of networks that can represent the independence assumptions of Bayesian (directed) and Markovian (undirected) networks (Koller and Friedman, 2009) and have a simple association with the joint probability distribution (Barfuss et al., 2016; Massara et al., 2016). This association is expressed as:

$$Q(\mathbf{X}) = \frac{\prod_{C \in Cliques} \phi_C(\mathbf{X}_C)}{\prod_{S \in Separators} \phi_S(\mathbf{X}_S)}, \tag{1}$$

where Q($\mathbf{X}$) is the estimated joint probability distribution, and $\phi(_C)$ and $\phi(_S)$ are the marginal probabilities within the subsets of variables of the 4-node cliques ($\mathbf{X}_C$) and 3-node separators ($\mathbf{X}_S$), respectively. This association to the joint probability distribution can be implemented using the Local/Global inversion method (LoGo; Barfuss et al., 2016). Thus, the only difference between the TMFG and LoGo networks is the edge weights (zero-order correlations and partial correlations regressed over all others, respectively). In this way, the TMFG network embeds conditional independence within its structure (see Figure 1). An example of each function is provided below:

```
# Construct TMFG network
tmfg <- TMFG(neoOpen)$A

# Construct LoGo network
logo <- LoGo(neoOpen)
```

These networks can be visualized side-by-side (Figure 1) using **qgraph** and the following code:

```
# Load qgraph
library(qgraph)

# NEO-PI-3 defined facets
facets <- c(rep("actions", 8), rep("aesthetics", 8), rep("fantasy", 8),
rep("feelings", 8), rep("ideas", 8), rep("values", 8))

# Visualize TMFG
A <- qgraph(tmfg, groups = facets, palette = "ggplot2")
# Visualize LoGo
B <- qgraph(logo, groups = facets, palette = "ggplot2")

# Visualize TMFG and LoGo side-by-side
layout(t(1:2))
Layout <- averageLayout(A, B)
qgraph(A, layout = Layout, esize = 20, title = "TMFG")
qgraph(B, layout = Layout, esize = 20, title = "LoGo")
```

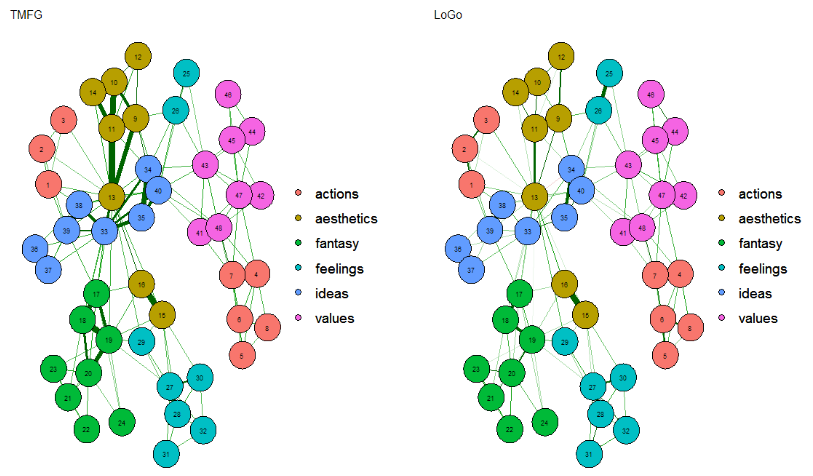


Figure 1: Plot of NEO-PI-3's items using the TMFG (left) and LoGo (right) network construction method

Network Construction Outputs Across all network construction methods, an adjacency matrix (or network) is output from the function. For most methods, this is the only output from the function.

Method	Output
`MaST()` (Chu and Liu, 1965; Edmonds, 1968)	adjacency matrix
`TMFG()` (Massara et al., 2016)	adjacency matrix (`A`) separators (`separators`) cliques (`cliques`)
`LoGo()` (Barfuss et al., 2016)	adjacency matrix
`ECO()` (Fallani et al., 2017)	adjacency matrix
`ECOplusMaST()` (Fallani et al., 2017)	adjacency matrix
`threshold()`	adjacency matrix (`A`) correlation critical value (`r.cv`)

Table 1: Network construction methods and outputs

For others, like `TMFG()`, a list containing the adjacency matrix (`A`) and other unique output from the function is supplied. Table 1 provides the output of each network construction method.

The unique arguments of TMFG pertain to the `cliques` and `separators`, which are used to associate the TMFG network with the inverse covariance matrix via `LoGo()`. These outputs are a wrapper provided for `LoGo()` and are usually unimportant to the researcher. `threshold()` is the other network construction method with a unique output, `r.cv`, which provides the correlation value that was used to threshold values in the network.

Common Network Construction Arguments In general, there are several common arguments that are applied in each network construction method. Additionally, there are some arguments that are specific to a single network construction method. For all network construction methods in **NetworkToolbox**, there are a few arguments designed to handle the content and structure of the data. Table 2 displays these arguments with the descriptions of their options.

Argument	Options	Description
`data`	`data`	input data or correlation matrix
`na.data`	`"listwise"`	removes any row with missing data
	`"pairwise"`	uses available data for given variable
	`"fiml"`	uses all information available
`normal`	`TRUE`	uses `cor_auto` from **qgraph**
	`FALSE`	uses Pearson's correlation

Table 2: Network construction arguments

The first argument is `data`, which can either be data or a correlation matrix. If input is data, then it should only include the variables of interest, without any identification, demographic, or descriptive variables (unless, of course, these are of interest). The second argument is `na.data`, which handles missing data. There are three options for `na.data`: `"listwise"`, `"pairwise"`, and `"fiml"`. The coarsest option is `"listwise"` deletion, which uses R's base function `na.omit()` to remove any case that has a missing observation. When using this argument, the researcher will automatically be notified exactly which rows were removed from network's construction. An example of this output is provided below:

```
Warning message: In TMFG(neoOpen, na.data = "listwise") :
10 rows were removed for missing data row(s):
234, 497, 71, 639, 99, 677, 652, 255, 150, 28
```

The `"pairwise"` deletion will use the data that is available for each variable when constructing the network. Note that relations from pairwise deletion could potentially be based on different subsets of cases, which could be problematic. By far, the best and recommended option is `"fiml"` or Full Information Maximum Likelihood (FIML), which is implemented by the **psych** package (Revelle, 2017). FIML uses all information available to produce a deterministic correlation matrix (i.e., the same result every time). For this reason, FIML has been the standard missing data approach in structural equation modeling (SEM) software. Notably, network construction does take a few seconds longer when using the `"fiml"` option.

The last argument, `normal`, handles whether correlations should be used that are transformed to relations that are multivariate normal. There are multiple R packages that offer multivariate normal testing like **MVN** (Korkmaz et al., 2014), so such functions are not included in **NetworkToolbox**. Multivariate normal data are of particular importance in network and graphical modeling because there is a direct relationship between the inverse covariance matrix (also known as the *precision* matrix) and the partial correlation (conditional independence) structure. Thus, conditional independence networks fundamentally rely on a multivariate normal distribution. The necessity of this assumption and the influence of deviating from multivariate normal in network estimation are worthy of future investigation (e.g., Loh and Wainwright, 2013).

When `normal` is set to `TRUE`, then the data are correlated using **qgraph**'s `cor_auto()`. This function detects ordinal variables and uses polyserial, polychoric, and Pearson's correlations. Other correlations (e.g., tetrachoric) and association measures can also be used but they must be computed using some other function (e.g., **psych**'s `tetrachoric()`; Revelle, 2017) before they are used as input into any of the network construction functions. When `normal` is set to `FALSE`, Pearson's correlations are computed. In all network construction methods in **NetworkToolbox**, `normal` defaults to `FALSE`. It is important to note that despite the notion that Pearson's correlation assumes normality, it does not; however, the test statistic, on which significance is based, may have a different distribution when the data is non-normal. In addition, Pearson's correlation is sensitive to outliers and highly skewed variables, so data transformations or alternative correlation measures (e.g., Spearman's rho) should be considered when normality is violated.

Dependency Network Analysis One argument that is specific to the `TMFG()` function is `depend`, which is used to construct a dependency network from data that has been run through the function `depend()`. The dependency network approach was introduced by Kenett et al. (2010) and produces edges in networks that show the direction of influence from one node to another (i.e., a directional network)—similar to relative importance networks (Grömping and others, 2006; Johnson and LeBreton, 2004; Robinaugh et al., 2014). This approach makes use of first-order partial correlations, which are the partial (or residual) effect of a given node *j* on the correlation between another pair of nodes *i* and *k*. The resulting partial correlation between nodes *i* and *k* is the correlation after the subtraction of the correlations between nodes *i* and *j* and between nodes *k* and *j*. In this way, the partial correlation provides a measure of node *j*'s influence on the correlation between nodes *i* and *k*. Therefore, the influence of node *j* on node *i* can be determined by taking the average (or the sum) influence of node *j* on the correlations of node *i* with all other nodes. Mathematically, this can be expressed as:

$$d(i,k|j) \equiv C(i,k) - PC(i,k|j) \tag{2}$$

where $C(i,k)$ is the correlation between node *i* and *k* and $PC(i,k\,|\,j)$ is the partial correlation of node *i* and *k* given *j*. By subtracting the partial correlation of nodes *i* and *k* given node *j* from the correlation of nodes *i* and *k*, the infrequent case of where node *j* appears to strongly affect the correlation $C(i,k)$ when $C(i,j)$, and $C(k,j)$ have small values is avoided. In order to determine the influence of node *j* on node *i*, the average influence of node *j* is defined across all relative effects of the correlations where node *k* is not node *j*:

$$D(i,j) = \frac{1}{N-1}\sum_{k\neq j}^{N-1} d(i,k|j) \tag{3}$$

As defined, the dependency matrix *D* has (i,j) elements, which are the dependency of node *i* on node *j*. Put differently, this measure specifies the average influence of node *j* on node *i* ($j \to i$). In a dependency network, the arrow pointing from node *j* to node *i* signifies that node *j* influences node *i*. Although the correlation matrix is symmetric, the dependency matrix is asymmetric, which means the influence of node *j* on node *i*, $D(j,i)$, is not equal to the influence of node *i* on node *j*, $D(i,j)$. It's important to note that these directional influences are not interpreted as causal relations but instead as predictive relationships. These networks are, however, a step towards inferring causal relations between two nodes and may function as a "pre-test" for data and methods that can infer causal relations (Jacob et al., 2016). Therefore, the advantage of the dependency network approach is that the mutual influence between two nodes is not assumed to be equal but rather one node is assumed to have more influence in the relationship than the other.

The TMFG algorithm was adapted, in accordance with Kenett, Tumminello, Madi, Gur-Gershgoren, Mantegna, and Ben-Jacob (2010), to handle the asymmetric relations that are produced by the dependency network approach. Kenett et al.'s (2010) adaption sorts the weights of $D(i,j)$ in a descending order, and whichever link of $D(i,j)$ or $D(j,i)$ is greater is retained. This is done to keep information about the main direction of influence and to avoid multiple links (although the *actual* direction of influence could be both ways). Below is an example of how to apply the dependency network approach and the network it produces (Figure 2):

```
# Dependency matrix
depmat <- depend(neoOpen)

# Construct dependency TMFG network
deptmfg <- TMFG(depmat, depend = TRUE)$A

# Visualize dependency TMFG network
layout(c(1, 1))
qgraph(deptmfg, layout="spring", groups = facets, palette = "ggplot2",
label.prop = 1, vsize = 4, title = "Dependency TMFG")
```

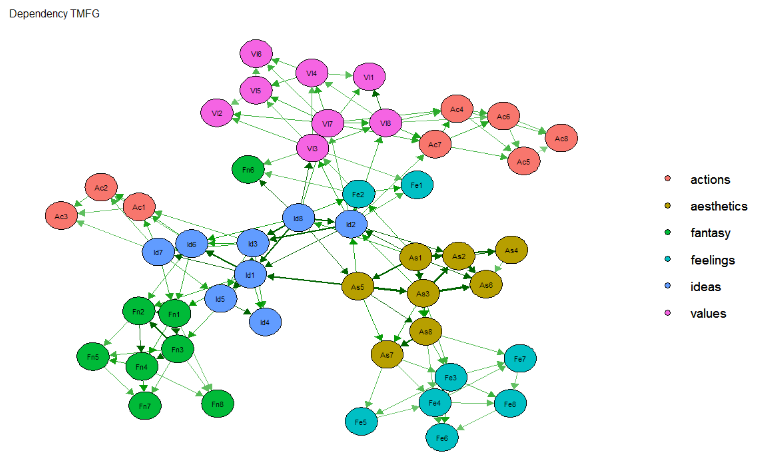


Figure 2: Plot of NEO-PI-3's items using the dependency TMFG network construction method

Network measures

After network construction, network measures can be applied to quantify the information in the network. In **NetworkToolbox**, there are a several network measures available. First, I will start with local network characteristics, which have received the most attention in the psychometric network literature.

Local network characteristics The local network characteristics of a network describe how each node contributes to the overall network. Node centrality measures are the most common way to measure local network characteristics. Node centrality measures quantify each node's influence in the network. Some standard measures that are used are betweenness centrality (BC; `betweenness()`; Freeman, 1977), closeness centrality (LC; `closeness()`), degree (k; `degree()`), and node strength (NS; `strength()`). Because these measures are frequently used in the network science literature, I will assume the reader has encountered these measures before.

Randomized shortest paths betweenness centrality One measure that closely resembles the standard centrality measures is the randomized shortest paths betweenness centrality (RSPBC; Kivimäki et al., 2016). The difference between the standard BC and the RSPBC is the computation of the shortest paths—the smallest number of steps to get from one node to another—in the network. The standard BC measures how often a node is on the absolute shortest path (i.e., the most direct route) from one node to another. In contrast, the RSPBC measures how often a node is on the random shortest path (i.e., random "jumps") from one node to another. The randomness of the jumps are adjusted using the Bolztmann probability distribution, which can be manipulated with the argument `beta` (Kivimäki et al., 2016):

```
rspbc(A, beta = 0.01)
```

Higher `beta` values will decrease the randomness of the paths with `beta` = 10 being closer to the standard BC; lower `beta` values will increase the randomness of the paths with `beta` = .0001 being closer to degree. The beta argument defaults to .01, which is based on Kivimäki et al.'s (2016) recommendation. In **NetworkToolbox**, the RSPBC is adjusted so that the lowest value is one (the node is included in its own shortest path) and the rest of the values are relative to this point (i.e., the minimum RSPBC value - 1 is subtracted from each node's RSPBC value).

The improvement of the RSPBC over the standard BC is its distribution of values. Often, the standard BC has a skewed distribution where few nodes have extremely large values and most nodes have low values or values of zero. The RSPBC also has many large values but there is a greater gradation of values across the nodes, and nodes that are clearly "between" other nodes are not given a value of zero (as they would with the standard BC). The node 41 in Figure 1, for example, has a standard BC log value of 0 but a RSPBC log value of 6.62. In comparison, node 6 in Figure 1 has a larger log value for standard BC (1.10) but a smaller log value for RSPBC (6.34). To demonstrate these differences, a plot is sohwn in Figure 3.

```
# Randomized shortest paths betweenness centrality
bc <- betweenness(tmfg)
rbc <- rspbc(tmfg, beta = .01)

# Plot for comparison
plot(cbind(log(bc+1),log( rbc+1)), xlab = "Standard BC (log)",
ylab = "RSPBC (log)", main = "RSPBC on Standard BC", pch = 16)
text(6, 2, labels = paste("r = ", round(cor(bc, rbc, method = "kendall"), 2)))
```

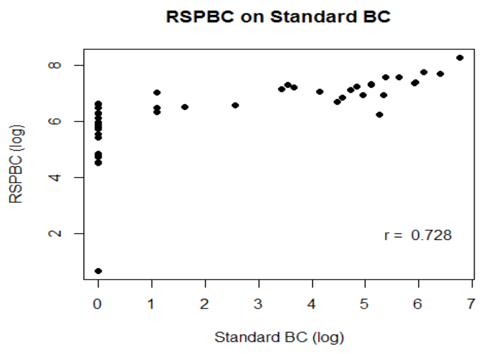


Figure 3: Plot of log-transformed RSPBC on log-transformed Standard BC

The rank-order comparison (*r* = .73) suggests that the measures are related but provide slightly different information. Indeed, the rank-order of the nodes is generally the same, with most of the differentiation being provided by the greater gradation of RSPBC values for nodes with small standard BC values. Notably, because RSPBC is continuous, rather than an all-or-nothing measure (like the standard BC), it's likely that it has greater reliability than the standard measure. The issue of standard BC's low stability is well known in the psychometric network literature (Epskamp et al., 2018). Therefore, the RSPBC stands as a reasonable alternative for researchers interested in measuring the betweenness centrality of their network. Moreover, the RSPBC's interpretation also provides a more reasonable explanation of a node's centrality influence in the network. The standard BC, for example, suggests that only the nodes on the absolute shortest path are influential in the spread of activation (i.e., the most central nodes' effect on other nodes). In contrast, the RSPBC suggests that this influence is mainly spread through the most central nodes but other avenues, that aren't the absolute most direct route but are still a short path to other nodes, could also potentially activate (or effect) other nodes.

Hybrid centrality Another local network characteristic is an overall measure of centrality. The hybrid() function computes hybrid centrality (HC), which is not a specific centrality measure but rather a composite of BC, LC, NS, *k*, and the eigenvector centrality (EC; Christensen et al., 2018b; Pozzi et al., 2013). Pozzi et al. (2013) used this measure to assess risk associated with core and peripheral stocks in the stock market, finding more peripheral stocks were less risky to invest in. Christensen et al. (2018b) used this measure to assess the overall centrality of items in a self-report schizotypy—sub-clinical characteristics of schizophrenia—inventory. They found that more central items in the network were related, above and beyond intermediate and peripheral items, to interview-rated schizophrenia-spectrum symptoms.

The HC rank orders each centrality measure (lower values receiving a higher rank) using weighted and unweighted networks. Then, the ranks are added together and subtracted by the number of centrality measures used (e.g., eight), forming the numerator. The denominator is the number of centrality measures times the number of nodes minus one. The mathematical expression is provided below:

$$HC = \frac{BC^w + BC^u + LC^w + BC^u + k^u + NS^w + EC^w + EC^u - 8}{8 \times (N-1)}, \tag{4}$$

where w signifies the weighted measure and u signifies the unweighted measure. Note that degree (k) and node strength (NS) are only included once because the weighted degree *is* node strength. Other than the adjacency matrix (`A`), the HC includes one argument for how `BC` should be computed. The options for `BC` include `"standard"` for standard BC and `"random"` for RSPBC, with the default being `"standard"`.

Node impact One final local network characteristic is node `impact()`, which measures a node's influence on the structure of the network. To be more precise, node impact quantifies the difference between the network's average shortest path length (ASPL; the mean of the shortest number of edges for all nodes to get to other nodes in the network) when the node is removed from the network minus the ASPL when all nodes are included in the network (Kenett et al., 2011, 2013). Positive values suggest that removal of the node increases the ASPL of the network; negative values suggest that removal of the node decreases the ASPL of the network. Thus, a node with a positive impact *increases* the interconnectedness of the network when present in the network and a node with a negative impact *decreases* the interconnectedness of the network when present in the network. Other than centrality, node impact could be interpreted in terms of the community (i.e., clustered sets of connected nodes) structure of the network. A positive impact value means that when the node is removed, the distinctiveness (or "orthogonal-ness") of the communities in the network increases (Cotter et al., 2018). Conversely, a negative impact value means that when the node is removed, the distinctiveness of the communities in the network decreases.

Meso-scale Network Characteristics Meso-scale features of networks are becoming increasing popular in network analysis (Blanken et al., 2018; Giscard and Wilson, 2018; Golino and Epskamp, 2017; Golino and Demetriou, 2017; Golino et al., 2018). Meso-scale network characteristics evaluate the sub-network structure of the network or how the entire network can be summarized into smaller components (i.e., communities). These communities have been shown to be equivalent to factors or dimensions (Golino and Epskamp, 2017). The advantage of using networks to identify hierarchical structure in data is that they do not rely on the researcher's interpretation of scree plots, eigenvalues, or component loadings, which are used in more traditional dimension reduction methods (such as Principal Components Analysis; PCA). Instead, the hierarchical structure in networks is emergent—that is, based on the relations between variables, nodes cluster together and form distinct communities that arise from the data.

Community detection is the main measure of meso-scale network characteristics as well as the foundation for other meso-scale characteristics. Therefore, the selection of a community detection algorithm is of the utmost importance. One of the most commonly applied algorithms is the walktrap algorithm (Golino and Epskamp, 2017; Pons and Latapy, 2006) but it is by no means the only one (for a review see Fortunato, 2010). The current standard in the psychometric network literature is to apply Exploratory Graph Analysis (EGA), which implements the walktrap algorithm using **igraph**. Currently, this approach has two implementations, which are based on different network construction methods: the graphical LASSO Golino and Epskamp (2017); Golino and Demetriou (2017) and TMFG (Golino et al., 2018). Both implementations have yielded promising results in simulation and real-world data, demonstrating comparable or better accuracy than traditional methods of dimension reduction (e.g., PCA, parallel analysis, cluster analysis; Christensen et al., in press; Golino and Epskamp, 2017; Golino et al., 2018). Because these methods are already supplied in **EGA** (Golino, 2018), I will not cover them here. Instead, I present a complementary measure that can be used to examine the "centralness" of communities in the overall network.

Community closeness centrality The concept of this measure extends from node-wise closeness centrality, with ASPL of the nodes within each community being used rather than the nodes. Each node's local ASPL (ASPL$_i$, i.e., each individual node's ASPL) is computed for all nodes. Then, the average across these nodes is taken and the reciprocal is calculated. This is done for each community. The interpretation of this measure is akin to the closeness node centrality—that is, communities closer to the overall center of the network have a higher value. This measure provides an objective classification of how central specific communities are in the network. Thus, the relative importance of a community in the network can be obtained. In psychometric networks, this is related to scale-inventory correlations. Below, an example is provided to demonstrate the close relationship between these two measurements.

```
# Unique facets
uniq <- unique(facets)

# Initialize matrix
corr <- matrix(0, nrow = length(uniq), ncol = 2)

# Name rows
row.names(corr) <- uniq

# Name columns
colnames(corr) <- c("Scale2Inv", "CommCent")

# Compute scale-to-inventory correlations
for(i in 1:length(uniq))
{
    # Identify facet members
    target <- which(facets == uniq[i])
    corr[i, 1] <- cor(rowMeans(neoOpen[, -target]), rowMeans(neoOpen[, target]))
}

# Compute community impact
corr[,2] <- comm.close(tmfg, comm = facets)

# Compute correlation
round(cor(corr, method = "kendall"), 2)
```

The rank-order correlation (r = .73) suggests that the measures are closely related but might not be taken as strong enough evidence for measuring the same thing. One reason for this difference is that the network's communities are organized differently than the theoretical definitions. Three items of the *actions* facet, for example, are not connected to the other items of the *actions* facet (Figure 1). This issue can be resolved by applying EGA to determine the network's community structure. Below is code to first apply and compare EGA dimensions to the theoretical dimensions.

```
# Load EGA
library(EGA)

# Estimate dimensions
ega <- EGA(neoOpen, model = "TMFG")
ega.var <- as.character(ega$dim.variables$items)

# Order by item
ega.ord <- ega$dim.variables[match(colnames(neoOpen), ega.var), 2]

# Visualize theoretical factors
A <- qgraph(tmfg, groups = as.factor(facets), palette = "ggplot2")

# Visualize walktrap factors
B <- qgraph(tmfg, groups = as.factor(ega.ord), palette = "ggplot2")

# Compare theoretical and walktrap factors
layout(t(1:2))
Layout <- averageLayout(A,B)
qgraph(A, layout = Layout, esize = 20, title = "Theoretical")
qgraph(B, layout = Layout, esize = 20, title = "Walktrap")
```

Notably, although a number of the EGA-derived communities were equivalent to the number of theoretical facets, the items do not identically correspond between their designations. This does not suggest that the theoretical facets are incorrect nor does it suggest that the network-derived facets are incorrect. Network models are sample specific, which means a network with a different sample might demonstrate results more or less in line with the theoretical facets or the facets provided by EGA. In general, however, these results reproduce the theoretical facets defined by the NEO-PI-3. Next, code is provided to compute the similarity between the scale-to-inventory correlations with the EGA dimensions.

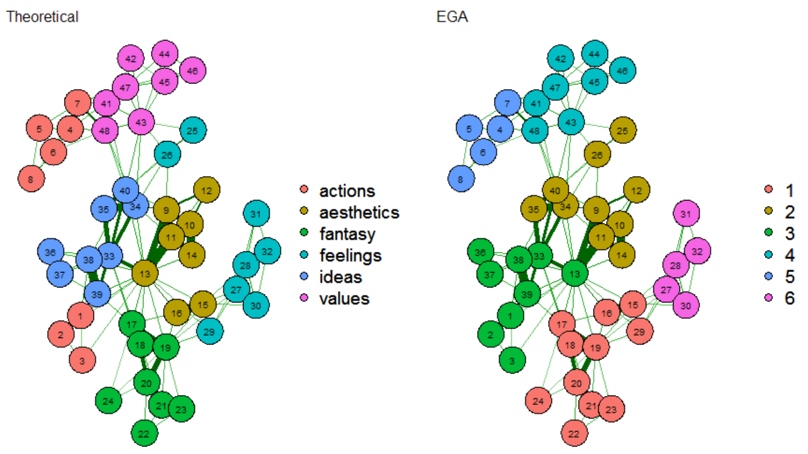


Figure 4: The theoretical facets are depicted on the left and the network-derived facets are depicted on the right

```
# Unique facets
uniq <- unique(ega.ord)

# Initialize matrix
corr <- matrix(0, nrow = length(uniq), ncol = 2)

# Name rows
row.names(corr) <- uniq

# Name columns
colnames(corr) <- c("Scale2Inv", "CommCent")

# Compute scale-to-inventory correlations
for(i in 1:length(uniq))
{
    # Identify facet members
    target <- which(ega.ord == uniq[i])
    corr[i, 1] <- cor(rowMeans(neoOpen[, -target]), rowMeans(neoOpen[, target]))
}

# Compute community closeness centrality
corr[, 2] <- comm.close(tmfg, comm = ega.ord)

# Compute correlation
round(cor(corr, method = "kendall"), 2)
```

With the EGA dimensions, there is a perfect rank-order correlation (r = 1) between the scale-to-inventory correlations and the community closeness centrality measure, suggesting there is a meaningful relationship between these two measures. In the context of networks more generally, this measure provides a quantitative approach to determining which communities are more central to the overall network.

Network adjusted means and sums

Network models offer an attractive alternative to latent variable models, such as factor analysis because they provide a data-driven approach for determining the structure of a construct. Networks also provide a way to go beyond sum scores by analyzing the smallest components of the overall construct (i.e., items and symptoms; Fried and Nesse, 2015). In addition, their emergent communities do not rely on constraints imposed by the researcher, which allows sample-specific expressions to emerge. To date, however, there has yet to be an approach developed to extract latent scores like traditional latent variable models (although network models have been combined with latent variable models; see Epskamp et al., 2017). Below, I introduce a function designed to extract network adjusted means and sums and propose a framework for establishing a latent network score. First, I briefly discuss the arguments and outputs of the function. After, I provide the conceptual framework for computing a network latent score, similar to latent scores calculated via confirmatory factor analysis (CFA). Finally, I demonstrate its effectiveness by comparing the network adjusted score to latent variable scores from a CFA model.

Arguments and output Below are the arguments for the network adjusted means and sums function, `nams`:

```
nams(data, A, comm = c("walktrap", "louvain"),
standardize = TRUE, adjusted = c("mean", "sum"), ...)
```

where `data` is an input for the data, which must be included in order to estimate the network adjusted mean or sum scores. The argument `A` is for an adjacency matrix (i.e., the network) associated with the data. Next, `comm` allows the researcher to define the communities of the network using a vector, which may be the output from a community detection algorithm or manually defined (e.g., theoretical communities). The default for `comm` is to treat the network as a single community. If a community detection algorithm (`"walktrap"` or `"louvain"`) is used, then `...` will handle additional arguments for the specified community detection algorithm. The argument `adjusted` is for whether the researcher would like the network adjusted mean or sum as the output (defaults to `"mean"`). Finally, the `standardize` argument is for whether the network adjusted scores should be standardized or not (defaults to `TRUE`). Note that when `standardize = TRUE`, the `adjusted` argument is ignored.

The function `nams()` outputs a list containing three objects: `NetAdjScore`, `FactorItems`, and `FactorCor`. `NetAdjScore` contains the network adjusted score, which is specified by the `adjusted` and `standardize` arguments. `FactorItems` displays a table of nodes corresponding to their respective factors (this is particularly useful when using a community detection algorithm). Finally, `FactorCor` provides the correlations between each community's network adjusted score including the overall network adjusted score.

Conceptual Framework The implementation of the this method requires a comprehensive measure of a node's influence in the network. As discussed above, the hybrid centrality is optimal for this task because it captures multiple centrality measures in a single overall centrality measure. The RSPBC is used for the betweenness centrality measure when computing the hybrid centrality in `nams()` because of its ability to differentiate lower values of betweenness (Figure 3).

Network as one community Using the NEO-PI-3 data as an example, the hybrid centrality of each node is used as a weight for each participant's response—that is, their response on 5-point Likert scale ranging from 1 (*Strongly Disagree*) to 5 (*Strongly Agree*). This is done by multiplying each participant's response to each variable by the hybrid centrality value for each variable. This process is repeated for each participant and variable in the network. The participant's network adjusted mean or sum can be computed by taking the average or sum across their weighted responses. Then, the participants average or total can be normalized by dividing by the mean of the hybrid centrality values. The formula for this metric is described below:

$$\theta_j | u_j = \frac{\sum_{i=1}^{n} h_i u_{ij}}{\frac{1}{n}\sum_{i=1}^{n} h_i}, \tag{5}$$

where i is a variable (i.e., a node), j is a case (e.g., a participant), u_{ij} is the response value for variable i and case j, n is the total number of nodes, h_i is the hybrid centrality value of node i, and $\theta_j \mid u_j$ is the latent score of participant j given the response pattern u_j. The stated equation provides the latent score estimate for the general construct that is being measured by the network. If the researcher is only interested in the construct associated with the entire network, then no additional computation is necessary.

Example Below I've provided code to compare the network's latent variable scores to the scores from a CFA latent variable model. For comparison, The network's latent variable scores are plotted on the CFA latent variable scores as well as the participant's mean scores on the CFA latent variable scores. First, the theoretical CFA model must be built and the latent variable scores extracted. To do so, the **lavaan** (Rosseel, 2012) package was used.

```
# Load lavaan
library(lavaan)

# Build NEO model
neo.model <- 'actions =~ Act1 + Act2 + Act3 + Act4 + Act5 + Act6 + Act7 + Act8
```

```
aesthetics =~ Aes1 + Aes2 + Aes3 + Aes4 + Aes5 + Aes6 + Aes7 + Aes8
fantasy =~ Fan1 + Fan2 + Fan3 + Fan4 + Fan5 + Fan6 + Fan7 + Fan8
feelings =~ Fee1 + Fee2 + Fee3 + Fee4 + Fee5 + Fee6 + Fee7 + Fee8
ideas =~ Ide1 + Ide2 + Ide3 + Ide4 + Ide5 + Ide6 + Ide7 + Ide8
values =~ Val1 + Val2 + Val3 + Val4 + Val5 + Val6 + Val7 + Val8
open =~ actions + aesthetics + fantasy + feelings + ideas + values'

# Fit CFA model
fit <- cfa(neo.model, data = neoOpen, estimator = "WLSMV")

# Compute latent variable scores
cfaScores <- lavPredict(fit)
cfaLV <- scale(cfaScores[, 7])
```

Then, the network's overall latent variable scores must be extracted using `nams()`:

```
# Compute network latent variable scores
netScores <- nams(neoOpen, tmfg, comm = facets)
netLV <- netScores$Standardized$overall
```

Finally, the plots can be generated:

```
# Plot network latent variable on CFA latent variable
layout(t(c(1, 2)))
plot(cfaLV, netLV,
     main = "Network Latent Variable\non CFA Latent Variable",
     ylab = "Network Latent Variable",
     xlab = "CFA Latent Variable")

# Compute correlation
netCor <- cor(cfaLV, netLV)

# Compute root mean square error
netRoot <- rmse(cfaLV, netLV)

# Add text to plot
text(-2, 2, labels = paste("r = ", round(netCor, 2),
                           "\nrmse = ", round(netRoot, 3)))

# Compute standardized participant means
pmeans <- scale(rowMeans(neoOpen))

# Plot participant means on CFA latent variable
plot(cfaLV, pmeans,
     main = "Participant Means on\nCFA Latent Variable",
     ylab = "Participant Means",
     xlab = "CFA Latent Variable")

# Compute correlation
meansCor <- cor(cfaLV, pmeans)

# Compute root mean square error
meansRoot <- rmse(cfaLV, pmeans)

# Add text to plot
text(-2, 2, labels = paste("r = ", round(meansCor, 2),
                           "\nrmse = ", round(meansRoot, 3)))
```

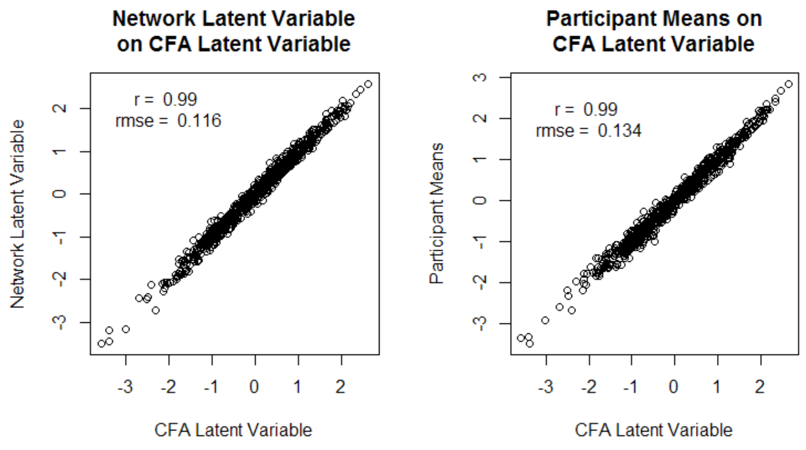


Figure 5: Plot of the standardized network latent scores on the standardized CFA latent scores (left) and the standardized participant's means on the standardized CFA latent scores (right)

In general, the network's latent scores are closer to the CFA's latent scores than the participant mean scores. The correlations, however, were high for both the network latent scores and the participant means. Figure 5 and the root mean square error show that the network's latent scores (rmse = 0.116) are more closely related to the CFA latent scores than the participant means (rmse = 0.134). Notably, the relationship between the network latent scores and the CFA latent scores is not perfect, suggesting there are some differences between the two measures.

Network as many communities The network's latent community scores differ slightly from the global network scores (i.e., the network as one community). For the community scores, it's important to consider the relative position of each community in the network. In order to factor in the positions of communities in the network, the community closeness centrality is computed for each community. The community closeness centrality provides a weight for nodes that are in communities that are more central in the network. Thus, nodes with low hybrid centrality that are in more central communities still receive additional weight because of their overall positioning in the network. The sum of the community closeness and global hybrid centrality is then used as the weight that is multiplied across the participant's response to each item within that community. The sum or mean of these facet scores are computed and normalized using the respective average of the weights in the community. This process repeats until all communities are computed. Finally, the mean or sum may not average or total to the overall network latent variable mean or sum. To adjust for this, the difference between the facets mean or sum total and the overall network latent mean or sum is taken and proportionally distributed across the facets.

Example Because the CFA latent scores and network latent scores were already computed, they can be immediately examined with the following code:

```
# Network latent facet scores
netFacet <- matrix(0, nrow = 6, ncol = 2)

for(i in 1:6)
{
    netFacet[i, 1] <- cor(cfaScores[, i], netScores$Standardized[, i])
    netFacet[i, 2] <- rmse(scale(cfaScores[, i]),netScores$Standardized[, i])
}

# Identify unique facets
uniq <- unique(facets)

# Participant facet means
meanFacet <- matrix(0, nrow = 6, ncol = 2)
for(i in 1:6)
{
    meanFacet[i, 1] <- cor(cfaScores[, i], rowMeans(neoOpen[, which(facets == uniq[i])]))
    meanFacet[i, 2] <- rmse(scale(cfaScores[, i]), scale(rowMeans(neoOpen[, which(facets == uniq[i])])))
}
```

```
# Compare network latent facet scores and participant facet means
comp <- cbind(netFacet, meanFacet)
row.names(comp) <- c("actions", "aesthetics",
                     "fantasy", "feelings",
                     "ideas", "values")
colnames(comp) <- c("netCor", "netRMSE", "meanCor", "meanRMSE")
```

Table 3 displays the results from the above code.

	Network Latent Facet Scores		**Participant Facet Means**	
	r	rmse	*r*	rmse
actions	**.96**	**.278**	.92	.394
aesthetics	.98	**.176**	.98	.220
fantasy	**.98**	**.174**	.97	.255
feelings	.97	.238	.97	**.204**
ideas	**.99**	**.171**	.98	.211
values	.97	.225	**.98**	**.217**

Table 3: Pearson's correlation (*r*) and root mean square error (rmse) are shown in relation to the CFA latent facet scores. Bolded values signify better values.

The results shown in Table 3 suggest that the network latent community scores are equal to or more related to the CFA latent facet scores than the participant facet means. Once again, this suggests that the network latent scores are effective for estimating latent scores more generally. Similar to the global latent scores, the network latent community scores were not perfectly related to the CFA latent facet scores. Taken together, however, this provides evidence that latent network scores are plausible. Moreover, the advantage of the proposed method is that network adjusted sums and means can be computed, meaning sum scores can be altered based on each item's position in the network, providing differentiation between identical sum scores—that is, unless two cases with identical sum scores have identical data patterns, they will have different scores based on the structure of the network.

Summary

Network science is a rapidly developing statistical approach in psychology. The appeal of the approach is the intuitive modeling of complexity that is intrinsic in many psychological phenomena. **NetworkToolbox** is designed to equip researchers with different network tools that have been developed over decades of work in graph theory and the complex systems domains. In this article, many functions of **NetworkToolbox** were introduced with relevant details for appropriate use. The examples of code used throughout this article can be used as a blueprint for any researcher interested in running their own psychometric network analysis. Notably, not all of the functions available in **NetworkToolbox** were discussed; however, documentation with examples are provided within the package. Many of the arguments discussed in this article are also used across the package. Notably, **NetworkToolbox** is not exhaustive—there are many other methods and measures for constructing, analyzing, and quantifying networks. Because of this, **NetworkToolbox** will continue to expand to equip researchers with the newest advances in the domain.

Computational details

The results of this paper were obtained using R 3.4.4 with the **Networktoolbox** 1.2.1 package and the networks were visualized using the **qgraph** 1.4.4 package. EGA was provided by the **EGA** 0.4 package (available from GitHub: `https://github.com/hfgolino/EGA`). The CFA model was generated using the **lavaan** 0.5-23.1097 (BETA) package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`.

Bibliography

T. Aste and T. Di Matteo. Sparse causality network retrieval from short time series. *Complexity*, 2017:1 – 13, 2017. URL `https://doi.org/10.1155/2017/4518429`. [p423]

A.-L. Barabási. The network takeover. *Nature Physics*, 8(1):14–16, 2011. URL https://doi.org/10.1038/nphys2188. [p422]

W. Barfuss, G. P. Massara, T. Di Matteo, and T. Aste. Parsimonious modeling with information filtering networks. *Physical Review E*, 94(6):062306, 2016. URL https://doi.org/10.1103/PhysRevE.94.062306. [p423, 424, 425]

T. F. Blanken, M. K. Deserno, J. Dalege, D. Borsboom, P. Blanken, G. A. Kerkhof, and A. O. Cramer. The role of stabilizing and communicating symptoms given overlapping communities in psychopathology networks. *Scientific Reports*, 8(1):5854, 2018. URL https://doi.org/10.1038/s41598-018-24224-2. [p429]

D. Borsboom. A network theory of mental disorders. *World Psychiatry*, 16(1):5–13, 2017. URL https://doi.org/10.1002/wps.20375. [p422]

D. Borsboom and A. O. Cramer. Network analysis: An integrative approach to the structure of psychopathology. *Annual Review of Clinical Psychology*, 9:91–121, 2013. URL https://doi.org/10.1146/annurev-clinpsy-050212-185608. [p422]

C. T. Butts and others. Social network analysis with **sna**. *Journal of Statistical Software*, 24(6):1–51, 2008. URL http://doi.org/10.18637/jss.v014.i06. [p422]

A. P. Christensen, K. N. Cotter, and P. J. Silvia. Reopening openness to experience: A network analysis of four openness to experience inventories. *Journal of Personality Assessment*, pages 1–15, 2018a. URL https://doi.org/10.1080/00223891.2018.1467428. [p423]

A. P. Christensen, Y. N. Kenett, T. Aste, P. J. Silvia, and T. R. Kwapil. Network structure of the wisconsin schizotypy scales-short forms: Examining psychometric network filtering approaches. *Behavior Research Methods*, 50:2531–2550, 2018b. URL https://doi.org/10.3758/s13428-018-1032-9. [p428]

A. P. Christensen, G. M. Gross, H. F. Golino, P. J. Silvia, and T. R. Kwapil. Exploratory graph analysis of the multidimensional schizotypy scale. *Schizophrenia Research*, in press. URL https://doi.org/10.1016/j.schres.2018.12.018. [p429]

Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965. URL https://ci.nii.ac.jp/naid/10030090917/en/. [p423, 425]

K. N. Cotter, A. P. Christensen, and P. J. Silvia. Understanding inner music: A dimensional approach to musical imagery. *Psychology of Aesthetics, Creativity, and the Arts*, 2018. [p429]

A. O. Cramer, S. Sluis, A. Noordhof, M. Wichers, N. Geschwind, S. H. Aggen, K. S. Kendler, and D. Borsboom. Dimensions of normal personality as networks in search of equilibrium: You can't like parties if you don't like people. *European Journal of Personality*, 26(4):414–431, 2012. URL https://doi.org/10.1002/per.1866. [p422]

G. Csardi and T. Nepusz. The **igraph** software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006. URL https://pdfs.semanticscholar.org/1d27/44b83519657f5f2610698a8ddd177ced4f5c.pdf. [p422]

C. G. DeYoung, L. C. Quilty, and J. B. Peterson. Between facets and domains: 10 aspects of the big five. *Journal of Personality and Social Psychology*, 93(5):880–896, 2007. URL https://doi.org/10.1037/0022-3514.93.5.880. [p423]

J. Edmonds. Optimum branchings. *Mathematics and the Decision Sciences*, 1(335-345), 1968. [p423, 425]

S. Epskamp, A. O. Cramer, L. J. Waldorp, V. D. Schmittmann, D. Borsboom, and others. **qgraph**: Network visualizations of relationships in psychometric data. *Journal of Statistical Software*, 48(4):1–18, 2012. URL https://doi.org/10.18637/jss.v048.i04. [p422]

S. Epskamp, M. Rhemtulla, and D. Borsboom. Generalized network psychometrics: Combining network and latent variable models. *Psychometrika*, 82(4):904–927, 2017. URL https://doi.org/10.1007/s11336-017-9557-x. [p431]

S. Epskamp, D. Borsboom, and E. I. Fried. Estimating psychological networks and their accuracy: A tutorial paper. *Behavior Research Methods*, 50(1):195–212, 2018. URL https://doi.org/10.3758/s13428-017-0862-1. [p423, 428]

F. D. V. Fallani, V. Latora, and M. Chavez. A topological criterion for filtering information in complex brain networks. *PLoS Computational Biology*, 13(1):e1005305, 2017. URL https://doi.org/10.1371/journal.pcbi.1005305. [p425]

S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010. URL https://doi.org/10.1016/j.physrep.2009.11.002. [p429]

L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977. URL https://www.jstor.org/stable/3033543. [p427]

E. I. Fried and R. M. Nesse. Depression sum-scores don't add up: Why analyzing specific depression symptoms is essential. *BMC Medicine*, 13(1):72, 2015. URL https://doi.org/10.1186/s12916-015-0325-4. [p431]

J. Friedman, T. Hastie, and R. Tibshirani. *Glasso: Graphical Lasso- Estimation of Gaussian Graphical Models*, 2014. URL https://CRAN.R-project.org/package=glasso. R package version 1.8. [p423]

P.-L. Giscard and R. C. Wilson. A centrality measure for cycles and subgraphs ii. *Applied Network Science*, 3(1):9, 2018. URL https://doi.org/10.1007/s41109-018-0064-5. [p429]

H. F. Golino. **EGA***: Exploratory Graph Analysis - Estimating the Number of Dimensions in Psychological Data*, 2018. URL http://github.com/hfgolino/EGA. R package version 0.4. [p429]

H. F. Golino and A. Demetriou. Estimating the dimensionality of intelligence like data using exploratory graph analysis. *Intelligence*, 62:54–70, 2017. URL https://doi.org/10.1016/j.intell.2017.02.007. [p429]

H. F. Golino and S. Epskamp. Exploratory graph analysis: A new approach for estimating the number of dimensions in psychological research. *PloS one*, 12(6):e0174035, 2017. URL https://doi.org/10.1371/journal.pone.0174035. [p429]

H. F. Golino, D. Shi, L. E. Garrido, A. P. Christensen, M. D. Nieto, R. Sadana, and J. A. Thiyagarajan. Investigating the performance of exploratory graph analysis and traditional techniques to identify the number of latent factors: A simulation and tutorial. *PsyArXiv*, pages 1–20, 2018. URL https://doi.org/10.31234/osf.io/gzcre. [p429]

S. M. Goodreau, M. S. Handcock, D. R. Hunter, C. T. Butts, and M. Morris. A **statnet** tutorial. *Journal of Statistical Software*, 24(9):1–27, 2008. URL https://doi.org/10.18637/jss.v024.i09. [p422]

U. Grömping and others. Relative importance for linear regression in R: The package **relaimpo**. *Journal of Statistical Software*, 17(1):1–27, 2006. URL https://doi.org/10.18637/jss.v017.i01. [p426]

Y. Jacob, Y. Winetraub, G. Raz, E. Ben-Simon, H. Okon-Singer, K. Rosenberg-Katz, T. Hendler, and E. Ben-Jacob. Dependency network analysis (depna) reveals context related influence of brain network nodes. *Scientific Reports*, 6:27444, 2016. URL https://doi.org/10.1038/srep27444. [p426]

J. W. Johnson and J. M. LeBreton. History and use of relative importance indices in organizational research. *Organizational Research Methods*, 7(3):238–257, 2004. URL https://doi.org/10.1177/1094428104266510. [p426]

D. Y. Kenett, M. Tumminello, A. Madi, G. Gur-Gershgoren, R. N. Mantegna, and E. Ben-Jacob. Dominating clasp of the financial sector revealed by partial correlation analysis of the stock market. *PloS one*, 5(12):e15032, 2010. URL https://doi.org/10.1371/journal.pone.0015032. [p426]

Y. N. Kenett, D. Y. Kenett, E. Ben-Jacob, and M. Faust. Global and local features of semantic networks: Evidence from the Hebrew mental lexicon. *PloS one*, 6(8):e23912, 2011. URL https://doi.org/10.1371/journal.pone.0023912. [p429]

Y. N. Kenett, D. Wechsler-Kashi, D. Y. Kenett, R. G. Schwartz, E. Ben Jacob, and M. Faust. Semantic organization in children with cochlear implants: Computational analysis of verbal fluency. *Frontiers in Psychology*, 4:543, 2013. URL https://doi.org/10.3389/fpsyg.2013.00543. [p429]

I. Kivimäki, B. Lebichot, J. Saramäki, and M. Saerens. Two betweenness centrality measures based on randomized shortest paths. *Scientific Reports*, 6:19668, 2016. URL https://doi.org/10.1038/srep19668. [p427, 428]

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009. [p424]

S. Korkmaz, D. Goksuluk, and G. Zararsiz. **MVN**: An R package for assessing multivariate normality. *The R Journal*, 6(2):151–162, 2014. URL https://doi.org/10.1111/j.1469-1809.1936.tb02137.x. [p426]

R. Kotov, R. F. Krueger, D. Watson, T. M. Achenbach, R. R. Althoff, R. M. Bagby, T. A. Brown, W. T. Carpenter, A. Caspi, L. A. Clark, and others. The hierarchical taxonomy of psychopathology (hitop): A dimensional alternative to traditional nosologies. *Journal of Abnormal Psychology*, 126(4):454–477, 2017. URL https://doi.org/10.1037/abn0000258. [p423]

K. Lee and M. C. Ashton. Psychometric properties of the hexaco-100. *Assessment*, 1073191116659134: 1–15, 2016. URL https://doi.org/10.1177/1073191116659134. [p423]

P.-L. Loh and M. J. Wainwright. Structure estimation for discrete graphical models: Generalized covariance matrices and their inverses. *The Annals of Statistics*, pages 3022–3049, 2013. URL http://dx.doi.org/10.1214/13-AOS1162. [p426]

R. N. Mantegna. Hierarchical structure in financial markets. *The European Physical Journal B Condensed Matter and Complex Systems*, 11(1):193–197, 1999. URL https://doi.org/10.1007/s100510050929. [p423]

R. Marcaccioli and G. Livan. A parametric approach to information filtering in complex networks: The pólya filter. *arXiv*, 2018. URL https://arxiv.org/abs/1806.09893. [p423]

K. E. Markon, R. F. Krueger, and D. Watson. Delineating the structure of normal and abnormal personality: An integrative hierarchical approach. *Journal of Personality and Social Psychology*, 88(1): 139–157, 2005. URL https://doi.org/10.1037/0022-3514.88.1.139. [p423]

G. P. Massara, T. Di Matteo, and T. Aste. Network filtering for big data: Triangulated maximally filtered graph. *Journal of Complex Networks*, 5(2):161–178, 2016. URL https://doi.org/10.1093/comnet/cnw015. [p423, 424, 425]

R. R. McCrae. A more nuanced view of reliability: Specificity in the trait hierarchy. *Personality and Social Psychology Review*, 19(2):97–112, 2015. URL https://doi.org/10.1177/1088868314541857. [p423]

R. R. McCrae and P. T. Costa Jr. Brief versions of the neo-pi-3. *Journal of Individual Differences*, 28(3): 116–128, 2007. URL https://doi.org/10.1027/1614-0001.28.3.116. [p423]

R. R. McCrae, P. T. Costa Jr, and T. A. Martin. The neo-pi-3: A more readable revised neo personality inventory. *Journal of Personality Assessment*, 84(3):261–270, 2005. URL https://doi.org/10.1207/s15327752jpa8403_05. [p423]

P. Pons and M. Latapy. Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, 10(2):191–218, 2006. URL https://doi.org/10.1007/11569596_31. [p429]

F. Pozzi, T. Di Matteo, and T. Aste. Spread of risk across financial markets: Better to invest in the peripheries. *Scientific Reports*, 3:1665, 2013. URL https://doi.org/10.1038/srep01665. [p428]

W. Revelle. **psych**: *Procedures for Psychological, Psychometric, and Personality Research*. Northwestern University, Evanston, Illinois, 2017. URL https://CRAN.R-project.org/package=psych. R package version 1.7.8. [p425, 426]

D. J. Robinaugh, N. J. LeBlanc, H. A. Vuletich, and R. J. McNally. Network analysis of persistent complex bereavement disorder in conjugally bereaved adults. *Journal of Abnormal Psychology*, 123(3): 510–522, 2014. URL http://dx.doi.org/10.1037/abn0000002. [p426]

Y. Rosseel. **lavaan**: An R package for structural equation modeling and more. version 0.5–12 (beta). *Journal of Statistical Software*, 48(2):1–36, 2012. URL https://doi.org/10.18637/jss.v048.i02. [p432]

V. D. Schmittmann, A. O. Cramer, L. J. Waldorp, S. Epskamp, R. A. Kievit, and D. Borsboom. Deconstructing the construct: A network perspective on psychological phenomena. *New Ideas in Psychology*, 31(1):43–53, 2013. URL https://doi.org/10.1016/j.newideapsych.2011.02.007. [p422]

W.-M. Song, T. Di Matteo, and T. Aste. Hierarchical information clustering by means of topologically embedded graphs. *PLoS One*, 7(3):e31929, 2012. URL https://doi.org/10.1371/journal.pone.0031929. [p423]

R. C. Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL https://www.R-project.org/. [p422]

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL http://www.jstor.org/stable/2346178. [p423]

M. Tumminello, T. Aste, T. Di Matteo, and R. N. Mantegna. A tool for filtering information in complex systems. *Proceedings of the National Academy of Sciences*, 102(30):10421–10426, 2005. URL https://doi.org/10.1073/pnas.0500298102. [p423]

C. D. van Borkulo, D. Borsboom, S. Epskamp, T. F. Blanken, L. Boschloo, R. A. Schoevers, and L. J. Waldorp. A new method for constructing networks from binary data. *Scientific Reports*, 4:5918, 2014. URL https://doi.org/10.1038/srep05918. [p422, 423]

C. G. Watson. **brainGraph***: Graph Theory Analysis of Brain MRI Data*, 2017. URL https://CRAN.R-project.org/package=brainGraph. R package version 1.0.0. [p422]

S. E. Woo, O. S. Chernyshenko, A. Longley, Z.-X. Zhang, C.-Y. Chiu, and S. E. Stark. Openness to experience: Its lower level structure, measurement, and cross-cultural equivalence. *Journal of Personality Assessment*, 96(1):29–45, 2014. URL https://doi.org/10.1080/00223891.2013.806328. [p423]

Alexander Christensen
Department of Psychology
P.O. Box 26170
University of North Carolina at Greensboro
Greensboro, NC, 27402-6170, USA
ORCiD: 0000-0002-9798-7037
apchrist@uncg.edu

jsr223: A Java Platform Integration for R with Programming Languages Groovy, JavaScript, JRuby, Jython, and Kotlin

by Floid R. Gilbert and David B. Dahl

Abstract The R package **jsr223** is a high-level integration for five programming languages in the Java platform: Groovy, JavaScript, JRuby, Jython, and Kotlin. Each of these languages can use Java objects in their own syntax. Hence, **jsr223** is also an integration for R and the Java platform. It enables developers to leverage Java solutions from within R by embedding code snippets or evaluating script files. This approach is generally easier than **rJava**'s low-level approach that employs the Java Native Interface. **jsr223**'s multi-language support is dependent on the Java Scripting API: an implementation of "JSR-223: Scripting for the Java Platform" that defines a framework to embed scripts in Java applications. The **jsr223** package also features extensive data exchange capabilities and a callback interface that allows embedded scripts to access the current R session. In all, **jsr223** makes solutions developed in Java or any of the **jsr223**-supported languages easier to use in R.

Introduction

About the same time Ross Ihaka and Robert Gentleman began developing R at the University of Auckland in the early 1990s, James Gosling and the so-called Green Project Team was working on a new programming language at Sun Microsystems in California. The Green Team did not set out to make a new language; rather, they were trying to move platform-independent, distributed computing into the consumer electronics marketplace. As Gosling explained, "All along, the language was a tool, not the end" (O'Connell, 1995). Unexpectedly, the programming language outlived the Green Project and sparked one of the most successful development platforms in computing history: Java. According to the TIOBE index, Java has been the most popular programming language, on average, over the last sixteen years. Java's success can be attributed to several factors. Perhaps the most important factor is platform-independence: the same Java program can run on several operating systems and hardware devices. Another important factor is that memory management is handled automatically for the programmer. Consequently, Java programs are easier to write and have fewer memory-related bugs than programs written in C/C++. These and other factors accelerated Java's adoption in enterprise systems which, in turn, established a thriving developer community that has created production-quality frameworks, libraries, and programming languages for the Java platform. Many successful Java solutions are relevant to data science today such as Hadoop, Hive, Spark, Cassandra, HBase, Mahout, Deeplearning4j, Stanford CoreNLP, and others.

In 2003, Simon Urbanek released **rJava** (2017), an integration package designed to avail R of the burgeoning development surrounding Java. The package has been very successful to this end. In the year 2018 alone, **rJava** registered over 1.95 million downloads on CRAN.[1] The **rJava** package is described by Urbanek as a low-level R to Java interface analogous to `.C` and `.Call`, the built-in R functions for calling compiled C code. Like R's integration for C, **rJava** loads compiled code into an R process's memory space where it can be accessed via various R functions. Urbanek achieves this feat using the Java Native Interface (JNI), a standard framework that enables native (i.e. platform-dependent) code to access and use compiled Java code. The **rJava** API requires users to specify classes and data types in JNI syntax. One advantage to this approach is that it gives users granular, direct access to Java classes. However, as with any low-level interface, the learning curve is relatively high and implementation requires verbose coding. A second advantage to using JNI is that it avoids the difficult task of dynamically interpreting or compiling source code. Of course, this is also a disadvantage: it limits **rJava** to using compiled code as opposed to embedding source code directly within R script.

Our **jsr223** package builds on **rJava** to provide a high-level interface to the Java platform. We accomplish this by embedding other programming languages in R that use Java objects in natural syntax. As we show in Section "**rJava** software review", this approach is generally simpler and more intuitive than **rJava**'s low-level JNI interface. To date, **jsr223** supports embedding five programming languages: Groovy, JavaScript, JRuby, Jython, and Kotlin. (JRuby and Jython are Java platform implementations of the Ruby and Python languages, respectively.) See Table 1 for a brief description of each language.

[1]Downloads tabulated by the **cranlogs** package (Csardi, 2015).

Language	Description
Groovy	Groovy is a scripting language that follows Java syntax very closely. Hence, **jsr223** enables developers to embed Java source code directly in R script. Groovy also supports an optionally typed, functional paradigm with relaxed syntax for less verbose code.
JavaScript	JavaScript is well known for its use in web applications. However, its popularity has overflowed into standalone solutions involving databases, plotting, machine learning, and network-enabled utilities, to name just a few. The **jsr223** package uses Nashorn, the ECMA-compliant JavaScript implementation for the Java platform.
JRuby	JRuby is the Ruby implementation for the Java platform. Ruby is a general-purpose, object-oriented language with unique syntax. It is often used with the web application framework Ruby on Rails. Ruby libraries, called *gems*, can be accessed via **jsr223**.
Jython	Jython is the Python implementation for the Java platform. Like R, the Python programming language is used widely in science and analytics. Python has many powerful language features, yet it is known for being concise and easy to read. Popular libraries SciPy and NumPy are available for the Java platform through JyNI (the Jython Native Interface).
Kotlin	Kotlin version 1.0 was released in 2016 making it the newest **jsr223**-supported language. It is a statically typed language that supports both functional and object-oriented programming paradigms. Kotlin has similarities to Java, but it often requires less code than Java to accomplish the same task. Kotlin and Java are the only languages officially supported by Google for Android application development.

Table 1: The five programming languages supported by **jsr223**. See the **jsr223** *User Manual* for code examples and language details.

The **jsr223** multi-language integration is made possible by the Java Scripting API (Oracle, 2016), an implementation of the specification "JSR-223: Scripting for the Java Platform" (Sun Microsystems, Inc., 2006). The JSR-223 specification includes two crucial elements: an interface for Java applications to execute code written in scripting languages, and a guide for scripting languages to create Java objects in their own syntax. Hence, JSR-223 is the basis for our package. However, no knowledge of JSR-223 or the Java Scripting API is necessary to use **jsr223**. Figures 1 and 2 show how **rJava** and **jsr223** facilitate access to the Java platform. Where **rJava** uses JNI, **jsr223** uses the Java Scripting API and embeddable programming languages.

The primary goal of **jsr223** is to enable R developers to leverage existing Java solutions with relative ease. We demonstrate two typical use cases in this document with subjects that are of particular interest to many data scientists: a natural language processor and a neural network classifier. In addition to Java solutions, R developers can use projects developed in any of the five **jsr223**-supported programming languages. In essence, **jsr223** opens R to a broader ecosystem.

For Java developers, **jsr223** facilitates writing high-performance, cross-platform R extensions using their preferred platform. The **jsr223** package also allows organizations that run enterprise Java applications to more readily develop dashboards and other business intelligence tools. Instead of writing R code to query raw data from a database, **jsr223** enables R packages to consume data directly from their application's Java object model where the data has been coalesced according to business rules. Java developers will also be interested to know that the **jsr223**-supported programming languages can implement interfaces and extend classes, just like the Java programming language. See "Extending existing Java solutions" in the **jsr223** *User Manual* for an in-depth code example that demonstrates extending Java classes and several other features.

In this paper, we present an introduction to the **jsr223** package. Section "**jsr223** package implementation and features overview" contains a brief description of the package's primary features and internals. The section "Typical use cases" provides code examples that highlight the **jsr223** package's core functionality. Finally, the "Software review" section puts the **jsr223** project in context with

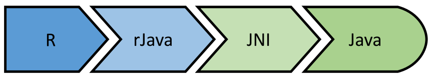


Figure 1: The **rJava** package facilitates low-level access to the Java platform through the Java Native Interface (JNI). Some knowledge of JNI is required.

comparisons to other relevant software solutions.

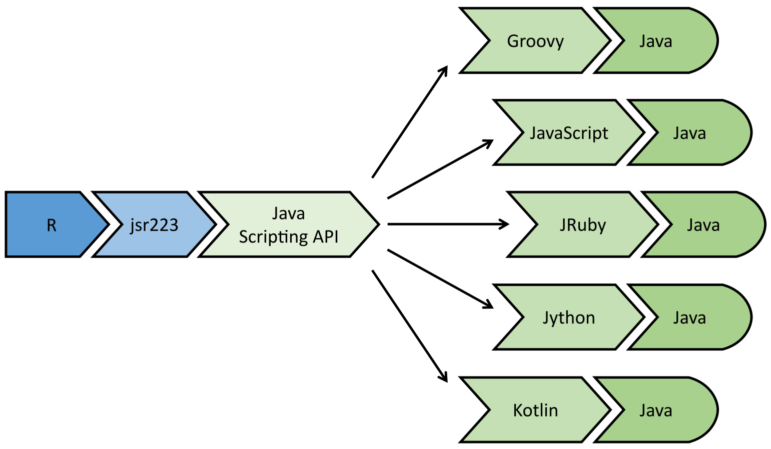


Figure 2: The **jsr223** package provides high-level access to the Java platform through five programming languages. Although **jrs223** uses the Java Scripting API in its implementation, users do not need to learn the API.

The jsr223 package implementation and features overview

The **jsr223** package supports most of the major programming languages that implement JSR-223. Technically, any JSR-223 implementation will work with our package, but we may not officially support some languages. The most notable exclusion is Scala; we don't support it simply because the JSR-223 implementation is not complete. (Consider, instead, the **rscala** package for a Scala/R integration (Dahl, 2018).) We also exclude languages that are not actively developed, such as BeanShell.

The **jsr223** package features extensive, configurable data exchange between R and Java via **jsr223**'s companion package **jdx** (Gilbert and Dahl, 2018). R vectors, factors, n-dimensional arrays, data frames, lists, and environments are converted to standard Java objects. Java scalars, n-dimensional arrays, maps, and collections are inspected for content and converted to the most appropriate R structure (vectors, n-dimensional arrays, data frames, or lists). Several data exchange options are available including row-major and column-major ordering schemes for data frames and n-dimensional arrays. Many language integrations for R provide a comparable feature set by using JSON (JavaScript Object Notation) libraries. In contrast, the **jsr223** package implements data exchange using custom Java routines to avoid the serialization overhead and loss of floating point precision inherent in JSON data conversion.

The **jsr223** package also supports converting the most common data structures from the **jsr223**-supported languages. For example, **jsr223** can convert Jython dictionaries and user-defined JavaScript objects to R objects. Behind the scenes, every Java-based programming language uses Java objects. For example, a Jython dictionary is backed by a Java object that defines the dictionary's behavior. The **jsr223** package uses **jdx** to inspect these Java objects for data and convert them to an appropriate R object. In most cases, the default conversion rules are intuitive and seamless. Details covering data exchange features and behavior can be found in the **jsr223** *User Manual* and the **jdx** package vignette.

The **jsr223** programming interface follows design cues from **rscala**, and **V8** (Ooms, 2017b). The application programming interface is implemented using **R6** (Chang, 2017) classes for a traditional object-oriented style of programming. **R6** objects wrap methods in an R environment making them accessible from the associated variable using list-like syntax (e.g., `myObject$myMethod()`).

The **jsr223** package uses **rJava** to load and communicate with the Java Virtual Machine (JVM): the abstract computing environment that executes compiled Java code. The **jsr223** package employs a client-server architecture and a custom multi-threaded messaging protocol to exchange data and handle script execution. This protocol optimizes performance by eliminating **rJava** calls that inspect generic return values and transform data, both which incur significant overhead. The protocol also facilitates callbacks that allow embedded scripts to manipulate variables and evaluate R code in the current R session. This callback implementation is lightweight, does not require any special R software configuration, and supports infinite callback recursion between R and the script engine (limited only by stack space).

Other distinguishing **jsr223** features include script compiling and string interpolation. Complete feature documentation is available in the **jsr223** *User Manual* vignette.

Typical use cases

This section includes examples that demonstrate typical use cases for the **jsr223** package. More code examples are available in the **jsr223** *User Manual*.

Using Java libraries

For this introductory example, we use Stanford's Core Natural Language Processing Java libraries (Manning et al., 2014) to identify grammatical parts of speech in a text. Natural language processing (NLP) is a key component in statistical text analysis and artificial intelligence. This example shows how so-called "glue" code can be embedded in R to quickly leverage the Stanford NLP libraries. It also demonstrates how easily **jsr223** converts Java data structures to R objects. The full script is available at `https://github.com/floidgilbert/jsr223/tree/master/examples/JavaScript/stanford-nlp.R`.

The first step: create a **jsr223** `"ScriptEngine"` instance that can dynamically execute source code. In this case, we use a JavaScript engine. The object is created using the `ScriptEngine$new` constructor method. This method takes two arguments: a scripting language's name and a character vector containing paths to the required Java libraries. In the code below, the `class.path` variable contains the required Java library paths. The new `"ScriptEngine"` object is assigned to the variable `engine`.

```
class.path <- c(
  "./protobuf.jar",
  "./stanford-corenlp-3.9.0.jar",
  "./stanford-corenlp-3.9.0-models.jar"
)
library("jsr223")
engine <- ScriptEngine$new("JavaScript", class.path)
```

Now we can execute JavaScript source code. The **jsr223** interface provides several methods to do so. In this example, we use the `%@%` operator; it executes a code snippet and discards the return value, if any. The code snippet imports the Stanford NLP `"Document"` class. The import syntax is peculiar to the JavaScript dialect. The result, `DocumentClass`, is used to instantiate objects or access static methods.

```
engine %@% 'var DocumentClass = Java.type("edu.stanford.nlp.simple.Document");'
```

The next code sample defines a JavaScript function named `getPartsOfSpeech`. It tags each element in a text with a grammatical part of speech (e.g., noun, adjective, or verb). The function parses the text using a new instance of the `"Document"` class. The parsing results are transferred to a list of JavaScript objects. Each JavaScript object contains the parsing information for a single sentence.

```
engine %@% '
  function getPartsOfSpeech(text) {
    var doc = new DocumentClass(text);
    var list = [];
    for (i = 0; i < doc.sentences().size(); i++) {
      var sentence = doc.sentences().get(i);
      var o = {
        "words":sentence.words(),
        "pos.tag":sentence.posTags(),
        "offset.begin":sentence.characterOffsetBegin(),
        "offset.end":sentence.characterOffsetEnd()
      }
      list.push(o);
    }
    return list;
  }
'
```

We use engine$invokeFunction to call the JavaScript function getPartsOfSpeech from R. The method invokeFunction takes the name of the function as the first parameter; any arguments that follow are automatically converted to Java objects and passed to the JavaScript function. The function's return value is converted to an R object. In this case, **jsr223** intuitively converts the list of JavaScript objects to a list of R data frames as seen in the output below. The parts of speech abbreviations are defined by the Penn Treebank Project (Taylor et al., 2003). A quick reference is available at https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

```
engine$invokeFunction(
  "getPartsOfSpeech",
  "The jsr223 package makes Java objects easy to use. Download it from CRAN."
)
```

```
## [[1]]
##      words pos.tag offset.begin offset.end
## 1      The      DT            0          3
## 2   jsr223      NN            4         10
## 3  package      NN           11         18
## 4    makes     VBZ           19         24
## 5     Java     NNP           25         29
## 6  objects     NNS           30         37
## 7     easy      JJ           38         42
## 8       to      TO           43         45
## 9      use      VB           46         49
## 10       .       .           49         50
##
## [[2]]
##      words pos.tag offset.begin offset.end
## 1 Download      VB           51         59
## 2       it     PRP           60         62
## 3     from      IN           63         67
## 4     CRAN     NNP           68         72
## 5        .       .           72         73
```

In this example, we effectively used Stanford's Core NLP library with a minimal amount of code. This same functionality can be replicated in any of the **jsr223**-supported programming languages.

Using Java libraries with complex dependencies

In this example we use Deeplearning4j (DL4J) (Eclipse Deeplearning4j Development Team, 2018) to build a neural network. DL4J is an open-source deep learning solution for the Java platform. It is notable both for its scalability and performance. DL4J can run on a local computer with a standard CPU, or it can use Spark for distributed computing and GPUs for massively parallel processing. DL4J is modular in design and it has a large number of dependencies. As with many other Java solutions, it is designed to be installed using a software project management utility like Apache Maven, Gradle, or sbt. These utilities feature dependency managers that automatically download a library's dependencies from a central repository and make them accessible to your project. This is similar to installing an R package from CRAN using install.packages; by default, any referenced packages are also downloaded and installed.

The primary goal of this example is to show how **jsr223** can easily leverage complex Java solutions with the help of a project management utility. We will install both Groovy and DL4J using Apache Maven. We will then integrate a Groovy script with R to create a simple neural network. The process is straightforward: i.) create a skeleton Java project; ii.) add dependencies to the project; iii.) build a class path referencing all of the dependencies; and iv.) pass the class path to **jsr223**. Though we use Maven here, the same concepts apply to any project management utility that supports Java.

To begin, visit the Maven web site (https://maven.apache.org/) and follow the installation instructions for your operating system. Next, create an empty folder for this sample project. Open a terminal (a system command prompt) and change the current directory to the project folder. Execute the following Maven command. It will create a skeleton Java project named 'stub' in a subfolder by the same name. The Java project is used only to retrieve dependencies; it is not required for the R project. If this is the first time Maven has been executed on your computer, several files will be downloaded to the local Maven repository cache on your computer.

```
mvn archetype:generate -DgroupId=none -DartifactId=stub -DinteractiveMode=false
```

Open the Maven project object model file, 'stub/pom.xml', in a plain text editor or an XML editor. Locate the XML element <dependencies>. It will be similar to the example displayed below. A <dependency> child element defines a single project dependency that will be retrieved from the Maven repository. Notice that a dependency has a group ID, an artifact ID, and a version. (*Artifact* is the general term for any file residing in a repository.) How do you know which dependencies are required for your project? They are often provided in the installation documentation. Or, if you are starting from a code example, dependencies can be located in a Maven repository using fully-qualified Java class names.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Maven dependency definitions can be located at https://search.maven.org. We will search for dependencies using the syntax 'g:<group-id> a:<artifact-id>'. This avoids erroneous results and near-matches. A search string for each dependency in our demonstration is provided in the bullet list below. Perform a search using the first bullet item. In the search results, click the version number under the column heading "Latest Version." On the right-hand side of the page that follows you will see an XML Maven dependency definition for the artifact. Copy the XML and insert it after the last </dependency> end tag in your 'pom.xml' file. It is not necessary to preserve indentations or other white space. Repeat this process for each of the remaining search strings below.

- g:org.apache.logging.log4j a:log4j-core
- g:org.slf4j a:slf4j-log4j12
- g:org.deeplearning4j a:deeplearning4j-core
- g:org.nd4j a:nd4j-native-platform
- g:org.datavec a:datavec-api
- g:org.codehaus.groovy a:groovy-all

Save the 'pom.xml' file. In your terminal window, change directories to the Java project folder ('stub') and execute the following Maven command. This will download all of the dependencies to a local repository cache on your computer. It will also create a file named 'jsr223.classpath' in the parent folder. It contains a class path referencing all of the dependencies that will be used by **jsr223**.

```
mvn dependency:build-classpath -Dmdep.outputFile="../jsr223.classpath"
```

Now everything is in place to create a neural network using Groovy and DL4J. To keep the example simple, we use a feedforward neural network to classify species in the iris data set. The example involves an R script ('dl4j.R') and a Groovy script ('dl4j.groovy'). Both scripts can be downloaded from https://github.com/floidgilbert/jsr223/tree/master/examples/Groovy/dl4j. Save both scripts in the same folder as 'jsr223.classpath'.

The R script First, we read in the class path created by Maven and create the Groovy script engine.

```
library(jsr223)

file.name <- "jsr223.classpath"
class.path <- readChar(file.name, file.info(file.name)$size)
engine <- ScriptEngine$new("groovy", class.path)
```

Next, we set a seed for reproducible results. The value is saved in a variable that will be retrieved by the Groovy script.

```
seed <- 10
set.seed(seed)
```

The code that follows splits the iris data into train and test matrices. The inputs are centered and scaled. The labels are converted to a binary matrix format: for each record, the number 1 is placed in the column corresponding to the correct label.

```
train.idx <- sample(nrow(iris), nrow(iris) * 0.65)
train <- scale(as.matrix(iris[train.idx, 1:4]))
train.labels <- model.matrix(~ -1 + Species, iris[train.idx, ])
test <- scale(as.matrix(iris[-train.idx, 1:4]))
test.labels <- model.matrix(~ -1 + Species, iris[-train.idx, ])
```

Finally, we execute the Groovy script. The results will be printed to the console.

```
result <- engine$source("dl4j.groovy")
cat(result)
```

The Groovy script The Groovy script here follows Java syntax with one exception: we provide no class. Instead, we place all of the code at the top level to be executed at once. This is merely a style choice to keep the code samples easy to follow. The script begins by importing the necessary classes.

```
import org.deeplearning4j.eval.Evaluation;
import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.layers.DenseLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.cpu.nativecpu.NDArray;
import org.nd4j.linalg.dataset.DataSet;
import org.nd4j.linalg.learning.config.Sgd;
import org.nd4j.linalg.lossfunctions.LossFunctions;
```

Next, we convert the train and test data from R objects to the `DataSet` objects consumed by DL4J. We retrieve the data from the R environment using the `get` method of **jsr223**'s built-in `R` object. The R matrices are automatically converted to multi-dimensional Java arrays. These arrays are used to instantiate the `NDArray` objects which, in turn, are used to instantiate the `DataSet` objects.

```
DataSet train = new DataSet(
    new NDArray(R.get("train")),
    new NDArray(R.get("train.labels"))
);
DataSet test = new DataSet(
    new NDArray(R.get("test")),
    new NDArray(R.get("test.labels"))
);
```

Pulling the data from the R environment using the `R` object is just one convenient way to share data between R and the Java environment. It is also possible to push data from R to the Groovy environment, or to pass the data as function parameters. Note: for very large data sets it is impractical to exchange data between R and Java using **jsr223** methods. Instead, load the data on the Java side for processing using DL4J classes optimized for big data.

Here we configure a feedforward neural network with backpropagation. The network consists of four inputs, a seven node hidden layer, a three node hidden layer, and a three node output layer. An explanation of the network's hyperparameters is beyond the scope of this discussion. See https://deeplearning4j.org/docs/latest/deeplearning4j-troubleshooting-training for a DL4J hyperameter reference.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(R.get("seed").intValue())
    .activation(Activation.TANH)
    .weightInit(WeightInit.XAVIER)
    .updater(new Sgd(0.1)) // Learning rate.
    .list()
    .layer(new DenseLayer.Builder().nIn(4).nOut(7).build())
    .layer(new DenseLayer.Builder().nIn(7).nOut(3).build())
    .layer(
        new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
            .activation(Activation.SOFTMAX)
            .nIn(3)
            .nOut(3)
            .build()
      )
    .backprop(true)
    .build();
```

We use the network configuration to initialize a model which is then trained over 200 epochs.

```
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();

for (int i = 0; i < 200; i++) {
    model.fit(train);
}
```

At last, the trained model is evaluated using the test data. The last line produces a text report including classification metrics and a confusion matrix.

```
Evaluation eval = new Evaluation(3); // 3 is the number of possible classes
INDArray output = model.output(test.getFeatures());
eval.eval(test.getLabels(), output);
eval.stats();
```

Results Executing the R script will produce the following console output. Our simple model performs reasonably well in this case, misclassifying two out of 53 observations.

```
## ========================Evaluation Metrics========================
##  # of classes:    3
##  Accuracy:        0.9623
##  Precision:       0.9628
##  Recall:          0.9628
##  F1 Score:        0.9628
## Precision, recall & F1: macro-averaged (equally weighted avg. of 3 classes)
##
##
## =========================Confusion Matrix=========================
##   0  1  2
## ----------
##  17  0  0 | 0 = 0
##   0 16  1 | 1 = 1
##   0  1 18 | 2 = 2
##
## Confusion matrix format: Actual (rowClass) predicted as (columnClass) N times
## ==================================================================
```

This example demonstrated that complex Java solutions can be integrated with R using **jsr223** and standard dependency management practices.

Using other language libraries

In addition to using Java libraries, **jsr223** can easily take advantage of solutions written in other languages. In some cases, integration is as simple as sourcing a script file. For example, many common JavaScript libraries like Underscore (http://underscorejs.org) and Voca (https://vocajs.com) can be sourced using a URL. The following example sources Voca and word-wraps a string.

```
engine$source(
  "https://raw.githubusercontent.com/panzerdp/voca/master/dist/voca.min.js",
  discard.return.value = TRUE
)
engine$invokeMethod("v", "wordWrap",
  "A long sentence to wrap using Voca methods.",
  list(width = 20)
)

## [1] "A long sentence to\nwrap using Voca\nmethods."
```

Compiled Groovy and Kotlin libraries are accessed in the same way as Java libraries: simply include the relevant class or JAR files when instantiating a script engine.

Ruby gems (i.e., libraries) can also be used with **jsr223**. The **jsr223** *User Manual* provides instructions and a code example that uses a gem to generate fake entities for sample data sets.

Core Python language features are fully accessible via **jsr223**. The **jsr223** *User Manual* contains a code example that implements a simple HTTP server in Python. The Python server calls back to R to retrieve HTML content. Compatibility with some Python libraries is limited on the Java platform. Please see "Python integrations software review" for more information.

Software review

There are many integrations that combine the strengths of R with other programming languages. These language integrations can generally be classified as either *R-major* or *R-minor*. R-major integrations use R as the primary environment to control some other embedded language environment. R-minor integrations are the inverse of R-major integrations. For example, **rJava** is an R-major integration that allows Java objects to be used within an R session. The Java/R Interface (**JRI**), in contrast, is an R-minor integration that enables Java applications to embed R.

The **jsr223** package provides an R-major integration for the Java platform and several programming languages. In this software review, we provide context for the **jsr223** project through comparisons with other R-major integrations. Popular R-minor language integrations such as **Rserve** (Urbanek, 2013) and **opencpu** (Ooms, 2017a) are not included in this discussion because their objectives and features do not necessarily align with those of **jsr223**. We do, however, include a brief discussion of an R language implementation for the JVM.

Before we compare **jsr223** to other R packages, we point out one unique feature that contrasts **jsr223** with all other integrations in this discussion: **jsr223** is the only package that provides a standard interface to integrate R with multiple programming languages. This key feature enables developers to take advantage of solutions and features in several languages without the need to learn multiple integration packages.

Our software review does not include integrations for Ruby and Kotlin because **jsr223** is the only R-major integration for those languages on CRAN.

An rJava software review

As noted in the introduction, **rJava** is the preeminent Java integration for R. It provides a low-level interface to compiled Java classes via the JNI. The **jsr223** package uses **rJava** together with the Java Scripting API to create a user-friendly, multi-language integration for R and the Java platform.

The following code example is taken from **rJava**'s web site `http://www.rforge.net/rJava`. It demonstrates the essential functions of the **rJava** API by way of creating and displaying a GUI window with a single button. The first two lines are required to initialize **rJava**. The next lines use the `.jnew` function to create two Java objects: a GUI frame and a button. The associated class names are denoted in JNI syntax. Of particular note is the first invocation of `.jcall`, the function used to call object methods. In this case, the `add` method of the frame object is invoked. For **rJava** to identify the appropriate method, an explicit return type must be specified in JNI notation as the second parameter to `.jcall` (unless the return value is `void`). The last parameter to `.jcall` specifies the object to be added to the frame object. It must be explicitly cast to the correct interface for the call to be successful.

```
library("rJava")
.jinit()
f <- .jnew("java/awt/Frame", "Hello")
b <- .jnew("java/awt/Button", "OK")
.jcall(f, "Ljava/awt/Component;", "add", .jcast(b, "java/awt/Component"))
.jcall(f, , "pack")
# Show the window.
.jcall(f, , "setVisible", TRUE)
# Close the window.
.jcall(f, , "dispose")
```

The snippet below reproduces the **rJava** example above using JavaScript. In comparison, the JavaScript code is more natural for most programmers to write and maintain. The fine details of method lookups and invocation are handled automatically: no explicit class names or type casts are required. This same example can be reproduced in any of the five other **jsr223**-supported programming languages.

```
var f = new java.awt.Frame('Hello');
f.add(new java.awt.Button('OK'));
f.pack();
// Show the window.
f.setVisible(true);
// Close the window.
f.dispose();
```

Using **jsr223**, the preceding code snippet can be embedded in an R script. The first step is to create an instance of a script engine. A JavaScript engine is created as follows.

```
library(jsr223)
engine <- ScriptEngine$new("JavaScript")
```

This engine object is now ready to evaluate script on demand. Source code can be passed to the engine using character vectors or files. The sample below demonstrates embedding JavaScript code in-line with character vectors. This method is appropriate for small snippets of code. (Note: If you try this example the window may appear in the background. Also, the window must be closed using the last line of code. These are limitations of the code example, not **jsr223**.)

```
# Execute code inline to create and show the window.
engine %@% "
  var f = new java.awt.Frame('Hello');
  f.add(new java.awt.Button('OK'));
  f.pack();
  f.setVisible(true);
"

# Close the window
engine %@% "f.dispose();"
```

To execute source code in a file, use the script engine object's `source` method: `engine$source(file.name)`. The variable `file.name` may specify a local file path or a URL. Whether evaluating small code snippets or sourcing script files, embedding source code using **jsr223** is straightforward.

In comparison to **rJava**'s low-level interface, **jsr223** allows developers to use Java objects without knowing the details of JNI and method lookups. However, it is important to note that **rJava** does include a high-level interface for invoking object methods. It uses the Java reflection API to automatically locate the correct method signature. This is an impressive feature, but according to the **rJava** web site, its high-level interface is much slower than the low-level interface and it does not work correctly for all scenarios.

The **jsr223**-compatible programming languages also feature support for advanced object-oriented constructs. For example, classes can be extended and interfaces can be implemented using any language. These features allow developers to quickly implement sophisticated solutions in R without developing, compiling, and distributing custom Java classes. This can speed development and deployment significantly.

The **rJava** package supports exchanging scalars, arrays, and matrices between R and Java. The following R code demonstrates converting an R matrix to a Java object, and vice versa, using **rJava**.

```
a <- matrix(rnorm(10), 5, 2)
# Copy matrix to a Java object with rJava
o <- .jarray(a, dispatch = TRUE)
# Convert it back to an R matrix.
b <- .jevalArray(o, simplify = TRUE)
```

Again, the **jsr223** package builds on **rJava** functionality by extending data exchange. Our package converts R vectors, factors, n-dimensional arrays, data frames, lists, and environments to generic Java objects.[2] In addition, **jsr223** can convert Java scalars, n-dimensional arrays, maps, and collections to base R objects. Several data exchange options are available, including row-major and column-major ordering schemes for data frames and n-dimensional arrays.

This code snippet demonstrates data exchange using **jsr223**. The variable `engine` is a **jsr223** ScriptEngine object. Similar to the preceding **rJava** example, this code copies a matrix to the Java environment and back again. The same syntax is used for all supported data types and structures.

[2]**rJava**'s interface can theoretically support n-dimensional arrays, but currently the feature does not produce correct results for n > 2. See the related issue at the **rJava** GitHub repository: "'`.jarray(..., dispatch=T)`' on multi-dimensional arrays creates Java objects with wrong content."

```
a <- matrix(rnorm(10), 5, 2)
# Copy an R object to Java using jsr223.
engine$a <- a
# Retrieve the object.
engine$a
```

The **rJava** package does not directly support callbacks into R. Instead, callbacks are implemented through **JRI**: the Java/R Interface. The **JRI** interface is included with **rJava**. However, to use **JRI**, R must be compiled with the shared library option '`--enable-R-shlib`'. The **JRI** interface is technical and extensive. In contrast, **jsr223** supports callbacks into R using a lightweight interface that provides just three methods to execute R code, set variable values, and retrieve variable values. The **jsr223** package does not use **JRI**, so there is no requirement for R to be compiled as a shared library.

In conclusion, **jsr223** provides an alternative integration for the Java platform that is easy to learn and use.

Groovy integrations software review

Besides **jsr223**, the only other Groovy language integration available on CRAN is **rGroovy** (Fuller, 2018). It is a simple integration that uses **rJava** to instantiate `groovy.lang.GroovyShell` and pass code snippets to its `evaluate` method. We outline the typical integration approach using **rGroovy**.

Class paths must set in the global option `GROOVY_JARS` *before* loading the **rGroovy** package.

```
options(GROOVY_JARS = list("groovy-all.jar", ...))
library("rGroovy")
```

After the package is loaded, the `Initialize` function is called to instantiate an instance of the Groovy script engine that will be used to handle script evaluation. The `Initialize` function has one optional argument named `binding`. This argument accepts an **rJava** object reference to a `groovy.lang.Binding` object that represents the bindings available to the Groovy script engine. Hence, **rJava** must be used to create, set, and retrieve values in the `bindings` object. The following code example demonstrates instantiating the Groovy script engine. We initialize the script engine bindings with a variable named `myValue` that contains a vector of integers. Notice that knowledge of **rJava** and JNI notation is required to create an instance of the `bindings` object, convert the vector to a Java array, cast the resulting Java array to the appropriate interface, and finally, call the `setVariable` method of the `bindings` object.

```
bindings <- rJava::.jnew("groovy/lang/Binding")
Initialize(bindings)
myValue <- rJava::.jarray(1:3)
myValue <- rJava::.jcast(myValue, "java/lang/Object")
rJava::.jcall(bindings, "V", method = "setVariable", "myValue", myValue)
```

Finally, Groovy code can be executed using the `Evaluate` method; it returns the value of the last statement, if any. In this example, we modify the last element of our `myValue` array, and return the contents of the array.

```
script <- "
  myValue[2] = 5;
  myValue;
"
Evaluate(groovyScript = script)

## [1] 1 2 5
```

The **rGroovy** package includes another function, `Execute`, that allows developers to evaluate Groovy code without using **rJava**. However, this interface creates a new Groovy script engine instance each time it is called. In other words, it does not allow the developer to preserve state between each script evaluation.

In this code example, we demonstrate Groovy integration with **jsr223**. After the library is loaded, an instance of a Groovy script engine is created. The class path is defined at the same time the script engine is created. The variable `engine` represents the script engine instance; it exposes several methods and properties that control data exchange behavior and code evaluation. The third line creates a binding named `myValue` in the script engine's environment; the R vector is automatically converted to a Java array. The fourth line executes Groovy code that changes the last element of the `myValue` Java array before returning it to the R environment.

```
library("jsr223")
engine <- ScriptEngine$new("Groovy", "groovy-all.jar")
engine$myValue <- 1:3
engine %~% "
  myValue[2] = 5;
  myValue;
"

## [1] 1 2 5
```

In comparison to **rGroovy**, the **jsr223** implementation is more concise and requires no knowledge of **rJava** or Java classes. Though not illustrated in this example, **jsr223** can invoke Groovy functions and methods from within R, it supports callbacks from Groovy into R, and it provides extensive and configurable data exchange between Groovy and R. These features are not available in **rGroovy**.

In summary, **rGroovy** exposes a simple interface for executing Groovy code and returning a result. Data exchange is primarily handled through **rJava**, and therefore requires knowledge of **rJava** and JNI. The **jsr223** integration is more comprehensive and does not require any knowledge of **rJava**.

JavaScript integrations software review

The most prominent JavaScript integration for R is Jeroen Ooms' **V8** package (2017b). It uses the open source V8 JavaScript engine (Google developers, 2018) featured in Google's Chrome browser. We discuss the three primary differences between **V8** and **jsr223**.

First, the JavaScript engine included with **V8** provides only essential ECMAscript functionality. For example, **V8** does not include even basic file and network operations. In contrast, **jsr223** provides access to the entire JVM which includes a vast array of libraries and computing functionality.

Second, all data exchanged between **V8** and R is serialized using JSON via the **jsonlite** package (Ooms et al., 2017). JSON is very flexible; it can represent virtually any data structure. However, JSON converts all values to/from string representations which adds overhead and imposes round-off error for floating point values. The **jsr223** package handles all data using native values which reduces overhead and preserves maximum precision. In many applications, the loss of precision is not critical as far as the final numeric results are concerned, but it does require defensive programming when checking for equality. For example, an application using **V8** must round two values to a given decimal place before checking if they are equal.

The following code example demonstrates the precision issue using the R constant `pi`. The JSON conversion is handled via **jsonlite**, just as in the **V8** package. We see that after JSON conversion the value of `pi` is not identical to the original value. In contrast, the **jsr223** conversion result is identical to the original value.

```
# `digits = NA` requests maximum precision.
library("jsonlite")
identical(pi, fromJSON(toJSON(pi, digits = NA)))

## [1] FALSE

library("jsr223")
engine <- ScriptEngine$new("js")
engine$pi <- pi
identical(engine$pi, pi)

## [1] TRUE
```

The third significant difference between **V8** and **jsr223** is syntax checking. **V8** includes an interface to check JavaScript code syntax. The Java Scripting API does not provide an interface for syntax checking, hence, **jsr223** does not provide this feature. We have investigated other avenues to check syntax, but none are uniformly reliable across all of the **jsr223**-supported languages. Moreover, this feature is not critical for most integration scenarios; syntax validation is more common in applications that involve interactive code editing.

Python integrations software review

In this section, we compare **jsr223** with two Python integrations for R: **reticulate** (Allaire et al., 2018) and **rJython** (Grothendieck and Bellosta, 2012). Of the many Python integrations available for R on

CRAN, **reticulate** is the most popular as measured by monthly downloads.[3] We also discuss **rJython** because, like **jsr223**, it targets Python on the JVM.

The **reticulate** package is a very thorough Python integration for R. It includes some refined interface features that are not available in **jsr223**. For example, **reticulate** enables Python objects to be manipulated in R script using list-like syntax. One major **jsr223** feature that **reticulate** does not support is callbacks (i.e., calling R from Python). Though there are many interface differences between **jsr223** and **reticulate** (too many to list here), the most practical difference arises from their respective Python implementations. The **reticulate** package targets CPython, the reference implementation of the Python script engine. As such, **reticulate** can take advantage of the many Python libraries compiled to machine code such as Pandas (McKinney, 2010). The **jsr223** package targets the JVM via Jython, and therefore supports accessing Java objects from Python script. It cannot, however, access the Python libraries compiled to machine code because they cannot be executed by the JVM. This isn't a complete dead-end for Jython; many important Python extensions are being migrated to the JVM by the Jython Native Interface project (`http://www.jyni.org`). These extensions can easily be accessed through **jsr223**.

The **rJython** package is similar to **jsr223** in that it employs Jython. Both **jsr223** and **rJython** can execute arbitrary Python code, call Python functions and methods directly from R, use Java objects, and copy data between environments. However, there are also several important differences.

Data exchange for **rJython** can be handled via JSON or direct calls to the Jython interpreter object via **rJava**. When using **rJava** for data exchange, **rJython** is essentially limited to vectors and matrices. When using JSON for data exchange, **rJython** converts R objects to Jython structures. In contrast, the **jsr223** supports a single data exchange interface that supports all major R data structures. It uses custom Java routines that avoid the overhead and roundoff error associated with JSON conversion. Finally, **jsr223** converts R objects to generic Java structures instead of Jython objects.

JSON data exchange for **rJython** is handled by the **rjson** (Couture-Beil, 2014) package. It does not handle some R structures as one would expect. For example, n-dimensional arrays and unnamed lists are both converted to one-dimensional JSON arrays. Furthermore, **rJython** converts data frames to Jython dictionaries, but dictionaries are always returned to R as named lists.

The **jsr223** package does not exhibit these limitations; it provides predictable data exchange for all major R data structures.

Unlike **jsr223**, the **rJython** package does not return the value of the last expression when executing Python code. Instead, scripts must assign a value to a global Python variable to be fetched by another **rJython** method. This does not promote fast code exploration and prototyping. In addition, **rJython** does not supply interfaces for callbacks, script compiling, or capturing console output.

In essence, **rJython** implements a basic interface to the Jython language. The **jsr223** package, in comparison, provides a more developed feature set.

Renjin software review

Renjin (Renjin developers, 2018) is an ambitious project whose primary goal is to create a drop-in replacement for the R language on the Java platform. The Renjin solution features R syntax extensions that allow Java classes to be created and used naturally within R script. The Renjin language implementation has two important limitations: (i) it does not support plotting; and (ii) it can't use R packages that contain native libraries (like C). The **jsr223** package, in contrast, is designed for the reference distribution of R. As such, it can be used in concert with any R package.

Renjin also distributes an R package called **renjin**. It is not available from CRAN. (Find the installation instructions at `http://www.renjin.org`.) The **renjin** package exports a single method that evaluates an R expression. It is designed only to improve execution performance for R expressions; it does not allow Java classes to be used in R script. Hence, the **renjin** package is not a Java platform integration.

Overall, Renjin is a promising Java solution for R, but it is not yet feature-complete. In comparison, **jsr223** presents a viable Java solution for R today.

Summary

Java is one of the most successful development platforms in computing history. Its popularity continues as more programming languages, tools, and technologies target the JVM. The **jsr223** package provides a high-level, user-friendly interface that enables R developers to take advantage of the flourishing Java

[3]The **reticulate** package has 3,681 downloads per month according to `http://rdocumentation.org`. The next most popular Python integration is **PythonInR** (Schwendinger, 2018) with 322 monthly downloads.

ecosystem. In addition, **jsr223**'s unified integration interface for Groovy, JavaScript, Python, Ruby, and Kotlin also facilitates access to solutions developed in these languages. In all, **jsr223** significantly extends the computing capabilities of the R software environment.

In this paper, we provided an introduction to the main features and advantages of the **jsr223** package. For more language-specific examples and a full treatment of software features, see the **jsr223** *User Manual* included in the package vignettes.

Bibliography

J. Allaire, Y. Tang, and M. Geelnard. *reticulate: Interface to 'Python'*, 2018. URL https://CRAN.R-project.org/package=reticulate. R package version 1.5. [p451]

W. Chang. *R6: Classes with Reference Semantics*, 2017. URL https://CRAN.R-project.org/package=R6. R package version 2.2.2. [p442]

A. Couture-Beil. *rjson: JSON for R*, 2014. URL https://CRAN.R-project.org/package=rjson. R package version 0.2.15. [p452]

G. Csardi. *cranlogs: Download Logs from the 'RStudio' 'CRAN' Mirror*, 2015. URL https://CRAN.R-project.org/package=cranlogs. R package version 2.1.0. [p440]

D. B. Dahl. *rscala: Bi-Directional Interface Between R and Scala with Callbacks*, 2018. URL http://CRAN.R-project.org/package=rscala. R package version 2.5.1. [p442]

Eclipse Deeplearning4j Development Team. *Deeplearning4j: Open-Source Distributed Deep Learning for the JVM*, 2018. URL http://deeplearning4j.org. [p444]

T. P. Fuller. *rGroovy: Groovy Language Integration*, 2018. URL https://CRAN.R-project.org/package=rGroovy. R package version 1.2. [p450]

F. R. Gilbert and D. B. Dahl. *jdx: 'Java' Data Exchange for 'R' and 'rJava'*, 2018. URL https://CRAN.R-project.org/package=jdx. R package version 0.1.2. [p442]

Google developers. *Chrome V8*, 2018. URL https://developers.google.com/v8/. [p451]

G. Grothendieck and C. J. G. Bellosta. *rJython: R Interface to Python via Jython*, 2012. URL https://CRAN.R-project.org/package=rJython. R package version 0.0-4. [p451]

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. [p443]

W. McKinney. Data structures for statistical computing in python. In S. van der Walt and J. Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010. [p452]

M. O'Connell. Java: The inside story. *SunWorld Online*, 1995. URL http://tech-insider.org/java/research/1995/07.html. [p440]

J. Ooms. *opencpu: Producing and Reproducing Results*, 2017a. URL https://CRAN.R-project.org/package=opencpu. R package version 2.0.5. [p448]

J. Ooms. *V8: Embedded JavaScript Engine for R*, 2017b. URL https://CRAN.R-project.org/package=V8. R package version 1.5. [p442, 451]

J. Ooms, D. T. Lang, and L. Hilaiel. *jsonlite: A Robust, High Performance JSON Parser and Generator for R*, 2017. URL https://CRAN.R-project.org/package=jsonlite. R package version 1.5. [p451]

Oracle. *Java Scripting API*, 2016. URL https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/prog_guide/toc.html. [p441]

Renjin developers. *Renjin*, 2018. URL http://www.renjin.org. [p452]

F. Schwendinger. *PythonInR: Use 'Python' from Within 'R'*, 2018. URL https://CRAN.R-project.org/package=PythonInR. R package version 0.1-4. [p452]

Sun Microsystems, Inc. *JSR-223: Scripting for the Java Platform*, 2006. URL https://jcp.org/en/jsr/detail?id=223. [p441]

A. Taylor, M. Marcus, and B. Santorini. The penn treebank: An overview. In N. Ide, editor, *Text, Speech and Language Technology*, volume 20. Springer-Verlag, 2003. URL `https://doi.org/10.1007/978-94-010-0201-1_1`. [p444]

S. Urbanek. *Rserve: Binary R Server*, 2013. URL `http://CRAN.R-project.org/package=Rserve`. R package version 1.7-3. [p448]

S. Urbanek. *rJava: Low-Level R to Java Interface*, 2017. URL `https://CRAN.R-project.org/package=rJava`. R package version 0.9-9. [p440]

Floid R. Gilbert
Master's Student
Department of Statistics
Brigham Young University
Provo, UT 84602
USA
`floid.r.gilbert@gmail.com`

David B. Dahl
Professor, Graduate Coordinator, and Associate Chair
Department of Statistics
Brigham Young University
Provo, UT 84602
USA
`dahl@stat.byu.edu`

bnclassify: Learning Bayesian Network Classifiers

by Bojan Mihaljević, Concha Bielza, and Pedro Larrañaga

Abstract The **bnclassify** package provides state-of-the art algorithms for learning Bayesian network classifiers from data. For structure learning it provides variants of the greedy hill-climbing search, a well-known adaptation of the Chow-Liu algorithm and averaged one-dependence estimators. It provides Bayesian and maximum likelihood parameter estimation, as well as three naive-Bayes-specific methods based on discriminative score optimization and Bayesian model averaging. The implementation is efficient enough to allow for time-consuming discriminative scores on medium-sized data sets. The **bnclassify** package provides utilities for model evaluation, such as cross-validated accuracy and penalized log-likelihood scores, and analysis of the underlying networks, including network plotting via the **Rgraphviz** package. It is extensively tested, with over 200 automated tests that give a code coverage of 94%. Here we present the main functionalities, illustrate them with a number of data sets, and comment on related software.

Introduction

Bayesian network classifiers (Bielza and Larrañaga, 2014; Friedman et al., 1997) are competitive performance classifiers (e.g., Zaidi et al., 2013) with the added benefit of interpretability. Their simplest member, the naive Bayes (NB) (Minsky, 1961), is well-known (Hand and Yu, 2001). More elaborate models exist, taking advantage of the Bayesian network (Pearl, 1988; Koller and Friedman, 2009) formalism for representing complex probability distributions. The tree augmented naive Bayes (Friedman et al., 1997) and the averaged one-dependence estimators (AODE) (Webb et al., 2005) are among the most prominent.

A Bayesian network classifier is simply a Bayesian network applied to classification, that is, to the prediction of the probability $P(c \mid \mathbf{x})$ of some discrete (class) variable C given some features $\mathbf{X}$. The **bnlearn** (Scutari and Ness, 2018; Scutari, 2010) package already provides state-of-the art algorithms for learning Bayesian networks from data. Yet, learning classifiers is specific, as the implicit goal is to estimate $P(c \mid \mathbf{x})$ rather than the joint probability $P(\mathbf{x}, c)$. Thus, specific search algorithms, network scores, parameter estimation, and inference methods have been devised for this setting. In particular, many search algorithms consider a restricted space of structures, such as that of augmented naive Bayes (Friedman et al., 1997) models. Unlike with general Bayesian networks, it makes sense to omit a feature X_i from the model as long as the estimation of $P(c \mid \mathbf{x})$ is no better than that of $P(c \mid \mathbf{x} \setminus x_i)$. Discriminative scores, related to the estimation of $P(c \mid \mathbf{x})$ rather than $P(c, \mathbf{x})$, are used to learn both structure (Keogh and Pazzani, 2002; Grossman and Domingos, 2004; Pernkopf and Bilmes, 2010; Carvalho et al., 2011) and parameters (Zaidi et al., 2013, 2017). Some of the prominent classifiers (Webb et al., 2005) are ensembles of networks, and there are even heuristics applied at inference time, such as the lazy elimination technique (Zheng and Webb, 2006). Many of these methods (e.g., Dash and Cooper, 2002; Zaidi et al., 2013; Keogh and Pazzani, 2002; Pazzani, 1996) are, at best, just available in standalone implementations published alongside the original papers.

The **bnclassify** package implements state-of-the-art algorithms for learning structure and parameters. The implementation is efficient enough to allow for time-consuming discriminative scores on relatively large data sets. It provides utility functions for prediction and inference, model evaluation with network scores and cross-validated estimation of predictive performance, and model analysis, such as querying structure type or graph plotting via the **Rgraphviz** package (Hansen et al., 2017). It integrates with the **caret** (Kuhn et al., 2017; Kuhn, 2008) and **mlr** (Bischl et al., 2017) packages for straightforward use in machine learning pipelines. Currently it supports only discrete variables. The functionalities are illustrated in an introductory vignette, while an additional vignette provides details on the implemented methods. It includes over 200 unit and integration tests that give a code coverage of 94 percent (see `https://codecov.io/github/bmihaljevic/bnclassify?branch=master`).

The rest of this paper is structured as follows. We begin by providing background on Bayesian network classifiers (Section Background) and describing the implemented functionalities (Functionalities). We then illustrate usage with a synthetic data set (Solving a conic linear optimization problem with sdpt3r) and compare the methods' running time, predictive performance and complexity over several data sets (Properties). Finally, we discuss implementation (Implementation), briefly survey related software (Related software), and conclude by outlining future work (Conclusion).

Background

Bayesian network classifiers

A Bayesian network classifier is a Bayesian network used for predicting a discrete class variable C. It assigns $\mathbf{x}$, an observation of n predictor variables (features) $\mathbf{X} = (X_1, \ldots, X_n)$, to the most probable class:

$$c^* = \operatorname*{argmax}_c P(c \mid \mathbf{x}) = \operatorname*{argmax}_c P(\mathbf{x}, c).$$

The classifier factorizes $P(\mathbf{x}, c)$ according to a Bayesian network $\mathcal{B} = \langle \mathcal{G}, \boldsymbol{\theta} \rangle$. $\mathcal{G}$ is a directed acyclic graph with a node for each variable in $(\mathbf{X}, C)$, encoding conditional independencies: a variable X is independent of its nondescendants in $\mathcal{G}$ given the values $\mathbf{pa}(x)$ of its parents. $\mathcal{G}$ thus factorizes the joint into local (conditional) distributions over subsets of variables:

$$P(\mathbf{x}, c) = P(c \mid \mathbf{pa}(c)) \prod_{i=1}^{n} P(x_i \mid \mathbf{pa}(x_i)).$$

Local distributions $P(C \mid \mathbf{pa}(c))$ and $P(X_i \mid \mathbf{pa}(x_i))$ are specified by parameters $\boldsymbol{\theta}_{(C,\mathbf{pa}(c))}$ and $\boldsymbol{\theta}_{(X_i,\mathbf{pa}(x_i))}$, with $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{(C,\mathbf{pa}(c))}, \boldsymbol{\theta}_{(X_1,\mathbf{pa}(x_1))}, \ldots, \boldsymbol{\theta}_{(X_n,\mathbf{pa}(x_n))}\}$. It is common to assume each local distribution has a parametric form, such as the multinomial, for discrete variables, and the Gaussian for real-valued variables.

Learning structure

We learn $\mathcal{B}$ from a data set $\mathcal{D} = \{(\mathbf{x}^1, c^1), \ldots, (\mathbf{x}^N, c^N)\}$ of N observations of $\mathbf{X}$ and C. There are two main approaches to learning the structure $\mathcal{G}$ from $\mathcal{D}$: (a) testing for conditional independence among triplets of sets of variables and (b) searching a space of possible structures in order to optimize a network quality score. Under assumptions such as a limited number of parents per variable, approach (a) can produce the correct network in polynomial time (Cheng et al., 2002; Tsamardinos et al., 2003). On the other hand, finding the optimal structure–even with at most two parents per variable–is NP-hard (Chickering et al., 2004). Thus, heuristic search algorithms, such as greedy hill-climbing, are commonly used (see e.g., Koller and Friedman, 2009). Ways to reduce model complexity, in order to avoid overfitting the training data $\mathcal{D}$, include searching in restricted structure spaces and penalizing dense structures with appropriate scores.

Common scores in structure learning are the penalized log-likelihood scores, such as the Akaike information criterion (AIC) (Akaike, 1974) and Bayesian information criterion (BIC) (Schwarz, 1978). They measure the model's fitting of the empirical distribution $\widehat{P}(c, \mathbf{x})$ adding a penalty term that is a function of structure complexity. They are decomposable with respect to $\mathcal{G}$, allowing for efficient search algorithms. Yet, with limited N and a large n, discriminative scores based on $P(c \mid \mathbf{x})$, such as conditional log-likelihood and classification accuracy, are more suitable to the classification task (Friedman et al., 1997). These, however, are not decomposable according to $\mathcal{G}$. While one can add a complexity penalty to discriminative scores (e.g., Grossman and Domingos, 2004), they are instead often cross-validated to induce preference towards structures that generalize better, making their computation even more time demanding.

For Bayesian network classifiers, a common (see Bielza and Larrañaga, 2014) structure space is that of augmented naive Bayes (Friedman et al., 1997) models (see Figure 1), factorizing $P(\mathbf{X}, C)$ as

$$P(\mathbf{X}, C) = P(C) \prod_{i=1}^{n} P(X_i \mid \mathbf{Pa}(X_i)), \tag{1}$$

with $C \in \mathbf{Pa}(X_i)$ for all X_i and $\mathbf{Pa}(C) = \varnothing$. Models of different complexity arise by extending or shrinking the parent sets $\mathbf{Pa}(X_i)$, ranging from the NB (Minsky, 1961) with $\mathbf{Pa}(X_i) = \{C\}$ for all X_i, to those with a limited-size $\mathbf{Pa}(X_i)$ (Friedman et al., 1997; Sahami, 1996), to those with unbounded $\mathbf{Pa}(X_i)$ (Pernkopf and O'Leary, 2003). While the NB can only represent linearly separable classes (Jaeger, 2003), more complex models are more expressive (Varando et al., 2015). Simpler models, with sparser $\mathbf{Pa}(X_i)$, may perform better with less training data, due to their lower variance, yet worse with more data as the bias due to wrong independence assumptions will tend to dominate the error.

The algorithms that produce the above structures are generally instances of greedy hill-climbing (Keogh and Pazzani, 2002; Sahami, 1996), with arc inclusion and removal as their search operators. Some (e.g., Pazzani, 1996) add node inclusion or removal, thus embedding feature selection (Guyon and Elisseeff, 2003) within structure learning. Alternatives include the adaptation (Friedman et al.,

1997) of the Chow-Liu (Chow and Liu, 1968) algorithm to find the optimal one-dependence estimator (ODE) with respect to decomposable penalized log-likelihood scores in time quadratic in n. Some structures, such as NB or AODE, are fixed and thus require no search.

Learning parameters

Given $\mathcal{G}$, learning $\boldsymbol{\theta}$ in order to best approximate the underlying $P(C, \mathbf{X})$ is straightforward. For discrete variables X_i and $\mathbf{Pa}(X_i)$, Bayesian estimation can be obtained in closed form by assuming a Dirichlet prior over $\boldsymbol{\theta}$. With all Dirichlet hyper-parameters equal to α,

$$\theta_{ijk} = \frac{N_{ijk} + \alpha}{N_{\cdot j \cdot} + r_i \alpha}, \tag{2}$$

where N_{ijk} is the number of instances in $\mathcal{D}$ such that $X_i = k$ and $\mathbf{pa}(x_i) = j$, corresponding to the j-th possible instantiation of $\mathbf{pa}(x_i)$, $N_{\cdot j \cdot}$ is the number of instances in which $\mathbf{pa}(x_i) = j$, while r_i is the cardinality of X_i. $\alpha = 0$ in Equation 2 yields the maximum likelihood estimate of θ_{ijk}. With incomplete data, the parameters of local distributions are no longer independent and we cannot separately maximize the likelihood for each X_i as in Equation 2. Optimizing the likelihood requires a time-consuming algorithm like expectation maximization (Dempster et al., 1977) which only guarantees convergence to a local optimum.

While the NB can separate any two linearly separable classes given the appropriate $\boldsymbol{\theta}$, learning by approximating $P(C, \mathbf{X})$ cannot recover the optimal $\boldsymbol{\theta}$ in some cases (Jaeger, 2003). Several methods (Hall, 2007; Zaidi et al., 2013, 2017) learn a weight $w_i \in [0, 1]$ for each feature and then update $\boldsymbol{\theta}$ as

$$\theta_{ijk}^{weighted} = \frac{(\theta_{ijk})^{w_i}}{\sum_{k=1}^{r_i} (\theta_{ijk})^{w_i}}.$$

A $w_i < 1$ reduces the effect of X_i on the class posterior, with $w_i = 0$ omitting X_i from the model, making weighting more general than feature selection. The weights can be found by maximizing a discriminative score (Zaidi et al., 2013) or computing the usefulness of a feature in a decision tree (Hall, 2007). Mainly applied to naive Bayes models, a generalization for augmented naive Bayes classifiers has been recently developed (Zaidi et al., 2017).

Another parameter estimation method for the naive Bayes is by means of Bayesian model averaging over the 2^n possible naive Bayes structures with up to n features (Dash and Cooper, 2002). It is computed in time linear in n and provides the posterior probability of an arc from C to X_i.

Inference

Computing $P(c \mid \mathbf{x})$ for a fully observed $\mathbf{x}$ means multiplying the corresponding $\boldsymbol{\theta}$. With an incomplete $\mathbf{x}$, however, exact inference requires summing over parameters of the local distributions and is NP-hard in the general case (Cooper, 1990), yet can be tractable with limited-complexity structures. The AODE ensemble computes $P(c \mid \mathbf{x})$ as the average of the $P_i(c \mid \mathbf{x})$ of the n base models. A special case is the lazy elimination (Zheng and Webb, 2006) heuristic which omits x_i from Equation 1 if $P(x_i \mid x_j) = 1$ for some x_j.

Functionalities

The package has four groups of functionalities:

1. Learning network structure and parameters
2. Analyzing the model
3. Evaluating the model
4. Predicting with the model

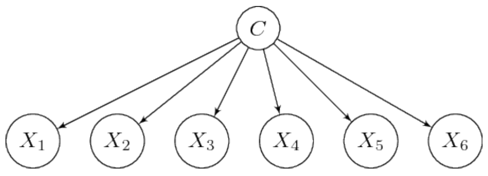
Learning is split into two separate steps, the first step is structure learning and the second, optional, step is parameter learning. The obtained models can be evaluated, used for prediction, or analyzed. The following provides a brief overview of this workflow. For details on some of the underlying methods please see the "methods" vignette.

Structures

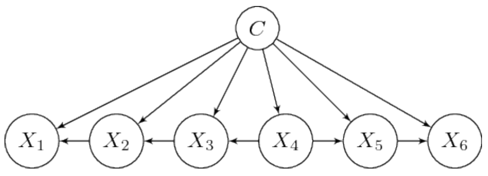
The learning algorithms produce the following network structures:

- Naive Bayes (NB) (Figure 1a) (Minsky, 1961)
- One-dependence estimators (ODE)
  - Tree-augmented naive Bayes (TAN) (Figure 1b) (Friedman et al., 1997)
  - Forest-augmented naive Bayes (FAN) (Figure 1c)
- k-dependence Bayesian classifier (k-DB) (Sahami, 1996; Pernkopf and Bilmes, 2010)
- Semi-naive Bayes (SNB)(Figure 1d) (Pazzani, 1996)
- Averaged one-dependence estimators (AODE) (Webb et al., 2005)

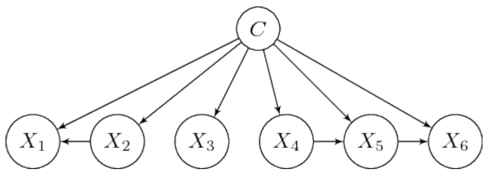
Figure 1 shows some of these structures and their factorizations of $P(c, \mathbf{x})$. We use k-DB in the sense meant by Pernkopf and Bilmes (2010) rather than that by Sahami (1996), as we impose no minimum on the number of augmenting arcs. SNB is the only structure whose complexity is not *a priori* bounded: the feature subgraph might be complete in the extreme case.



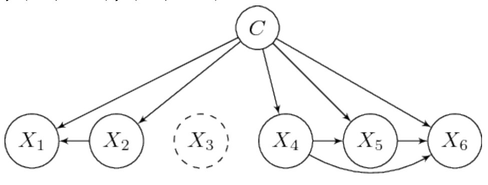
(a) $p(c, \mathbf{x}) = p(c)p(x_1|c)p(x_2|c)p(x_3|c)p(x_4|c)$ $p(x_5|c)p(x_6|c)$



(b) $p(c, \mathbf{x}) = p(c)p(x_1|c, x_2)p(x_2|c, x_3)p(x_3|c, x_4)p(x_4|c)$ $p(x_5|c, x_4)p(x_6|c, x_5)$



(c) $p(c, \mathbf{x}) = p(c)p(x_1|c, x_2)p(x_2|c)p(x_3|c)p(x_4|c)$ $p(x_5|c, x_4)p(x_6|c, x_5)$



(d) $p(c, \mathbf{x}) = p(c)p(x_1|c, x_2)p(x_2|c)p(x_4|c)$ $p(x_5|c, x_4)p(x_6|c, x_4, x_5)$

Figure 1: Augmented naive Bayes models produced by the **bnclassify** package. (a) NB; (b) TAN (c) FAN (d) SNB. k-DB and AODE not shown. The NB assumes that the features are independent given the class. ODE allows each predictor to depend on at most one other predictor: the TAN is a special case with exactly $n - 1$ augmenting arcs (i.e., inter-feature arcs) while a FAN may have less than $n - 1$. The k-DB allows for up to k parent features per feature X_i, with NB and ODE as its special cases with $k = 0$ and $k = 1$, respectively. The SNB does not restrict the number of parents but requires that connected feature subgraphs be complete (connected, after removing C, subgraphs in (d): $\{X_1, X_2\}$, and $\{X_4, X_5, X_6\}$), also allowing the removal of features (X_3 omitted in (d)). The AODE is not a single structure but an ensemble of n ODE models in which one feature is the parent of all others (a super-parent).

Algorithms

Each structure learning algorithm is implemented by a single R function. Table 1 lists these algorithms along with the corresponding structures that they produce, the scores they can be combined with, and their R functions. Below we provide their abbreviations, references, brief comments, and illustrate function calls.

Fixed structure

We implement two algorithms:

- NB
- AODE

The NB and AODE structures are fixed given the number of variables, and thus no search is required to estimate them from data. For example, we can get a NB structure with

```
n <- nb('class', dataset = car)
```

where `class` is the name of the class variable C and `car` the dataset containing observations of C and $\mathbf{X}$.

Optimal ODEs with decomposable scores

We implement one algorithm:

- Chow-Liu for ODEs (CL-ODE; Friedman et al. (1997))

Maximizing log-likelihood will always produce a TAN while maximizing penalized log-likelihood may produce a FAN since including some arcs can degrade such a score. With incomplete data our implementation does not guarantee the optimal ODE as that would require computing maximum likelihood parameters. The arguments of the `tan_cl()` function are the network score to use and, optionally, the root for features' subgraph:

```
n <- tan_cl('class', car, score = 'AIC', root = 'buying')
```

Greedy hill-climbing with global scores

The **bnclassify** package implements five algorithms:

- Hill-climbing tree augmented naive Bayes (HC-TAN) (Keogh and Pazzani, 2002)
- Hill-climbing super-parent tree augmented naive Bayes (HC-SP-TAN) (Keogh and Pazzani, 2002)
- Backward sequential elimination and joining (BSEJ) (Pazzani, 1996)
- Forward sequential selection and joining (FSSJ) (Pazzani, 1996)
- Hill-climbing k-dependence Bayesian classifier (k-DB)

These algorithms use the cross-validated estimate of predictive accuracy as a score. Only the FSSJ and BSEJ perform feature selection. The arguments of the corresponding functions include the number of cross-validation folds, `k`, and the minimal absolute score improvement, `epsilon`, required for continuing the search:

```
fssj <- fssj('class', car, k = 5, epsilon = 0)
```

Structure	Search algorithm	Score	Feature selection	Function
NB	-	-	-	`nb`
TAN/FAN	CL-ODE	log-lik, AIC, BIC	-	`tan_cl`
TAN	TAN-HC	accuracy	-	`tan_hc`
TAN	TAN-HCSP	accuracy	-	`tan_hcsp`
SNB	FSSJ	accuracy	forward	`fssj`
SNB	BSEJ	accuracy	backward	`bsej`
AODE	-	-	-	`aode`
kDB	kDB	accuracy	-	`kdb`

Table 1: Implemented structure learning algorithms.

Parameters

The **bnclassify** package only handles discrete features. With fully observed data, it estimates the parameters with maximum likelihood or Bayesian estimation, according to Equation 2, with a single α for all local distributions. With incomplete data it uses *available case analysis* and substitutes $N_{\cdot j \cdot}$ in Equation 2 with $N_{ij\cdot} = \sum_{k=1}^{r_i} N_{ijk}$, i.e., with the count of instances in which $\mathbf{Pa}(X_i) = j$ and X_i is observed.

We implement two methods for weighted naive Bayes parameter estimation:

- Weighting attributes to alleviate naive Bayes' independence assumption (WANBIA) (Zaidi et al., 2013)
- Attribute-weighted naive Bayes (AWNB) (Hall, 2007)

We implement one method for estimation by means of Bayesian model averaging over all NB structures with up to n features:

- Model averaged naive Bayes (MANB) (Dash and Cooper, 2002)

It makes little sense to apply WANBIA, MANB, and AWNB to structures other than NB. WANBIA, for example, learns the weights by optimizing the conditional log-likelihood of the NB. Parameter learning is done with the `lp()` function. For example,

```
a <- lp(n, smooth = 1, manb_prior = 0.5)
```

computes Bayesian parameter estimates with $\alpha = 1$ (the `smooth` argument) for all local distributions, and updates them with the MANB estimation obtained with a 0.5 prior probability for each class-to-feature arc.

Utilities

Single-structure-learning functions, as opposed to those that learn an ensemble of structures, return an S3 object of class `"bnc_dag"`. The following functions can be invoked on such objects:

- Plot the network: `plot()`
- Query model type: `is_tan()`, `is_ode()`, `is_nb()`, `is_aode()`, ...
- Query model properties: `narcs()`, `families()`, `features()`, ...
- Convert to a **gRain** object: `as_grain()`

Ensembles are of type `"bnc_aode"` and only `print()` and model type queries can be applied to such objects. Fitting the parameters (by calling `lp()`) of a `"bnc_dag"` produces a `"bnc_bn"` object. In addition to all `"bnc_dag"` functions, the following are meaningful:

- Predict class labels and class posterior probability: `predict()`
- Predict joint distribution: `compute_joint()`
- Network scores: `AIC()`,`BIC()`,`logLik()`,`clogLik()`
- Cross-validated accuracy: `cv()`
- Query model properties: `nparams()`
- Parameter weights: `manb_arc_posterior()`, `weights()`

The above functions for `"bnc_bn"` can also be applied to an ensemble with fitted parameters.

Documentation

This vignette provides an overview of the package and background on the implemented methods. Calling `?bnclassify` gives a more concise overview of the functionalities, with pointers to relevant functions and their documentation. The "usage" vignette presents more detailed usage examples and shows how to combine the functions. The "methods" vignette provides details on the underlying methods and documents implementation specifics, especially where they differ from or are undocumented in the original paper.

Usage example

The available functionalities can be split into four groups:

1. Learning network structure and parameters
2. Analyzing the model
3. Evaluating the model
4. Predicting with the model

We illustrate these functionalities with the synthetic `car` data set with six features. We begin with a simple example for each functionality group and then elaborate on the options in the following sections. We first load the package and the dataset:

```
library(bnclassify)
data(car)
```

Then we learn a naive Bayes structure and its parameters:

```
nb <- nb('class', car)
nb <- lp(nb, car, smooth = 0.01)
```

Then we get the number of arcs in the network:

```
narcs(nb)
```

```
[1] 6
```

Then we get the 10-fold cross-validation estimate of accuracy:

```
cv(nb, car, k = 10)
```

```
[1] 0.8628258
```

Finally, we classify the entire data set:

```
p <- predict(nb, car)
head(p)
```

```
[1] unacc unacc unacc unacc unacc unacc
Levels: unacc acc good vgood
```

Learning

The functions for structure learning, shown in Table 1, correspond to the different algorithms. They all receive the name of the class variable and the data set as their first two arguments, which are then followed by optional arguments. The following runs the CL-ODE algorithm with the AIC score, followed by the FSSJ algorithm to learn another model:

```
ode_cl_aic <- tan_cl('class', car, score = 'aic')
set.seed(3)
fssj <- fssj('class', car, k = 5, epsilon = 0)
```

The bnc() function is a shorthand for learning structure and parameters in a single step,

```
ode_cl_aic <- bnc('tan_cl', 'class', car, smooth = 1, dag_args = list(score = 'aic'))
```

where the first argument is the name of the structure learning function while and optional arguments go in dag_args.

Analyzing

Printing the model, such as the above ode_cl_aic object, provides basic information about it.

```
ode_cl_aic
```

```

  Bayesian network classifier

  class variable:        class
  num. features:   6
  num. arcs:   9
  free parameters:   131
  learning algorithm:    tan_cl
```

While plotting the network is especially useful for small networks, printing the structure in the **deal** (Bottcher and Dethlefsen, 2013) and **bnlearn** format may be more useful for larger ones:

```
ms <- modelstring(ode_cl_aic)
strwrap(ms, width = 60)
```

```
[1] "[class] [buying|class] [doors|class] [persons|class]"
[2] "[maint|buying:class] [safety|persons:class]"
[3] "[lug_boot|safety:class]"
```

We can query the type of structure–params() lets us access the conditional probability tables (CPTs), while features() lists the features:

```
is_ode(ode_cl_aic)
```

```
[1] TRUE
```

```
params(nb)$buying

       class
buying         unacc          acc         good        vgood
  low   0.2132243562 0.2317727320 0.6664252607 0.5997847478
  med   0.2214885458 0.2994740131 0.3332850521 0.3999077491
  high  0.2677680077 0.2812467451 0.0001448436 0.0001537515
  vhigh 0.2975190903 0.1875065097 0.0001448436 0.0001537515

length(features(fssj))

[1] 5
```

For example, `fssj()` has selected five out of six features.

The `manb_arc_posterior()` function provides the MANB posterior probabilities for arcs from the class to each of the features:

```
manb <- lp(nb, car, smooth = 0.01, manb_prior = 0.5)
round(manb_arc_posterior(manb))

  buying    maint    doors  persons lug_boot   safety
       1        1        0        1        1        1
```

With the posterior probability of 0% for the arc from `class` to `doors`, and 100% for all others, MANB renders `doors` independent from the class while leaving the other features' parameters unaltered. We can see this by printing out the CPTs:

```
params(manb)$doors

       class
doors   unacc  acc good vgood
  2      0.25 0.25 0.25  0.25
  3      0.25 0.25 0.25  0.25
  4      0.25 0.25 0.25  0.25
  5more  0.25 0.25 0.25  0.25

all.equal(params(manb)$buying, params(nb)$buying)

[1] TRUE
```

For more functions for querying a structure with parameters (`"bnc_bn"`) see `?inspect_bnc_bn`. For a structure without parameters (`"bnc_dag"`), see `?inspect_bnc_dag`.

Evaluating

Several scores can be computed:

```
logLik(ode_cl_aic, car)

'log Lik.' -13307.59 (df=131)

AIC(ode_cl_aic, car)

[1] -13438.59
```

The `cv()` function estimates the predictive accuracy of one or more models with a single run of stratified cross-validation. In the following we assess the above models produced by NB and CL-ODE algorithms:

```
set.seed(0)
cv(list(nb = nb, ode_cl_aic = ode_cl_aic), car, k = 5, dag = TRUE)

        nb ode_cl_aic
 0.8582303  0.9345913
```

Above, `k` is the desired number of folds, and `dag = TRUE` evaluates structure and parameter learning, while `dag = FALSE` keeps the structure fixed and evaluates just the parameter learning. The output gives 86% and 93% accuracy estimates for NB and CL-ODE, respectively. The **mlr** and **caret** packages provide additional options for evaluating predictive performance, such as different metrics, and **bnclassify** is integrated with both (see the "usage" vignette).

Predicting

As shown above, we can predict class labels with `predict()`. We can also get the class posterior probabilities:

```
pp <- predict(nb, car, prob = TRUE)
# Show class posterior distributions for the first six instances of car
head(pp)
```

```
         unacc          acc         good        vgood
[1,] 1.0000000 2.171346e-10 8.267214e-16 3.536615e-19
[2,] 0.9999937 6.306269e-06 5.203338e-12 5.706038e-19
[3,] 0.9999908 9.211090e-06 5.158884e-12 4.780777e-15
[4,] 1.0000000 3.204714e-10 1.084552e-15 1.015375e-15
[5,] 0.9999907 9.307467e-06 6.826088e-12 1.638219e-15
[6,] 0.9999864 1.359469e-05 6.767760e-12 1.372573e-11
```

Properties

We illustrate the algorithms' running times, resulting structure complexity and predictive performance on the datasets listed in Table 2. We only used complete data sets as time-consuming inference with incomplete data makes cross-validated scores costly for medium-sized or large data sets. The structure and parameter learning methods are listed in the legends of Figure 2, Figure 3, and Figure 4.

N	n	r_c	Dataset
1728	7	4	car
958	10	2	tic-tac-toe
435	17	2	voting
351	35	2	ionosphere
562	36	19	soybean
3196	37	2	kr-vs-kr
3190	61	3	splice

Table 2: Data sets used, from the UCI repository (Lichman, 2013). Incomplete rows have been removed. The number of classes (i.e., distinct class labels) is r_c.

Figure 2 shows that the algorithms with cross-validated scores, followed by WANBIA, are the most time-consuming. Running time is still not prohibitive: TAN-HC ran for 139 seconds on kr-vs-kp and 282 seconds on splice, adding 27 augmenting arcs on the former and 7 on the latter (a added arcs mean a iterations of the search algorithm). Note that their running time is linear in the number of cross-validation folds k; using $\mathsf{k} = 10$ instead of $\mathsf{k} = 5$ would have roughly doubled the time.

CL-ODE tended to produce the most complex structures (see Figure 3), with FSSJ learning complex models on car, soybean and splice, yet simple ones, due to feature selection, on voting and tic-tac-toe. The NB models with alternative parameters, WANBIA and MANB, have as many parameters as the NB, because we are not counting the length-n weights vector, rather just the parameters $\boldsymbol{\theta}$ of the resulting NB (the weights simply produce an alternative parameterization of the NB).

In terms of accuracy, NB and MANB performed comparatively poorly on car, voting, tic-tac-toe, and kr-vs-kp, possibly because of many wrong independence assumptions (see Figure 4). WANBIA may have accounted for some of these violations on voting and kr-vs-kp, as it outperformed NB and MANB on these datasets, showing that a simple model can perform well on them when adequately parameterized. More complex models, such as CL-ODE and AODE, performed better on `car`.

Implementation

With complete data, **bnclassify** implements prediction for augmented naive Bayes models as well as for ensembles of such models. It multiplies the corresponding $\boldsymbol{\theta}$ in logarithmic space, applying the *log-sum-exp* trick before normalizing, to reduce the chance of underflow. On instances with missing entries, it uses the **gRain** package (Højsgaard, 2016, 2012) to perform exact inference, which is noticeably slower. Network plotting is implemented by the **Rgraphviz** package. Some functions are implemented in C++ with **Rcpp** for efficiency. The package is extensively tested, with over 200 unit and integrated tests that give a 94% code coverage.

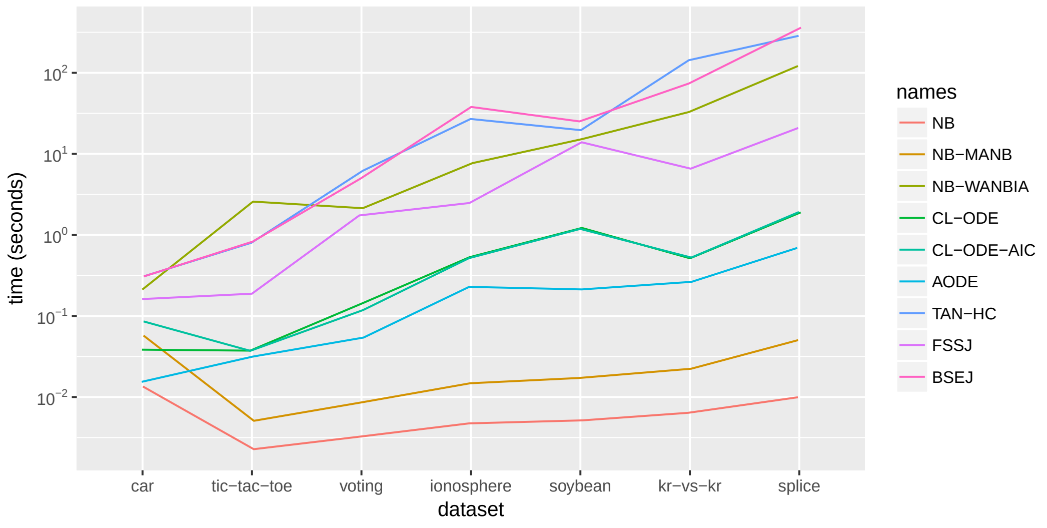


Figure 2: Running times of the algorithms on a Ubuntu 16.04 machine with 8 GB of RAM and a 2.5 GHz processor, on a $\log_{10}$ scale. We used the default options for all algorithms and `k = 5` and `epsilon = 0` for the wrappers. CL-ODE-AIC is CL-ODE with the AIC rather than the log-likelihood score. The lines have been horizontally and vertically jittered to avoid overlap where identical.

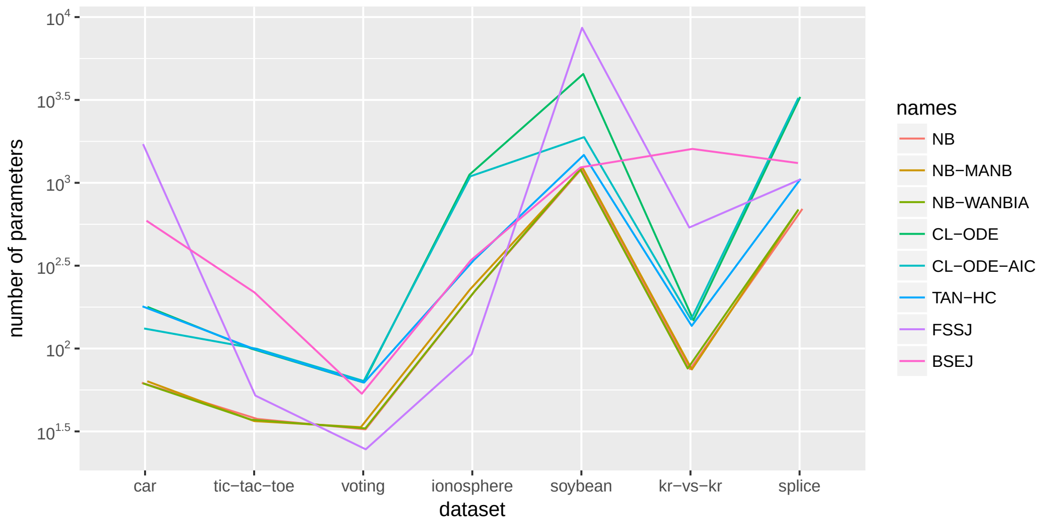


Figure 3: The number of Bayesian network parameters $\boldsymbol{\theta}$ of the resulting structures, on a $\log_{10}$ scale. The lines have been horizontally and vertically jittered to avoid overlap where identical.

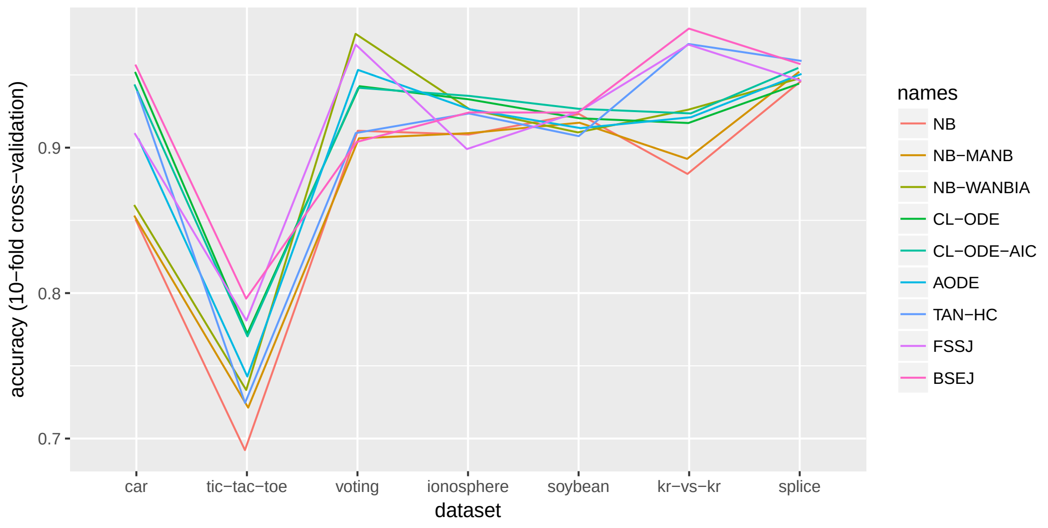


Figure 4: Accuracy of the algorithms estimated with stratified 10-fold cross-validation. The lines have been horizontally and vertically jittered to avoid overlap where identical.

Related software

NB, TAN, and AODE are available in general-purpose tools such as **bnlearn** and Weka. WANBIA (`https://sourceforge.net/projects/rawnaivebayes`) and MANB (`http://www.dbmi.pitt.edu/content/manb`) are only available in stand-alone software, published along with the original publications. We are not aware of available implementations of the remaining methods implemented in **bnclassify**.

The **bnlearn** package implements algorithms for learning general purpose Bayesian networks. Among them, algorithms for Markov blanket learning by testing for independencies, such as IAMB (Tsamardinos and Aliferis, 2003) and GS (Margaritis and Thrun, 2000), can be very useful for classification as they can look for the Markov blanket of the class variable. The **bnlearn** package combines the search algorithms, such as greedy hill-climbing and tabu search (Glover and Laguna, 2013), only with generative scores such as penalized log-likelihood. Among classification models, it implements the discrete NB and CL-ODE. It does not handle incomplete data and provides cross-validation and prediction only for the NB and TAN models, but not for the unrestricted Bayesian networks.

Version 3.8 of Weka (Hall et al., 2009; Bouckaert, 2004) provides variants of the AODE (Webb et al., 2005) as well as the CL-ODE and NB. It implements five additional search algorithms, such as K2 (Cooper and Herskovits, 1992), tabu search, and simulated annealing (Kirkpatrick et al., 1983), combining them only with generative scores. Except for the NB, Weka only handles discrete data and uses simple imputation (replacing with the mode or mean) to handle incomplete data. It provides two constraint-based algorithms, but performs conditional independence tests in an ad-hoc way (Bouckaert, 2004). Weka provides Bayesian model averaging for parameter estimation (Bouckaert, 1995).

The Java library jBNC (`http://jbnc.sourceforge.net/`, version 1.2.2) learns ODE classifiers from Sacha et al. (2002) by optimizing penalized log-likelihood or the cross-validated estimate of accuracy. The CGBayes (version 7.14.14) package (McGeachie et al., 2014) for MATLAB implements conditional Gaussian networks (Lauritzen and Wermuth, 1989). It provides four structure learning algorithms, including a variant of K2 and a greedy hill-climber, all optimizing the marginal likelihood of the data given the network.

Conclusion

The **bnclassify** package implements several state-of-the art algorithms for learning Bayesian network classifiers. It also provides features such as model analysis and evaluation. It is reasonably efficient and can handle large data sets. We hope that **bnclassify** will be useful to practitioners as well as researchers wishing to compare their methods to existing ones.

Future work includes handling real-valued feature via conditional Gaussian models. Straightforward extensions include adding flexibility to the hill-climbing algorithm, such as restarts to avoid local minima.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 785907 (HBP SGA2), the Spanish Ministry of Economy and Competitiveness through the Cajal Blue Brain (C080020-09; the Spanish partner of the EPFL Blue Brain initiative) and TIN2016-79684-P projects, from the Regional Government of Madrid through the S2013/ICE-2845-CASI-CAM-CM project, and from Fundación BBVA grants to Scientific Research Teams in Big Data 2016.

Bibliography

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. [p456]

C. Bielza and P. Larrañaga. Discrete Bayesian network classifiers: A survey. *ACM Computing Surveys*, 47(1), 2014. [p455, 456]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, Z. Jones, G. Casalicchio, and M. Gallo. *mlr: Machine Learning in R*, 2017. URL `https://CRAN.R-project.org/package=mlr`. R package version 2.11. [p455]

S. G. Bottcher and C. Dethlefsen. *deal: Learning Bayesian Networks with Mixed Variables*, 2013. URL `https://CRAN.R-project.org/package=deal`. R package version 1.2-37. [p461]

R. Bouckaert. Bayesian network classifiers in Weka. Technical Report 14/2004, Department of Computer Science, University of Waikato, 2004. [p465]

R. R. Bouckaert. *Bayesian Belief Networks: From Construction to Inference*. PhD thesis, Universiteit Utrecht, 1995. [p465]

A. M. Carvalho, T. Roos, A. L. Oliveira, and P. Myllymäki. Discriminative learning of Bayesian networks via factorized conditional log-likelihood. *Journal of Machine Learning Research*, 12:2181–2210, 2011. [p455]

J. Cheng, R. Greiner, J. Kelly, D. A. Bell, and W. Liu. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137:43–90, 2002. [p456]

D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004. [p456]

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968. [p457]

G. F. Cooper. The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks. *Artificial Intelligence*, 42(2-3):393–405, 1990. [p457]

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992. [p465]

D. Dash and G. F. Cooper. Exact model averaging with naive Bayesian classifiers. In *19th International Conference on Machine Learning (ICML-2002)*, pages 91–98, 2002. [p455, 457, 459]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. [p457]

N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29: 131–163, 1997. [p455, 456, 458, 459]

F. Glover and M. Laguna. Tabu Search. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 3261–3362. Springer-Verlag, New York, NY, 2013. [p465]

D. Grossman and P. Domingos. Learning Bayesian Network Classifiers by Maximizing Conditional Likelihood. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 361–368, 2004. [p455, 456]

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. [p456]

M. Hall. A decision tree-based attribute weighting filter for naive Bayes. *Knowledge-Based Systems*, 20 (2):120–126, 2007. [p457, 459]

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. [p465]

D. J. Hand and K. Yu. Idiot's Bayes - Not so Stupid after All? *International Statistical Review*, 69(3): 385–398, 2001. [p455]

K. D. Hansen, J. Gentry, L. Long, R. Gentleman, S. Falcon, F. Hahne, and D. Sarkar. *Rgraphviz: Provides Plotting Capabilities for R Graph Objects*, 2017. URL `https://doi.org/10.18129/B9.bioc.Rgraphviz`. R package version 2.20.0. [p455]

S. Højsgaard. Graphical independence networks with the `gRain` package for `R`. *Journal of Statistical Software*, 46(10):1–26, 2012. [p463]

S. Højsgaard. *gRain: Graphical Independence Networks*, 2016. URL `https://CRAN.R-project.org/package=gRain`. R package version 1.3-0. [p463]

M. Jaeger. Probabilistic classifiers and the concept they recognize. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*, pages 266–273, 2003. [p456, 457]

E. J. Keogh and M. J. Pazzani. Learning the structure of augmented Bayesian classifiers. *International Journal on Artificial Intelligence Tools*, 11(4):587–601, 2002. [p455, 456, 459]

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598): 671–680, 1983. [p465]

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, Cambridge, MA, USA, 2009. [p455, 456]

M. Kuhn. Building predictive models in `R` using the `caret` package. *Journal of Statistical Software*, 28(5): 1–26, 2008. [p455]

M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, R. Lescarbeau, A. Ziem, L. Scrucca, Y. Tang, C. Candan, and T. Hunt. *caret: Classification and Regression Training*, 2017. URL `https://CRAN.R-project.org/package=caret`. R package version 6.0-78. [p455]

S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17(1):31–57, 1989. [p465]

M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`. [p463]

D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. In *Advances in Neural Information Processing Systems 12*, pages 505–511. MIT Press, 2000. [p465]

M. J. McGeachie, H.-H. Chang, and S. T. Weiss. CGBayesNets: Conditional Gaussian Bayesian network learning and inference with mixed discrete and continuous data. *PLoS Computational Biology*, 10(6): e1003676, 2014. [p465]

M. Minsky. Steps toward artificial intelligence. *Transactions on Institute of Radio Engineers*, 49:8–30, 1961. [p455, 456, 458]

M. Pazzani. Constructive induction of Cartesian product attributes. In *Proceedings of the Information, Statistics and Induction in Science Conference*, pages 66–77, 1996. [p455, 456, 458, 459]

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, CA, USA, 1988. [p455]

F. Pernkopf and J. A. Bilmes. Efficient heuristics for discriminative structure learning of Bayesian network classifiers. *Journal of Machine Learning Research*, 11:2323–2360, 2010. [p455, 458]

F. Pernkopf and P. O'Leary. Floating search algorithm for structure learning of Bayesian network classifiers. *Pattern Recognition Letters*, 24(15):2839–2848, 2003. [p456]

J. P. Sacha, L. S. Goodenday, and K. J. Cios. Bayesian learning for cardiac spect image interpretation. *Artificial Intelligence in Medicine*, 26(1):109–143, 2002. [p465]

M. Sahami. Learning limited dependence Bayesian classifiers. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-1996)*, volume 96, pages 335–338, 1996. [p456, 458]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. [p456]

M. Scutari. Learning Bayesian networks with the `bnlearn` R package. *Journal of Statistical Software*, 35 (3):1–22, 2010. [p455]

M. Scutari and R. Ness. *Bnlearn: Bayesian Network Structure Learning, Parameter Learning and Inference*, 2018. URL `https://CRAN.R-project.org/package=bnlearn`. R package version 4.3. [p455]

I. Tsamardinos and C. F. Aliferis. Towards principled feature selection: Relevancy, filters and wrappers. In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*. Morgan Kaufmann Publishers: Key West, FL, USA, 2003. [p465]

I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Algorithms for large scale Markov blanket discovery. In *Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2003)*, pages 376–381. AAAI Press, 2003. [p456]

G. Varando, C. Bielza, and P. Larrañaga. Decision boundary for discrete Bayesian network classifiers. *Journal of Machine Learning Research*, 16:2725–2749, 2015. [p456]

G. I. Webb, J. R. Boughton, and Z. Wang. Not so Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning*, 58(1):5–24, 2005. [p455, 458, 465]

N. A. Zaidi, J. Cerquides, M. J. Carman, and G. I. Webb. Alleviating naive Bayes attribute independence assumption by attribute weighting. *Journal of Machine Learning Research*, 14:1947–1988, 2013. [p455, 457, 459]

N. A. Zaidi, G. I. Webb, M. J. Carman, F. Petitjean, W. Buntine, M. Hynes, and H. De Sterck. Efficient Parameter Learning of Bayesian Network Classifiers. *Machine Learning*, 106(9):1289–1329, 2017. [p455, 457]

F. Zheng and G. I. Webb. Efficient lazy elimination for averaged one-dependence estimators. In *Proceedings of the 23rd International Conference on Machine Learning*, volume 148, pages 1113–1120. ACM, 2006. [p455, 457]

Bojan Mihaljević
Computational Intelligence Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid
Boadilla del Monte, 28660, Spain
ORCiD: 0000-0002-1656-6135
`bmihaljevic@fi.upm.es`

Concha Bielza
Computational Intelligence Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid
Boadilla del Monte, 28660, Spain
ORCiD: 0000-0001-7109-2668
`mcbielza@fi.upm.es`

Pedro Larrañaga
Computational Intelligence Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid
Boadilla del Monte, 28660, Spain
ORCiD: 0000-0003-0652-9872
`pedro.larranaga@fi.upm.es`

Dynamic Simulation and Testing for Single-Equation Cointegrating and Stationary Autoregressive Distributed Lag Models

by Soren Jordan and Andrew Q. Philips

Abstract While autoregressive distributed lag models allow for extremely flexible dynamics, interpreting the substantive significance of complex lag structures remains difficult. In this paper we discuss **dynamac** (dynamic autoregressive and cointegrating models), an `R` package designed to assist users in estimating, dynamically simulating, and plotting the results of a variety of autoregressive distributed lag models. It also contains a number of post-estimation diagnostics, including a test for cointegration for when researchers are estimating the error-correction variant of the autoregressive distributed lag model.

Introduction

Many important social processes vary systematically over time. Models designed to account for such dynamics have become more common in the social sciences. Applied examples range from prime ministerial approval (Clarke et al., 2000), to the determinants of prison admission rates (Jacobs and Helms, 1996), to preferences for different types of government spending (Wlezien, 1995), to how changing tax rates affect income (Mertens and Montiel Olea, 2018).

Dynamic processes take many forms. Some of these processes are integrated, such that the current value ("realization") of a series is an expression of all past values in the series plus some new innovation (i.e. $y_t \sim I(1)$ if $y_t = y_{t-1} + \epsilon_t$). These series are referred to as I(1), non-stationary, or having a "unit root."[1] Other processes revert back to some mean level over time. Since they are not integrated, often these series are termed stationary, or I(0). Stationary series are characterized by constant mean, variance, and covariance; non-stationary series violate one or more of these properties. An especially interesting relationship exists when two (or more) integrated variables are in a long-run relationship with each other, even if in the short term the series may move apart. Such series are said to be cointegrated.[2]

The job of properly modeling such a diverse set of dynamic processes is challenging. Researchers want to test whether a theoretical regressor of interest, X, has an effect on a dependent variable y, accounting for each series' own dynamics. Moreover, these effects might occur in both the short-run and long-run of the series. Additionally, in cases where we have multiple I(1) variables, we must test whether the variables are in a cointegrating relationship. Researchers still need a way to interpret the effects of these complex models. When modeling dynamics, then, we might be particularly drawn to a strategy that contributes to each of these goals.

We advocate for one particular specification because of its generalizability, flexibility and robustness to a variety of data-generating proceeses: autoregressive distributed lag (ARDL) models. ARDL models can account for multiple lags of independent variables, either in levels or in first-differences, as well as multiple lags of the dependent variable. Moreover, utilizing the ARDL-bounds testing procedure of Pesaran et al. (2001), we can test whether variables in the ARDL model are also in a cointegrating relationship. However, the very flexibility of this modeling strategy produces its own challenges. If the researcher includes multiple lags (or differences, or lagged differences) of some independent variable, it becomes increasingly difficult to discern various effects of a change in X on y, beyond the individual coefficients estimated in the ARDL model. These are more complicated than static models (De Boef and Keele, 2008), since they often involve both short- and long-run effects of X on y.[3]

We introduce a new `R` package, **dynamac** (Jordan and Philips, 2018), to help ameliorate these two challenges. The core of the package consists of two functions: `pssbounds`, which implements

[1]A unit-root is one common form of non-stationarity; a series could be non-stationary due to a linear trend, for instance.

[2]Stationary variables cannot be in cointegrating relationships, as their long-run response is to return to their mean value, not a linear combination of some other series.

[3]See Beck and Katz (2011) or Blackwell and Glynn (2018) for a discussion of this in the cross-sectional time series context.

the ARDL-bounds testing procedure for cointegration (Pesaran et al., 2001), and `dynardl`, which estimates ARDL models and simulates the effect of some X on y by way of dynamic simulations. The simulations provided by the latter function help provide substantive inferences of some X on y (1) in both the short run *and* long run, (2) when X appears in multiple forms (such as first-differences, lagged first-differences, and/or lagged levels), and (3) accounting for similar specifications of other control variables Z. We also include functions to easily take the lag, first-difference, and lagged-first difference of a series, as well as post-estimation diagnostics users can employ to ensure residuals from their resulting model are white noise. In doing so, we complement other `R` packages designed to show substantive and statistical significance of dynamic models (for instance, the **dynsim** package, Gandrud et al., 2015, 2016), as well as broader classes of models such as those compatible with **Zelig** (Choirat et al., 2018).

Below we explain the context of ARDL models generally, as well as cointegration testing and the ARDL-bounds test in particular. We then discuss **dynamac** functions to help estimate both the ARDL-bounds test as well as the ARDL models (and their stochastic simulations). We provide illustrative examples of each function, and conclude by offering suggestions for future research.

The general ARDL model

The autoregressive distributed lag model has a very general form:

$$\begin{aligned} y_t, \Delta y_t =& \alpha_0 + \delta T + \sum_{p=1}^{P} \phi_p y_{t-p} + \sum_{l_1=0}^{L_1} \theta_{1l} x_{1,t-l_1} + \cdots + \sum_{l_k=0}^{L_k} \theta_{kl} x_{t-l_k} + \\ & \sum_{m=1}^{M} \alpha_m \Delta y_{t-m} + \sum_{q_1=0}^{Q_1} \beta_{1q_1} \Delta x_{1,t-q_1} + \cdots + \sum_{q_k=0}^{Q_k} \beta_{kq_k} \Delta x_{k,t-q_k} + \epsilon_t \end{aligned} \tag{1}$$

The left-hand side can be estimated either in levels (y_t) or in first-differences (Δy_t). This may be a function of a constant, a linear trend, up to p lags of the dependent variable, a series of lags for each of the k regressors—appearing either contemporaneously or with a lag—m lagged first-differences of the dependent variable, and contemporaneous and/or lagged first-differences of each of the regressors. Lagged first-differences may enter into Equation 1 for theoretical reasons, or to help ensure that the resulting residuals ϵ are white noise.[4] Since this model obviously results in a heavy parameterization, restrictions are often imposed. Two of the most common restrictions are the ARDL(1,1) model with all-stationary data:

$$y_t = \alpha_0 + \phi_1 y_{t-1} + \theta_{1,0} x_{1,t} + \cdots + \theta_{k,0} x_{k,t} + \theta_{1,1} x_{1,t-1} + \cdots + \theta_{k,1} x_{k,t-1} + \epsilon_t \tag{2}$$

as well as its non-stationary and cointegrating variant, often referred to as an error-correction model:[5]

$$\Delta y_t = \alpha_0 + \phi_1 y_{t-1} + \theta_{1,1} x_{1,t-1} + \cdots + \theta_{k,1} x_{k,t-1} + \beta_1 \Delta x_{1,t} + \cdots + \beta_k \Delta x_{k,t} + \epsilon_t \tag{3}$$

For clarity, lagged first-differences are not shown, although they may be added to Equations 2 and 3, either for theoretical reasons or to purge autocorrelation from the residuals.[6]

The strength and drawback of the ARDL model should be immediately apparent: while researchers can specify a variety of lagged levels and differences to account for theory as well as ensure white-noise (random) residuals (that is, no residual autocorrelation), the same flexibility can make inferences regarding the total effect of any individual X variable difficult to discern. This is exacerbated even more when we consider that these variables might have short-run and long-run effects. In other words, an immediate change in an X variable might have an immediate impact in the contemporaneous period t, but if lagged values of X also appear in the model, this effect will persist for multiple time periods. Moreover, if the dependent variable is entered into the model in lagged levels through ϕ_p, the effects of the independent variable X in some time period t persist over time. There is a cumulative—or long-run—effect across time, given a change in X in a single time period. In simple cases this can be calculated as a non-linear combination of the lagged parameter estimate and the parameter on the lagged dependent variable (De Boef and Keele, 2008). Yet even these calculations become tedious with multiple lags of the regressors and the dependent variable.

We circumvent this difficulty by employing stochastic simulations rather than direct interpretation of coefficients in order to show statistical and substantive significance. Dynamic stochastic simulations

[4] i.e., $\epsilon \sim N(0, \sigma^2)$. We discuss the importance of white-noise residuals below.

[5] Of course, all-stationary series may be estimated in an error-correction model as well (De Boef and Keele, 2008).

[6] i.e., users could add: $\sum_{m=1}^{M} \alpha_m \Delta y_{t-m} + \sum_{q_1=0}^{Q_1} \beta_{1q_1} \Delta x_{1,t-q_1} + \cdots + \sum_{q_k=0}^{Q_k} \beta_{kq_k} \Delta x_{k,t-q_k}$.

provide an alternative to direct hypothesis testing of coefficients, instead focusing on simulating meaningful counterfactuals from model coefficients many times and drawing inferences from the central tendencies of the simulations. Such simulations are becoming increasingly popular with increased computing power, as demonstrated in the social sciences by recent methodological and applied work (Tomz et al., 2003; Breunig and Busemeyer, 2012; Williams and Whitten, 2012; Philips et al., 2015, 2016; Gandrud et al., 2015; Choirat et al., 2018).

Recall that the other challenge we wish to confront is that of identifying cointegration in autoregressive distributed lag models in error-correction form. In instances of small-sample data, especially of time points t of 80 or less, traditional tests like the Engle-Granger "two-step" method (Engle and Granger, 1987) or the Johansen (1991, 1995) approaches too often conclude cointegration when it does not exist (Philips, 2018). An alternative test, proposed by Pesaran et al. (2001), is more conservative, meaning it does not conclude cointegration when it does not exist, especially in small samples. This test, which we call the ARDL-bounds test, is desirable for the social sciences, where shorter time series (smaller samples) are quite common. The difficulty of the ARDL-bounds test is that it requires a specific form of the ARDL model and unique critical values. While these are not insurmountable obstacles, the usefulness of the test would be extended if software existed to get and test these critical values for the user.

In **dynamac**, these challenges are resolved through `pssbounds` (named for Pesaran, Shin and Smith, the authors of the test), which implements the ARDL-bounds test for cointegration, and `dynardl`, which estimates autoregressive distributed lag models and implements the stochastic simulations we describe above. The commands also have the virtue of working together: estimating an ARDL model in error-correction form with `dynardl` by nature stores the values necessary to execute the ARDL-bounds test with `pssbounds`. Below, we use a substantive example to motivate our discussion of the package in greater detail.

Modeling using dynamac

We illustrate the process—autoregressive distributed lag modeling, testing for cointegration with `pssbounds`, and interpretation of X through stochastic simulations—using data originally from Wright (2017) on public concern about inequality in the United States.[7] For our example, assume that public concern about inequality in the US, Concern (`concern`), is a function of the share of income going to the top ten percent, Income Top 10 (`incshare10`). We also hypothesize that the unemployment rate, Unemployment (`urate`), affects concern over the short-run (i.e., is not cointegrating):

$$\text{Concern}_t = f(\text{Income Top } 10_t + \Delta\text{Unemployment}_t) \tag{4}$$

Before estimating any model using **dynamac**, users should first check for stationarity. A variety of unit root tests can be performed using the **urca** package (Pfaff et al., 2016). These suggest that all three series are integrated of order I(1), as they appear integrated in levels but stationary in first-differences (Δ), shown in Table 1.

Table 1: Unit root tests

	Augmented DF	Phillips-Perron	Dickey-Fuller GLS	KPSS
Concern	0.688	-3.437*	-0.893	0.642*
ΔConcern	-3.507*	-7.675*	-3.124*	0.814*
Unemployment	-0.612	-2.762	-2.802*	0.224
ΔUnemployment	-5.362*	-4.879*	-5.308*	0.064
Income Top 10	2.992	0.442	0.994	2.482*
ΔIncome Top 10	-3.170*	-6.244*	-4.032*	0.218

One augmenting lag included for all tests. $*: p < 0.05$. Augmented Dickey-Fuller, PP, and DF-GLS have null hypothesis of a unit-root, while KPSS has a null of stationarity.

Given that all series appear to be I(1), we proceed with estimating a model in `dynardl` in error-correction form, and then testing for cointegration between concern about inequality and the share

[7] Available at: http://dx.doi.org/10.7910/DVN/UYUU9G. For simplicity, we use a smaller version of Wright's dataset with missing values at the beginning of the series removed.

of income of the top 10 percent. In general, we suggest using this strategy—outlined in Philips (2018)—along with alternative tests for cointegration.[8] Our error-correction model appears as:

$$\begin{aligned}\Delta\text{Concern}_t = \alpha_0 + \phi_1\text{Concern}_{t-1} + \theta_1\text{Income Top }10_{t-1} + \\ \beta_1\Delta\text{Income Top }10_t + \beta_2\Delta\text{Unemployment}_t + \epsilon_t\end{aligned} \tag{5}$$

where we assume $\epsilon_t \sim N(0, \sigma^2)$. Now we estimate this model with `dynardl`.

Estimation using `dynardl`

The syntax for `dynardl`, the main function in the package, is as follows:

- `formula`, a formula of the type $y \sim x_1 + x_2 + x_3 + \ldots x_k$. Note the variables do not need to be lagged or differenced in the specification; these transformations will be handled automatically by `dynardl`.
- `data`, an optional argument specifying a particular dataset in which the variables from `formula` can be found.
- `lags`, a *list* of variables to be lagged and their corresponding lagged levels. For instance, if an analyst had two variables, X_1 and X_2, that he or she wished to incorporate with a single lag at $t-1$, he or she would specify `lags = list("X1" = 1,"X2" = 1)`. If he or she wanted to incorporate multiple lags on X_2 from time periods $t-1$ and $t-2$, the code would appear as `lags = list("X1" = 1,"X2" = c(1,2,...)`, expanded for however many lags were desired.
- `diffs`, a *vector* of variables to be differenced, i.e., included as ΔX_t. Only first-differences are supported. For instance, if an analyst wanted to first-difference X_1 and X_3, he or she would include `diffs = c("X1",X3")`.
- `lagdiffs`, a *list* of variables to be included with lagged first-differences. The syntax is identical to `lags`, but instead of incorporating lagged levels of the relevant variable X_k at $t-1$, lagged differences $\Delta X_{k,t-1}$ are included. For instance, if the analyst wanted to include a first lagged difference of $X_{1,t}$ at $t-1$, he or she would include `lagdiffs = list("X1" = 1)`.
- `levels`, a *vector* of variables to be included in levels contemporaneously—i.e., at time t. If the analyst wanted to include X_2 and X_3 in levels at time t, he or she would include `levels = c("X2","X3")`.
- `ec`, a TRUE/FALSE option for whether the model should be estimated in error-correcting form. If `ec = TRUE`, the dependent variable will be run in differences. The default is FALSE.
- `trend`, a TRUE/FALSE option for whether a linear trend should be included in the regression. The default is FALSE.
- `constant`, a TRUE/FALSE option for whether a constant should be included in the regression. The default is TRUE.
- `modelout`, a TRUE/FALSE option for whether model output from the ARDL model should be printed in the console. The default is FALSE.
- `simulate`, a TRUE/FALSE option for whether any shocks should be simulated. Since simulations are potentially computationally intensive, if the analyst is simply using `dynardl` to test for cointegration using the bounds procedure, or wishes to just view the model output, he or she might not wish to estimate simulations in the interim. The default is FALSE.[9]

Reading the above, `dynardl` is simply an engine for regression, but one that allows users to focus on theoretical specification rather than technical coding. All variables in the model are entered into the formula. In this sense, `dynardl` can be used in *any* ARDL context, not just ones in which the user is also expecting cointegration testing or dynamic simulations.We estimate our example model shown in Equation 5 using `dynardl` as follows:

```
data(ineq)

res1 <- dynardl(concern ~ incshare10 + urate, data = ineq,
        lags = list("concern" = 1, "incshare10" = 1),
        diffs = c("incshare10", "urate"),
        ec = TRUE, simulate = FALSE )
```

[8]These include the Johansen (1995) cointegration test (available in **urca**) or the proposed test by Engle and Granger (1987), among others.

[9]For this first example, we will treat this option as FALSE.

```
[1] "Error correction (EC) specified;
dependent variable to be run in differences."
```

We specify the formula by letting `dynardl` know what the dependent variable and the regressors are. Next, in the `lags` option, we let the program know we want a lag at $t-1$ for the dependent variable and for Income Top 10.[10] Since Unemployment did not appear in lag form in Equation 5, it does not appear in `lags`. In `diffs`, we let `dynardl` know to include the first-difference for both Income Top 10 and Unemployment. The option `ec = TRUE` estimates the dependent variable in error-correction form (i.e., takes the first-difference of Concern), and the program will issue a message indicating to the user that such a transformation has taken place.[11] By setting `simulate = FALSE`, we save on computing time by estimating the model without performing stochastic simulations needed to make substantive inferences through stochastic simulations, as shown in the next sections. This is useful if the residuals of our first model indicate residual autocorrelation and require respecification.

Presenting the model summary is the usual `summary(foo)`:

```
summary(res1)

Call:
lm(formula = as.formula(paste(paste(dvnamelist), "~", paste(colnames(IVs),
    collapse = "+"), collapse = " ")))

Residuals:
      Min        1Q    Median        3Q       Max
-0.025848 -0.005293  0.000692  0.006589  0.031563

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     0.122043   0.027967   4.364 7.87e-05 ***
l.1.concern    -0.167655   0.048701  -3.443   0.0013 **
d.1.incshare10  0.800585   0.296620   2.699   0.0099 **
d.1.urate       0.001118   0.001699   0.658   0.5138
l.1.incshare10 -0.068028   0.031834  -2.137   0.0383 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01169 on 43 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.3671,       Adjusted R-squared:  0.3083
F-statistic: 6.236 on 4 and 43 DF,  p-value: 0.0004755
```

As shown from the regression results, `dynardl` has included a constant, the lagged dependent variable, `l.1.concern`, the first difference of the two regressors (Income Top 10 and Unemployment), as well as the lag of Income Top 10.[12] While changes in Income Top 10 affect changes in Concern in the short-run, changes in Unemployment do not have a statistically significant effect in the short-run. The lag of Income Top 10 is negative and statistically significant. Moreover, the parameter on the lagged dependent variable is negative, between 0 and -1, and statistically significant, giving us cursory evidence of a cointegrating process taking place; we use a statistical test for this below.

Post-estimation diagnostics

An essential component of ARDL modeling is ensuring that the residuals from any ARDL estimation are white noise. One symptom of residual autocorrelation in the presence of a lagged dependent variable (where $\phi_1 \neq 0$) is that OLS will result in biased and inconsistent estimates (c.f., Keele and Kelly, 2006, pp. 189-191). Autocorrelation is especially pernicious when using the ARDL-bounds cointegration test, since the test relies on the assumption of, "serially uncorrelated errors for the validity of the bounds tests" (Pesaran et al., 2001, p. 311). Moreover, a number of scholars stress that

[10]If we wanted further lags, we could specify it here, for instance `lags = list("concern" = c(1,2), "incshare10" = 2, "urate" = 2)` would add lags at $t-1$ and $t-2$ for concern, and at $t-2$ for `incshare10` and `urate`.

[11]Setting this option to '`FALSE`' would estimate the dependent variable in levels. If `ec = TRUE`, the saved object will also contain `$pssbounds`, described below.

[12]Lagged variables are denoted by `l.X.variable`, where `X` is the lag. First-differences are denoted by `d.variable`. Lagged first-differences are denoted by `l.d.X.variable`, where `X` is the lag.

autocorrelation is likely indicative of model misspecification more broadly—which may cause omitted variable bias in addition to incorrect standard errors—and thus testing for autocorrelation can help inform users as to which covariates should be included or excluded, as well as the correct dynamic specification (Mizon, 1995; Keele and Kelly, 2006).

To assist users in model selection and residual testing, we offer `dynardl.auto.correlated`. This function takes the residuals from an ARDL model estimated by `dynardl` and conducts two tests for autocorrelation—the Shaprio-Wilk test for normality and the Breusch-Godfrey test for higher-order serial correlation—as well as calculates the fit statistics for the Akaike information criterion (AIC), Bayesian information criterion (BIC), and log-likelihood. The arguments for this function are:

- `x`, a `dynardl` model object.
- `bg.type`, a character string, either `"Chisq"` or `"F"` of the type of Breusch-Godfrey test for higher-order serial correlation. This returns either the Chi-squared test statistic or the F statistic. The default is `"Chisq"`.
- `digits`, an integer for the number of digits to round fit statistics to (by default, this is three digits).
- `order`, an integer for the highest possible order of autocorrelation to test in the data. By default, this is computed by the length of the series.
- `object.out`, a TRUE/FALSE option for whether to print this output into an object. This might be useful if the analyst is comparing multiple ARDL objects and testing for white-noise residuals on the basis of fit criteria like AIC. In this case, the analyst could assign the output to some object for later comparison.

To run this post-estimation diagnostics for our model above, we type:

```
dynardl.auto.correlated(res1)

------------------------------------------------------
Breusch-Godfrey LM Test
Test statistic: 3.704
p-value: 0.054
H_0: no autocorrelation up to AR 1

------------------------------------------------------
Shapiro-Wilk Test for Normality
Test statistic: 0.965
p-value: 0.158
H_0: residuals are distributed normal

------------------------------------------------------
Log-likelihood: 148.094
AIC: -284.187
BIC: -272.96
Note: AIC and BIC calculated with k = 5 on T = 48 observations.

------------------------------------------------------
Breusch-Godfrey test indicates we reject the null hypothesis of no
autocorrelation at p < 0.10.
 Add lags to remove autocorrelation before running dynardl simulations.
```

Given the results of the Breusch-Godfrey LM test for autocorrelation, there appears to be some autocorrelation still in the residuals. To mitigate this, we can add lagged first-differences to our model (Philips, 2018). For instance, to add a lagged first-difference of the dependent variable:

```
res2 <- dynardl(concern ~ incshare10 + urate, data = ineq,
        lags = list("concern" = 1, "incshare10" = 1),
        diffs = c("incshare10", "urate"),
        lagdiffs = list("concern" = 1),
        ec = TRUE, simulate = FALSE)
```

For brevity the results are not shown here, but they indicate that AR(1) autocorrelation is no longer present in the residuals after adding a lagged first-difference of the dependent variable.

Cointegration testing using pssbounds

Recall the definition of cointegration: two or more integrated series of order I(1) are cointegrated if some linear combination of them results in a stationary series.[13] This means that they have some long-run relationship; although temporary shocks may move the series apart, there is an attracting force that brings them back into a stable equilibrium relationship over the long-run. These movements are commonly referred to as being "corrected" or "re-equilibrated"; cointegrated series do not simply happen to move together, but are quite literally brought back into long-run relationship by an attracting force. Since not all I(1) series are cointegrating, failing to account for cointegration—in other words, including I(1) series in an error-correction model when they are not cointegrating—can lead to a drastic increase in Type I error (c.f., Grant and Lebo, 2016).

A variety of cointegration tests have been proposed, which we do not discuss here. Instead, we advocate for a test that has been shown to demonstrate strong small sample properties (Philips, 2018); the ARDL-bounds test for cointegration (Pesaran et al., 2001). The ARDL-bounds procedure proceeds as follows. First, analysts must ensure that the regressors are not of order I(2) or more, and that any seasonal component of the series has been removed. Second, analysts must ensure that the dependent variable y is I(1). A wide variety of unit root tests are designed to assist in this, including the Dickey-Fuller, Elliott-Rothenberg-Stock, Phillips-Perron, and Kwiatkowski-Phillips-Schmidt-Shin tests. We reference these tests in Table 1.

Once the analyst ensures that the dependent variable y is I(1), and the independent variables are not of order I(2) and contain no seasonal components, he or she estimates an ARDL model in error-correction form.[14] That form resembles Equation 6, where the dependent variable is run in differences:

$$\begin{aligned}\Delta y_t =&\alpha_0 + \phi_1 y_{t-1} + \theta_{1,1} x_{1,t-1} + \cdots + \theta_{k,1} x_{k,t-1} + \beta_1 \Delta x_{1,t} + \cdots + \beta_k \Delta x_{k,t} + \\ &\sum_{m=1}^{M} \alpha_m \Delta y_{t-m} + \sum_{q_1=1}^{Q_1} \beta_{1q_1} \Delta x_{1,t-q_1} + \cdots + \sum_{q_k=1}^{Q_k} \beta_{kq_k} \Delta x_{k,t-q_k} + \epsilon_t\end{aligned} \tag{6}$$

Two points are key. First, any independent variables that are potentially I(1) must be entered in lagged levels at $t-1$. Second, the resulting residuals of the ARDL model must be white noise in order to perform the cointegration test (approximately normally distributed, with no residual autocorrelation). If they are not, the analyst should incorporate additional lags of the first difference of the variables until they are (Philips, 2018). A variety of tests exist to help determine whether the residuals are white noise, as well as aiding in model specification, including information criteria (like SBIC and AIC), Breusch-Godfrey tests, Durbin's Alternative test, and Cook-Weisberg. Users can find most of these in `dynardl.auto.correlated`, discussed in the previous section.

The special case of the ARDL model in Equation 6 is that the X variables appearing in levels ($\theta_1 \ldots \theta_k$) are the only ones that can possibly be in a cointegrating relationship with the dependent variable y. The ARDL-bounds test for cointegration works by using a Wald or F-test on the following restriction from Equation 6:

$$H_0 : \phi_1 = \theta_{1,1} = \cdots = \theta_{k,1} = 0 \tag{7}$$

In other words, that the coefficients on variables appearing in first lags are jointly equal to zero. The null hypothesis is that of no cointegration. Rejecting the null hypothesis indicates there is a cointegrating relationship between the series. In addition to the F-test, a one-sided t-test may be used to test the null hypothesis that the coefficient on the lagged dependent variable (in levels) is equal to zero, or:

$$H_0 : \phi_1 = 0 \tag{8}$$

The alternative to this test is that $\phi_1 < 0$, or that y is cointegrating with the regressors. This is known as the bounds t-test.

The procedure and tests themselves are relatively straightforward to implement, but are complicated by two factors. The first is that critical values for these tests are non-standard, meaning that they are not readily testable in canned procedures. The correct (asymptotic) critical values are

[13]For instance, given $y_t = \kappa_0 + \kappa_1 x_t + z_t$, cointegration would exist if $z_t \sim$I(0). Higher orders of cointegration may also be cointegrating—something known as multicointegration—but these are not supported by the ARDL-bounds test.

[14]Note this does not require the analyst to make the sharp distinction between an I(0) or an I(1) regressor, a special benefit of the ARDL-bounds procedure. This is especially useful when "near integrated" series can appear to be integrated in small samples even though, if we were to observe more innovations, the series would ultimately return to some mean value (that is, it would be stationary).

provided by Pesaran et al. (2001), and the small sample critical values for the F-statistic are given by Narayan (2005).[15] Moreover, the appropriate critical values depend on the "case" of the regression: a combination of whether the regression includes a (restricted) trend term (or not) and a (restricted) constant (or not). As Philips (2018) describes, whether the resulting F-statistic is higher than the I(1) critical value, below the I(0) critical value, or between the two, gives differential evidence on the integrated nature of the X variables and the potential cointegrating relationship between them.

Just looking at the above discussion: even if the test is powerful, it can be difficult to implement and interpret correctly. Accordingly, we offer two functions to help implement the test and get dynamic inferences. The first function, `dynardl`, estimates the ARDL relationship described above (to obtain the resulting F-statistic and t-statistic). The second function, `pssbounds`, takes the results of the ARDL model and implements the ARDL-bounds test for cointegration, providing the user with the correct critical values and an easy-to-understand description of hypothesis testing.

To summarize, if we find that the dependent variable and regressors are all I(1), they can only appear in lagged levels in the error-correction model if there is evidence of cointegration. **dynamac** contains the Pesaran et al. (2001) ARDL-bounds cointegration test to assist users in testing for this. To implement the bounds testing procedure, we introduce the function `pssbounds`. This function has the following syntax:

- `data`, an optional argument expecting a `dynardl` model, which calculates the following arguments for the user. If none is given, the user must supply each of the following:
- `obs`, an integer for the number of observations in the model time series.
- `fstat`, the F statistic on the restriction that each of the first lags in the model (including the lagged dependent variable) are jointly equal to zero.
- `tstat`, the t statistic on the lagged dependent variable in the error-correction model.
- `case`, a numeric 1-2-3-4-5 or numeral I-II-III-IV-V of the case of the regression.
  - 1: No intercept and no trend
  - 2: Restricted intercept and no trend
  - 3: Unrestricted intercept and no trend (the most common case)
  - 4: Unrestricted intercept and restricted trend
  - 5: Unrestricted intercept and unrestricted trend
- `k`, the number of regressors appearing in first lags.
- `digits`, an integer for the number of digits to round fit statistics to (by default, this is three digits).
- `object.out`, a TRUE/FALSE option for whether to print this output into an object. The default is FALSE.

This command can be implemented in two ways. First, users can run the appropriate model (following the instructions of Philips (2018)) and pass the relevant arguments—`obs`, `fstat`, `tstat`, `case`, and `k`—to `pssbounds`. Alternatively—and as we advocate—users can estimate the model using `foo <-dynardl(...)` and then, post-estimation, run `pssbounds(foo)`.[16]

Moving back to our example, to test for cointegration, we use the `pssbounds` command:[17]

```
res2 <- dynardl(concern ~ incshare10 + urate, data = ineq,
        lags = list("concern" = 1, "incshare10" = 1),
        diffs = c("incshare10", "urate"),
        lagdiffs = list("concern" = 1),
        ec = TRUE, simulate = FALSE)

pssbounds(res2)

PESARAN, SHIN AND SMITH (2001) COINTEGRATION TEST

 Observations: 47
```

[15]No small-sample critical values are given for the t-test, so it should not be used to conclusively decide whether a cointegrating relationship exists; rather, only for confirmatory evidence.

[16]As described earlier, this only applies to models where `ec = TRUE`, as only error-correcting models can potentially be cointegrating.

[17]To re-emphasize, this test is inappropriate if the residuals are not white-noise (Pesaran et al., 2001, p. 311). Therefore, it is crucial to users to first test for stationarity, run `dynardl`, and adjust their model as needed before testing for cointegration.

```
Number of Regressors (k): 1
Case: 3

------------------------------------------------------
-                       F-test                       -
------------------------------------------------------
                <------- I(0) ------------ I(1) ----->
10% critical value       4.19            4.94
5% critical value        5.22            6.07
1% critical value        7.56            8.685


F-statistic = 12.204
------------------------------------------------------
-                       t-test                       -
------------------------------------------------------
                <------- I(0) ------------ I(1) ----->
10% critical value       -2.57            -2.91
5% critical value        -2.86            -3.22
1% critical value        -3.43            -3.82


t statistic = -3.684
------------------------------------------------------

t-statistic note: Small-sample critical values not provided for Case III.
Asymptotic critical values used.
```

The program displays the number of observations from the regression, the number of regressors appearing in lagged levels (i.e., the cointegrating equation), and the case (unrestricted intercept and no trend); all of these are necessary to obtain the special critical values used by Pesaran et al. (2001).[18] This situates the F-statistic and t-statistic with the correct critical values for the appropriate test. Users are then free to compare the test statistics to the critical values and make the appropriate deduction regarding cointegration in the sample model.

In our example, since the value of the F-statistic exceeds the critical value at the upper I(1) bound of the test at the 1% level, we may conclude that Income Top 10 and Concern about inequality are in a cointegrating relationship. As an auxiliary test, `pssbounds` displays a one-sided test on the t-statistic on the lagged dependent variable.[19] Since the t-statistic of -3.684 falls below the 5% critical value I(1) threshold, this lends further support for cointegration. Taken together, these findings indicate that there is a cointegrating relationship between concern about inequality and the income share of the top 10 percent, and that Equation 6 is appropriately specified.[20]

Gaining substantive and statistical significance of results using `dynardl`

Once users have decided on the appropriate theoretical model—accounting for the appropriate lags of the independent variables, first-differenced variables, and so on—and drawn conclusions regarding cointegration using `pssbounds` (if applicable), their resulting models might be somewhat complicated. Our solution to inferences in the face of this complexity is the use of stochastic simulations.[21]

If a user runs `dynardl` with the argument `simulate = TRUE`, then the following options become available:

- `shockvar`, the independent variable to be shocked in the simulation. This is the independent variable for which we want to estimate an effect. Analysis should enter this in quotes, like `shockvar = "X1"`. This is the only option required without a default: we must know which variable we want inferences about.

[18]Pesaran et al. (2001) provide asymptotic critical values. We use small-sample critical values, calculated by Narayan (2005), when available.

[19]We call this an auxiliary test since only asymptotic critical values are available, while small sample critical values are available for most F-tests. When not available, `pssbounds` issues a warning note.

[20]Had we not found evidence of cointegration, (either the F-statistic fell between or below the I(1) and I(0) critical values), we could adjust our model accordingly, as outlined by Philips (2018).

[21]If users are aiming for analysis to be able to be repeated, they should set a seed before engaging in stochastic simulations.

- `shockval`, the amount by which the independent variable should be shocked. The default is a standard deviation of the shock variable.
- `time`, the time period within the `range` of the simulation when to shock the independent variable of interest (the `shockvar`). The default is 10.
- `qoi`, whether we should summarize the response of the dependent variable with the mean or the median. Although the default is `mean`, if there is underlying skew in the distribution, it might be better summarized by `median` (Rainey, 2017).
- `forceset`, a *list* of variables and their values that should be set to particular values during the simulations. By default, variables in levels are held at their means, and variables in differences are held at 0. If `forceset` is not specified, the simulation will be estimated using these default values. However, if the analyst wanted to investigate the change in y when some variables are at different values, he or she should do this with `forceset`. These values can be any user-defined value, including means, medians, percentiles, or other values of interest. For instance, if X_1 was meant to be held at 3 (rather than the mean of X_1), the analyst would specify `forceset = list("X1" = 3)`.
- `range`, an integer for how long the simulation is to run (the default is 20 periods)
- `burnin`, the number of time periods before the range begins. `burnin` allows time for the variables to equilibriate. The default is 20. This means that 20 time periods are simulated, then the range of the simulation begins. Users are not shown the `burnin` time periods.
- `sims`, the number of simulations to run. The quantities of interest are created from these simulations. The default number is 1000.
- `sig`, an integer for any user-desired significance level in the form of $1 - p$. So if the user wants a p value of 0.10, `sig` would be 90. The default level is 95.
- `expectedval`, a TRUE/FALSE option for whether the expected values (which average errors within simulations) or predicted values (which do not) are desired. Unless the analyst has a strong reason to use expected values, we recommend using predicted values by including `expectedval = FALSE`, the default.

`dynardl` takes this set of arguments and generates the appropriate autoregressive distributed lag given the levels, lagged levels, first-differences, and lagged first-differences of the variables provided. Once this model is estimated using OLS, `dynardl` simulates the quantities of interest. Specifically, the coefficients $\hat{\beta}$ are simulated as draws (with number of `sims`) from a multivariate normal distribution with mean of $\hat{\beta}$ and variance from the variance-covariance of the estimated model (using the **MASS** package from Ripley (2018)). We introduce uncertainty into these simulations by simulating $\hat{\sigma}^2$ scaled by random draws from the chi-squared distribution. We then use this estimate of $\hat{\sigma}^2$ to add back in fundamental random error to the predicted value of each simulation. These predictions come from set levels of the independent variables: the X variables in levels are held at their means and other variables in differences are held at 0. Users can override this default by specifying a different value for any variable using `forceset`. Note that the equation is given time to equilibriate (through the use of `burnin` periods) before the independent variable is shocked.

Perhaps the most important argument is `shockvar`. This is the variable for which the user wants some sort of inference regarding the effect of an independent variable on the dependent variable: the response in the dependent variable is created by shocking the `shockvar`. At time `time`, the `shockvar` is "shocked" by the amount of `shockval`. This shock is distributed appropriately, as defined by the model specified. For instance, if the `shockvar` is entered into the model in lagged levels at time $t - 1$, that X variable would be shocked at `time` $+1$. Just like in an interactive model, one of the best ways to gain inferences regarding the effect of X on y is by varying a single X at a single point in time. Accordingly, only a single `shockvar` can be specified.

The only thing left to resolve is what to do with our `sims`. We are not particularly interested in any individual simulation in any individual time period. Rather, we can get a sense of the central effect of the `shockvar` on y by doing some meaningful analysis across the simulations within any time period. Accordingly, in `foo$simulation`, `dynardl` stores the desired summary quantity of interest (either `qoi = "mean"` or `qoi = "median"`) of y in each time period. It also stores the lower and upper percentiles for the 75%, 90% and 95% intervals. Users can add additional intervals with `sig`. Additionally, users can specify whether they want predicted values (the average across the simulations within a time period) or expected values (where each individual simulation essentially averages out fundamental error). This is through the option `expectedval = FALSE` or `TRUE`, respectively; we highly recommend the former.

Moving back to our substantive example, now that we have our model and found the relationship to be cointegrating, we move onto interpreting the results through dynamic simulations. Again, although analytic calculations of short- and long-run effects are possible (c.f., De Boef and Keele, 2008), with a multitude of lags and lagged first differences, the ability to interpret the effect on y, given a change in a regressor, can become even more difficult. One of the easiest ways of getting a sense of the effect of the shockvar on y is through plotting the simulation results (Tomz et al., 2003; Breunig and Busemeyer, 2012; Williams and Whitten, 2012; Philips et al., 2015, 2016; Gandrud et al., 2015; Blackwell and Glynn, 2018; Choirat et al., 2018). To this end, we offer two more functions: `area.simulation.plot` and `spike.simulation.plot`. The only difference is whether an area plot or a spike plot is desired (both are illustrated below). The arguments for both functions are:

- `x`, a dynardl model object with a simulation to plot.
- `response`, whether the the simulated response of the dependent variable should be plotted in absolute `levels` of the response, or in changes from the mean value of the dependent variable (`response = "mean.changes"`). The default is `response = "levels"`.
- `bw`, whether the plot should be produced in black and white (for publications) or color. The default is `bw = FALSE` (a color plot).

To plot these simulations, all the researcher needs to do is imagine the counterfactual he or she wishes to investigate. For instance, what is the effect of a 7-percentage-point increase in Income Top 10 on Concern? To examine statistical and substantive significance of this counterfactual, we estimate the following model below and plot it. Note that users may want to first set a seed in order to ensure that the plots are reproducible, since the simulations are stochastic. The resulting plot is shown in Figure 1.

```
set.seed(2059120)
res3 <- dynardl(concern ~ incshare10 + urate, data = ineq,
        lags = list("concern" = 1, "incshare10" = 1),
        diffs = c("incshare10", "urate"),
        lagdiffs = list("concern" = 1),
        ec = TRUE, simulate = TRUE,
        shockvar = c("incshare10"), shockval = .07)

area.simulation.plot(res3)
```

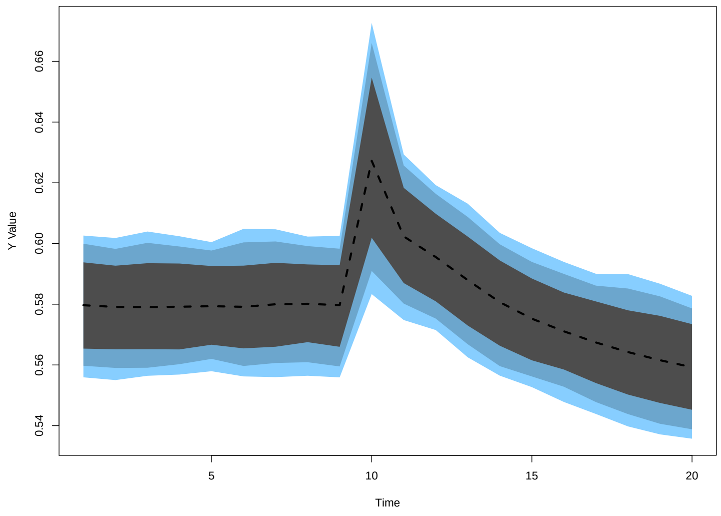


Figure 1: Area plot produced using `area.simulation.plot()`

The black dotted line shows the predicted value, while the colored areas, from darkest to lightest, show the 75, 90, and 95 percentiles of the predictions from the simulations, something akin to a credible interval or confidence interval. The shock to Income Top 10 occurs by default at $t = 10$, though users can change this using the `time` option. If instead we desire a spikeplot, we could type:

```
spike.simulation.plot(res3)
```

The resulting plot is shown in Figure 2.

Figures 1 and 2 both illustrate that the immediate effect of a 7-percentage-point increase in Income Top 10 is to increase Concern by about 4 percentage points. Over time, this effect decays, eventually

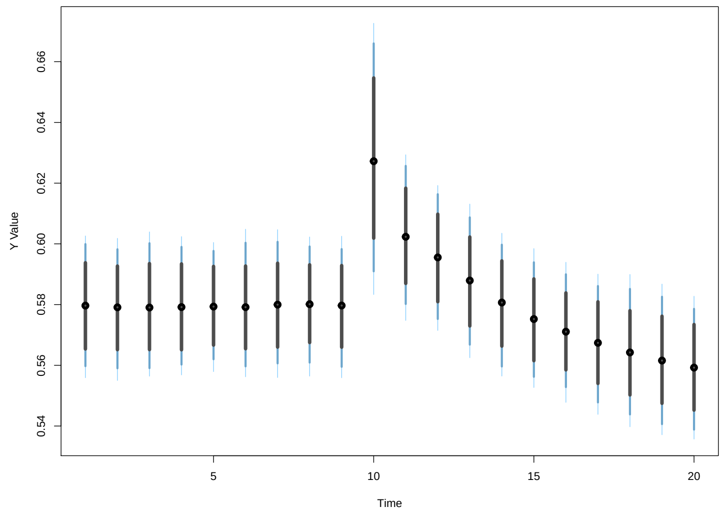


Figure 2: Spike plot produced using `spike.simulation.plot()`

moving Concern to a new value of 0.56 from its pre-shock mean of 0.58. For clarity: these effects are produced by estimating the model formula given by the user, taking the estimated coefficients from the model, drawing a number of simulated coefficients from a multivariate normal distribution with mean and variance given by the estimated coefficients and variance-covariance from the model, and using them to predict values of the dependent variable (and incorporating fundamental error by drawing stochastic terms from $\hat{\sigma}^2$). It then tracks the response in these predicted values to simulated shifts in the independent variables. We might also note that the dependent variable *could* still be responding: users could increase the `range` of the simulation time periods to ensure that the dependent variable has enough time to respond.

Users may have noticed fluctuations in the confidence intervals shown in Figures 1 and 2. This is due to the stochastic sampling technique used to create these simulations. When using `dynardl`, users have the option of increasing the number of simulations performed using the `sims` option. This should have the effect of smoothing out period-to-period fluctuations in the confidence intervals, at the cost of increased computing time.

Additional tools for dynamic analysis with dynamac

`dynardl` has a number of additional features users may find useful. First, not only is it suitable for error-correction models; `dynardl` can model a variety of stationary ARDL processes. To see this, we show a simple example using data from Durr et al. (2000), who examine how ideological movements in the United States affect aggregate public support for the Supreme Court. Our model is:

$$\begin{aligned}\text{Supreme Court Support}_t = &\alpha + \phi_1 \text{Supreme Court Support}_{t-1} + \beta_1 \text{Ideological Divergence}_t + \\ &\beta_2 \text{Congressional Approval}_t + \epsilon_t\end{aligned} \tag{9}$$

where support for the US Supreme Court (1973-1993) (`dcalc`) is a function of a constant, the lag of Supreme Court Support, the ideological divergence between the Supreme Court and the public (Ideological Divergence, `iddiv`), and the level of Congressional Approval (congapp).[22] To estimate this model using `dynardl`, we type:

```
data(supreme.sup)

res4 <- dynardl(dcalc ~ iddiv + congapp, data = supreme.sup,
        lags = list("dcalc" = 1),
        levels = c("iddiv", "congapp"),
        ec = FALSE, simulate = FALSE)
```

[22]As stressed in previous sections, stationarity testing is recommended on all dependent and independent variables before determining whether a stationary or error-correcting ARDL model should be estimated. This is a stylized example only meant to demonstrate additional capabilities of **dynamac**.

```
[1] "Dependent variable to be run in levels."
```

Anytime we specify `levels`, any variables appearing in the list will be included contemporaneously (i.e., appear at time *t* without any lags or differences) in the model. As with the error-correction specification, we can obtain regression results by typing:

```
summary(res4)

Call:
lm(formula = as.formula(paste(paste(dvnamelist), "~", paste(colnames(IVs),
        collapse = "+"), collapse = " ")))

Residuals:
     Min       1Q   Median       3Q      Max
-12.1912  -3.7482  -0.2058   2.6513  11.9549

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  41.6319    11.7367   3.547 0.001078 **
l.1.dcalc     0.2437     0.1352   1.802 0.079758 .
iddiv        -5.4771     2.5803  -2.123 0.040535 *
congapp       0.4732     0.1270   3.725 0.000649 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.626 on 37 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.4693,        Adjusted R-squared:  0.4263
F-statistic: 10.91 on 3 and 37 DF,  p-value: 2.833e-05
```

As the results show, the lag of Supreme Court Support is only weakly related to Supreme Court Support. Moreover, as Ideological Divergence grows, Supreme Court Support decreases. In contrast, as Congressional Approval increases, Supreme Court Support also increases. We can also bring a substantive interpretation to these effects by again using stochastic simulations. We can investigate the counterfactual of a 15-percentage-point decrease in Congressional Approval on Supreme Court Support. Moreover, we demonstrate the difference between the number of simulations used to generate predictions in Figure 3.

```
# 1000 sims
res4 <- dynardl(dcalc ~ iddiv + congapp, data = supreme.sup,
        lags = list("dcalc" = 1),
        levels = c("iddiv", "congapp"),
        ec = FALSE, simulate = TRUE,
        shockval = -15, shockvar = c("congapp"))

# 15000 sims
res5 <- dynardl(dcalc ~ iddiv + congapp, data = supreme.sup,
        lags = list("dcalc" = 1),
        levels = c("iddiv", "congapp"),
        ec = FALSE, simulate = TRUE,
        shockval = -15, shockvar = c("congapp"),
        sims = 15000)

par(mfrow = c(2, 1))
area.simulation.plot(res4)
area.simulation.plot(res5)
```

The effect of the 15-percentage-point decrease in Congressional Approval is about a 10-point decrease in Supreme Court Support. To uncover this effect, the upper plot uses 1000 simulations, while the bottom uses 15000 and results in a much smoother measure of uncertainty in predictions across the simulated timeframe, with only a slight increase in estimation time.[23] Of course, it is important to note that simulations do not *lower* uncertainty (simulating more values does not decrease the underlying variance). More simulations just improve and smooth our estimates within each time period.

[23]For comparison, in this model, 1000 simulations took 3.98 seconds, while 15,000 took 4.12 seconds.

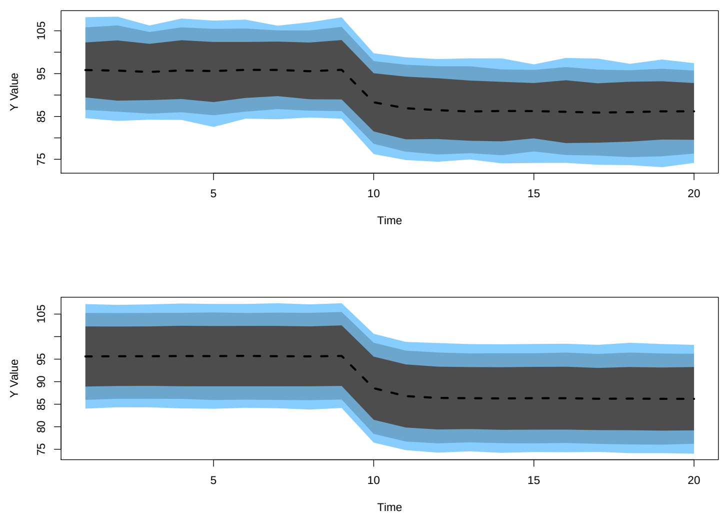


Figure 3: Effect of a 15-percentage-point decrease in Congressional Approval; 1000 simulations (default) (top) and 15000 (bottom)

While the default range for simulations is 20 time periods, with a shock occurring at time $t = 10$—as shown in Figure 3 for instance—users can change these using the `time` and `range` options. In addition, `dynardl` offers the ability to set variables at particular values other than their means throughout the simulation through the option `forceset`.[24] This would be ideal when the mean value is not particularly intuitive or of substantive interest, such as when dealing with a dichotomous variable. Instead, users can specify particular values using the option `forceset`. As an example of this, we re-estimate the model from above, but extend the total simulation range to $t = 50$, the shock time to $t = 20$, and set both Congressional Approval and Ideological Divergence to their 75th percentiles instead of their means:

```
res6 <- dynardl(dcalc ~ iddiv + congapp, data = supreme.sup,
        lags = list("dcalc" = 1),
        levels = c("iddiv", "congapp"),
        ec = FALSE, simulate = TRUE,
        shockval = -15, shockvar = c("congapp"),
        sims = 15000, time = 20, range = 50,
        forceset = list("iddiv" = 0.22, "congapp" = 72))

spike.simulation.plot(res6)
```

The results of a 15-percentage-point drop in Congressional Approval on Supreme Court Support are shown in Figure 4. It is clear that there is a relatively quick negative effect on Supreme Court Support in response to a drop in Congressional Approval, and Supreme Court Support remains at around 88 for the rest of the time periods.

dynamac also offers a few ancillary functions to help assist in dynamic modeling using autoregressive distributed lag models and `dynardl` in particular. These include three basic time-series operators to lag, difference, and lag-difference variables. Users are free to utilize these functions outside of estimating the models.

For lagging variables, we use `lshift(x,l)`, where `x` is the variable to be lagged and `l` is the number of periods to be lagged. If a user wanted a second lag (that is X_{t-2}, he or she would use `lshift(X,2)`:

```
head(ineq$incshare10)
[1] 0.3198 0.3205 0.3198 0.3182 0.3151 0.3175

head(lshift(ineq$incshare10, 2)) # second lag
[1]     NA     NA 0.3198 0.3205 0.3198 0.3182
```

For differencing variables, we use `dshift(x)`, where `x` is the variable to be first-differenced:

[24]Recall that, by default, variables in levels are set to their means throughout the simulation (unless shocked), while variables in differences are set to zero.

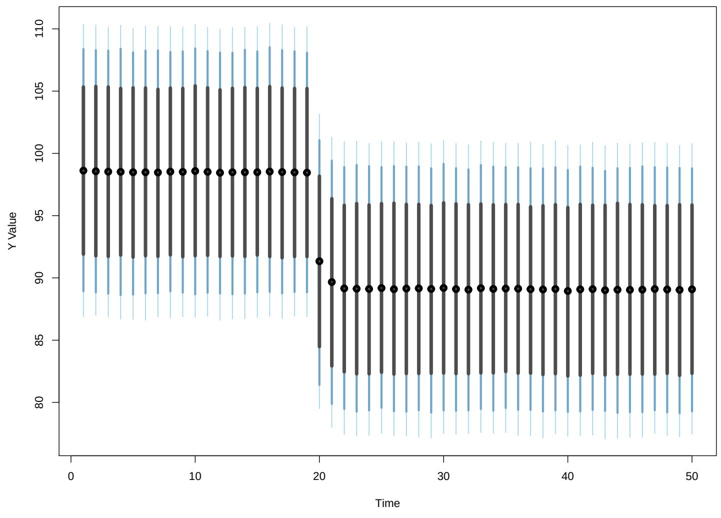


Figure 4: Extending the simulation range and using `forceset`

```
head(dshift(ineq$incshare10))
[1]      NA  0.0007 -0.0007 -0.0016 -0.0031  0.0024
```

For lag-differencing variables, we use `ldshift(x,l)`, where `x` is the variable to be lag-differenced and `l` is the number of periods for the difference to be lagged. If a user wanted for the difference of X from two time periods ago (ΔX_{t-2}) to be calculated, he or she would use `ldshift(X,2)`:

```
head(ldshift(ineq$incshare10, 2))
[1]      NA      NA      NA  0.0007 -0.0007 -0.0016
```

Additional example: the GDP-energy nexus

As a final example, we model the relationship between a country's energy consumption and its economic output. There is a substantial body of literature that focuses on both the short-and long-run effects between energy and GDP—as well as the directionality of the relationship—making it a prime candidate for time series analysis (Ozturk, 2010; Tugcu et al., 2012). In fact, a number of articles utilize the bounds testing approach discussed above (e.g., Odhiambo, 2009; Fuinhas and Marques, 2012). In the analysis below, we focus on two key variables: the natural log of gross domestic product (in constant 2010 US dollars) and the log of energy consumption (in million tons of oil equivalent). These data are for the French economy, measured annually from 1965 to 2017.[25] For this example, we assume that energy consumption, `lnenergy`, drives GDP, `lnGDP_cons2010USD`, not the other way around.

As Figure 5 suggests, both series appear non-stationary, with probably a trend as well. Unit-root tests (not shown) confirm that both series contain a unit root. Therefore, we proceed to estimate an ARDL model in error-correction form with a deterministic trend, test for cointegration, and interpret the results.

```
data(france.data)

res7 <- dynardl(lnGDP_cons2010USD ~ lnenergy, data = france.data,
        lags = list("lnGDP_cons2010USD" = 1, "lnenergy" = 1),
        diffs = c("lnenergy"), trend = TRUE,
        ec = TRUE, simulate = FALSE)

summary(res7)
Call:
lm(formula = as.formula(paste(paste(dvnamelist), "~", paste(colnames(IVs),
    collapse = "+"), collapse = " ")))
```

[25]GDP data are from the World Bank World Development Indicators. Data on energy consumption are from the BP Statistical Review of World Energy (June 2018), available at: https://www.bp.com/en/global/corporate/energy-economics/statistical-review-of-world-energy.html.

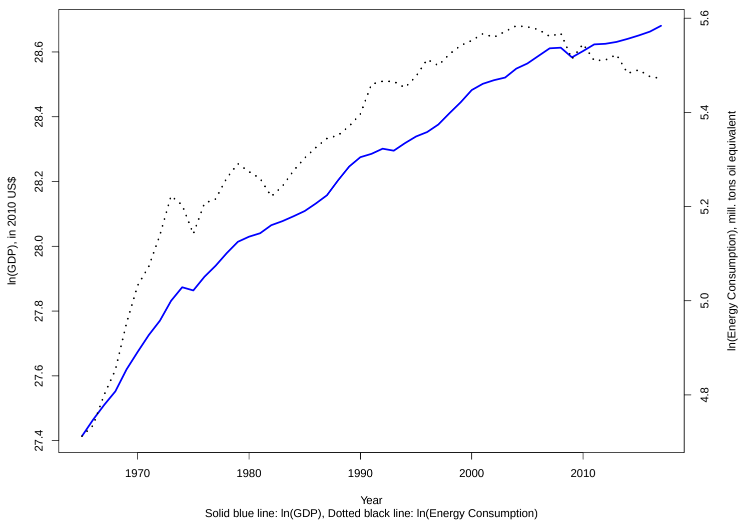


Figure 5: GDP and energy consumption in France, 1965-2017

```
Residuals:
       Min         1Q     Median         3Q        Max 
-0.0262561 -0.0056128  0.0000717  0.0053919  0.0246083 

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)    
(Intercept)            6.995130   1.743518   4.012 0.000214 ***
l.1.lnGDP_cons2010USD -0.283335   0.071552  -3.960 0.000253 ***
d.1.lnenergy           0.257645   0.055622   4.632 2.88e-05 ***
l.1.lnenergy           0.170680   0.048796   3.498 0.001036 ** 
trendvar               0.003824   0.001063   3.598 0.000769 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.01112 on 47 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.6688,	Adjusted R-squared:  0.6406 
F-statistic: 23.72 on 4 and 47 DF,  p-value: 8.84e-11
```

As is clear from the regression results, there appears to be a fairly slow rate of re-equilibration when the two series are out of their long-run equilibrium relationship, as shown by the coefficient on the lagged dependent variable. Moreover, both the first-difference and lag of the log of energy consumption, as well as the trend variable, are statistically significant. Before proceeding to test for cointegration, we ensure the residuals are white-noise using `dynardl.auto.correlated`:

```
dynardl.auto.correlated(res7)

 ------------------------------------------------------
 Breusch-Godfrey LM Test
 Test statistic: 2.563
 p-value: 0.109
 H_0: no autocorrelation up to AR 1

 ------------------------------------------------------
 Shapiro-Wilk Test for Normality
 Test statistic: 0.971
 p-value: 0.241
 H_0: residuals are distributed normal

 ------------------------------------------------------
 Log-likelihood: 162.789
 AIC: -313.578
 BIC: -301.87
 Note: AIC and BIC calculated with k = 5 on T = 52 observations.

 ------------------------------------------------------
```

The Breusch-Godfrey test suggests there is no residual autocorrelation of order AR(1), and the residuals are approximately normally distributed. In addition, using the information criteria reported in dynardl.auto.correlated, we find that the model with a trend term estimated above is more consistent with the data than one without (these results are not reported here). Given that the model appears to be well-specified, we proceed to test for cointegration. Note that the program automatically selects critical values from "Case 5", an unrestricted intercept and trend:

```
pssbounds(res7)

PESARAN, SHIN AND SMITH (2001) COINTEGRATION TEST

Observations: 52
Number of Regressors (k): 1
Case: 5

------------------------------------------------------
-                       F-test                       -
------------------------------------------------------
                <------- I(0) ------------ I(1) ----->
10% critical value       5.8           6.515
5% critical value        6.93           7.785
1% critical value        9.8           10.675


F-statistic = 8.2
------------------------------------------------------
-                       t-test                       -
------------------------------------------------------
                <------- I(0) ------------ I(1) ----->
10% critical value       -3.13            -3.4
5% critical value        -3.41            -3.69
1% critical value        -3.96            -4.26


t statistic = -3.96
------------------------------------------------------

t-statistic note: Small-sample critical values not provided for Case V.
Asymptotic critical values used.
```

Since the F-statistic (i.e., the test that all variables appearing in lagged levels are equal to zero) is larger than the I(1) critical value in the table at the 5% level, we conclude that cointegration between energy consumption and economic output exists. In addition, the one-sided t-test of the coefficient on the lagged dependent variable also supports the conclusion of cointegration. Given our model results, and finding evidence of cointegration, we turn to a substantive interpretation of our findings: what would be the effect of a one standard deviation decrease in energy consumption on GDP?

```
res7 <- dynardl(lnGDP_cons2010USD ~ lnenergy, data = france.data,
        lags = list("lnGDP_cons2010USD" = 1, "lnenergy" = 1),
        diffs = c("lnenergy"), trend = TRUE,
        ec = TRUE, simulate = TRUE, shockvar = "lnenergy", shock = -0.2289)

[1] "Error correction (EC) specified;
dependent variable to be run in differences."
[1] "Deterministic linear trend added to model formula."
[1] "dynardl estimating ..."
  |============================================================| 100%
```

```
spike.simulation.plot(res7)
```

As shown by the spikeplot in Figure 6, the pre-shock (i.e., any prediction before $t = 10$) values are consistently increasing over time; this is due to the trend term. At $t = 10$, the one standard deviation decrease in energy consumption occurs. This effect does not result in a statistically significant decrease in the contemporaneous period, but continues to lower GDP until two periods after the shock occurs. Moreover, it is not until 10 years after this hypothetical shock that GDP returns to its pre-shock level.

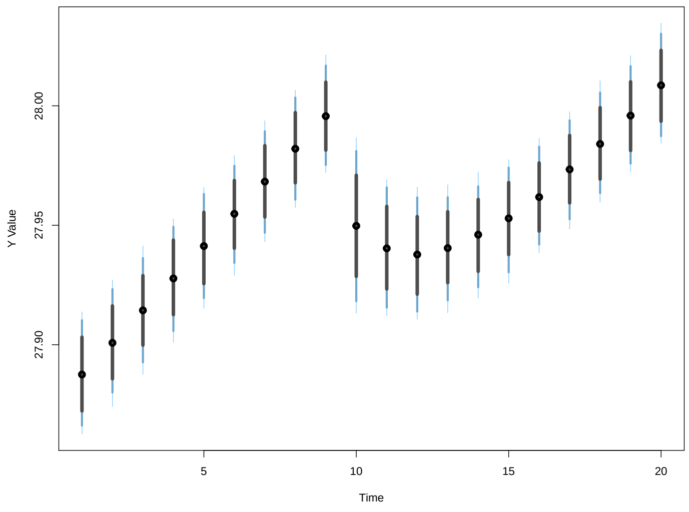


Figure 6: The effect of a one standard deviation decrease in energy consumption on economic growth

Conclusion

In this paper we have introduced **dynamac**, a suite of functions designed to assist applied time series analysts. This centers around `dynardl`, a multipurpose function to estimate autoregressive distributed lag models. This flexible program models both stationary ARDL relationships as well as cointegrating ones, and it allows users to easily change model specifications (e.g., lags, differences, and lag-differences). We also include two post-estimation diagnostic commands to check for white-noise residuals and cointegration. Last, through several applied examples we have shown the advantages of dynamic simulation for interpreting the substantive significance of the results of single-equation time series models, something users can easily do using **dynamac**.

There are a number of interesting potential future research areas. Most useful would be extending ARDL modeling to the panel context. While other packages are designed to handle multiple panels (c.f., Gandrud et al., 2016), the ability to incorporate models that account for unit heterogeneity, such as random or fixed intercepts, would prove useful to users. In addition, adding the ability to estimate alternative models in the dynamic panel data context, such as those that account for causal feedback and time-varying covariates (Blackwell and Glynn, 2018), or generalized method of moments (GMM) estimators, would also assist practitioners.

Bibliography

N. Beck and J. N. Katz. Modeling dynamics in time-series–cross-section political economy data. *Annual Review of Political Science*, 14:331–352, 2011. URL `https://doi.org/10.1146/annurev-polisci-071510-103222`. [p469]

M. Blackwell and A. N. Glynn. How to make causal inferences with time-series cross-sectional data under selection on observables. *American Political Science Review*, 112(4):1067–1082, 2018. URL `https://doi.org/10.1017/S0003055418000357`. [p469, 479, 486]

C. Breunig and M. R. Busemeyer. Fiscal austerity and the trade-off between public investment and social spending. *Journal of European Public Policy*, 19(6):921–938, 2012. URL `https://doi.org/10.1080/13501763.2011.614158`. [p471, 479]

C. Choirat, C. Gandrud, J. Honaker, K. Imai, G. King, and O. Lau. *Zelig: Everyone's Statistical Software*, 2018. URL `https://CRAN.R-project.org/package=Zelig`. R package version 5.1.6. [p470, 471, 479]

H. D. Clarke, K. Ho, and M. C. Stewart. Major's lesser (not minor) effects: Prime ministerial approval and governing party support in Britain since 1979. *Electoral Studies*, 19(2):255–273, 2000. URL https://doi.org/10.1016/S0261-3794(99)00051-7. [p469]

S. De Boef and L. Keele. Taking time seriously. *American Journal of Political Science*, 52(1):184–200, 2008. URL https://doi.org/10.1111/j.1540-5907.2007.00307.x. [p469, 470, 478]

R. H. Durr, A. D. Martin, and C. Wolbrecht. Ideological divergence and public support for the Supreme Court. *American Journal of Political Science*, pages 768–776, 2000. URL https://doi.org/10.2307/2669280. [p480]

R. F. Engle and C. W. Granger. Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, 55(2):251–276, 1987. URL https://doi.org/10.2307/1913236. [p471, 472]

J. A. Fuinhas and A. C. Marques. Energy consumption and economic growth nexus in Portugal, Italy, Greece, Spain and Turkey: An ARDL bounds test approach (1965–2009). *Energy Economics*, 34(2): 511–517, 2012. URL https://doi.org/10.1016/j.eneco.2011.10.003. [p483]

C. Gandrud, L. K. Williams, and G. D. Whitten. *Dynsim: Dynamic Simulations of Autoregressive Relationships*, 2015. URL https://CRAN.R-project.org/package=dynsim. R package version 1.2.1. [p470, 471, 479]

C. Gandrud, L. K. Williams, and G. D. Whitten. Visualize dynamic simulations of autoregressive relationships in R. *The Political Methodologist*, 23(2):6–10, 2016. [p470, 486]

T. Grant and M. J. Lebo. Error correction methods with political time series. *Political Analysis*, 24(1): 3–30, 2016. URL https://doi.org/10.1093/pan/mpv027. [p475]

D. Jacobs and R. E. Helms. Toward a political model of incarceration: A time-series examination of multiple explanations for prison admission rates. *American Journal of Sociology*, 102(2):323–357, 1996. URL https://doi.org/10.1086/230949. [p469]

S. Johansen. Estimation and hypothesis testing of cointegration vectors in Gaussian vector autoregressive models. *Econometrica: Journal of the Econometric Society*, 59(6):1551–1580, 1991. URL https://doi.org/10.2307/2938278. [p471]

S. Johansen. *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*. Oxford University Press, 1995. URL https://doi.org/10.1093/0198774508.001.0001. [p471, 472]

S. Jordan and A. Q. Philips. *Dynamac: Dynamic Simulation and Testing for Single-Equation ARDL Models*, 2018. URL https://CRAN.R-project.org/package=dynamac. R package version 0.1.6. [p469]

L. Keele and N. J. Kelly. Dynamic models for dynamic theories: The ins and outs of lagged dependent variables. *Political Analysis*, 14(2):186–205, 2006. URL https://doi.org/10.1093/pan/mpj006. [p473, 474]

K. Mertens and J. L. Montiel Olea. Marginal tax rates and income: New time series evidence. *The Quarterly Journal of Economics*, 133(4):1803–1884, 2018. URL https://doi.org/10.1093/qje/qjy008. [p469]

G. E. Mizon. A simple message for autocorrelation correctors: Don't. *Journal of Econometrics*, 69(1): 267–288, 1995. URL https://doi.org/10.1016/0304-4076(94)01671-L. [p474]

P. K. Narayan. The saving and investment nexus for China: Evidence from cointegration tests. *Applied Economics*, 37(17):1979–1990, 2005. URL https://doi.org/10.1080/00036840500278103. [p476, 477]

N. M. Odhiambo. Energy consumption and economic growth nexus in Tanzania: An ARDL bounds testing approach. *Energy Policy*, 37(2):617–622, 2009. URL https://doi.org/10.1016/j.enpol.2008.09.077. [p483]

I. Ozturk. A literature survey on energy–growth nexus. *Energy Policy*, 38(1):340–349, 2010. URL https://doi.org/10.1016/j.enpol.2009.09.024. [p483]

M. H. Pesaran, Y. Shin, and R. J. Smith. Bounds testing approaches to the analysis of level relationships. *Journal of Applied Econometrics*, 16(3):289–326, 2001. URL https://doi.org/10.1002/jae.616. [p469, 470, 471, 473, 475, 476, 477]

B. Pfaff, E. Zivot, and M. Stigler. *Urca: Unit Root and Cointegration Tests for Time Series Data*, 2016. URL https://CRAN.R-project.org/package=urca. R package version 1.3-0. [p471]

A. Q. Philips. Have your cake and eat it too? Cointegration and dynamic inference from autoregressive distributed lag models. *American Journal of Political Science*, 62(1):230–244, 2018. URL https://doi.org/10.1111/ajps.12318. [p471, 472, 474, 475, 476, 477]

A. Q. Philips, A. Rutherford, and G. D. Whitten. The dynamic battle for pieces of pie—modeling party support in multi-party nations. *Electoral Studies*, 39:264–274, 2015. URL https://doi.org/10.1016/j.electstud.2015.03.019. [p471, 479]

A. Q. Philips, A. Rutherford, and G. D. Whitten. Dynamic pie: A strategy for modeling trade-offs in compositional variables over time. *American Journal of Political Science*, 60(1):268–283, 2016. URL https://doi.org/10.1111/ajps.12204. [p471, 479]

C. Rainey. Transformation-induced bias: Unbiased coefficients do not imply unbiased quantities of interest. *Political Analysis*, 25(3):402–409, 2017. URL https://doi.org/10.1017/pan.2017.11. [p478]

B. Ripley. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2018. URL https://CRAN.R-project.org/package=MASS. R package version 7.3-49. [p478]

M. Tomz, J. Wittenberg, and G. King. Clarify: Software for interpreting and presenting statistical results. *Journal of Statistical Software*, 8(1):1–30, 2003. URL https://doi.org/10.18637/jss.v008.i01. [p471, 479]

C. T. Tugcu, I. Ozturk, and A. Aslan. Renewable and non-renewable energy consumption and economic growth relationship revisited: Evidence from G7 countries. *Energy Economics*, 34(6):1942–1950, 2012. URL https://doi.org/10.1016/j.eneco.2012.08.021. [p483]

L. K. Williams and G. D. Whitten. But wait, there's more! Maximizing substantive inferences from TSCS models. *The Journal of Politics*, 74(03):685–693, 2012. URL https://doi.org/10.1017/s0022381612000473. [p471, 479]

C. Wlezien. The public as thermostat: Dynamics of preferences for spending. *American Journal of Political Science*, pages 981–1000, 1995. URL https://doi.org/10.2307/2111666. [p469]

G. Wright. The political implications of American concerns about economic inequality. *Political Behavior*, pages 1–23, 2017. URL https://doi.org/10.1007/s11109-017-9399-3. [p471]

Soren Jordan
Department of Political Science
Auburn University
United States
sorenjordanpols@gmail.com

Andrew Q. Philips
Department of Political Science
University of Colorado Boulder
United States
andrew.philips@colorado.edu

The politeness Package: Detecting Politeness in Natural Language

by Michael Yeomans, Alejandro Kantor, Dustin Tingley

Abstract This package provides tools to extract politeness markers in English natural language. It also allows researchers to easily visualize and quantify politeness between groups of documents. This package combines and extends prior research on the linguistic markers of politeness (Brown and Levinson, 1987; Danescu-Niculescu-Mizil et al., 2013; Voigt et al., 2017). We demonstrate two applications for detecting politeness in natural language during consequential social interactions—distributive negotiations, and speed dating.

Introduction

Politeness is a universal dimension of human communication (Goffman, 1967; Lakoff, 1973; Brown and Levinson, 1987). In practically all settings, a speaker can choose to be more or less polite to their audience, and this can have consequences for the speakers' social goals. Politeness is encoded in a discrete and rich set of linguistic markers that modify the information content of an utterance. Sometimes politeness is an act of commission (for example, saying "please" and "thank you") <and sometimes it is an act of omission (for example, declining to be contradictory). Furthermore, the mapping of politeness markers often varies by culture, by context (work vs. family vs. friends), by a speaker's characteristics (male vs. female), or the goals (buyer vs. seller). Many branches of social science might be interested in how (and when) people express politeness to one another, as one mechanism by which social co-ordination is achieved.

The `politeness` package is designed to make it easier to detect politeness in English natural language, by quantifying relevant characteristics of polite speech, and comparing them to other data about the speaker. For example, researchers may want to know whether politeness is associated with some situational or trait-level covariate (text as description). Or researchers might want to know whether people respond differently to polite rather than impolite language (text as treatment). Or they might want to know how an intervention affects the production of politeness (text as outcome). Finally, all of these analytical approaches can coalesce to support a theoretical model in which speakers strategically choose their politeness as a way to affect their audience's behavior (text as mediator).

Politeness typically draws from a common pool of linguistic markers for social co-ordination between speaker and listener. But the weight and valence of each marker depends on the context. Thus, we do not try to provide a single "politeness dictionary" for all contexts. Instead our approach draws from common methods in computational linguistics that use algorithms to select from a curated set of features. (Manning and Schütze, 1999; Grimmer and Stewart, 2013; Jurafsky and Martin, 2014). That is, we draw on existing linguistic theory to calculate a wide set of potentially relevant features from the text. But we then estimate the weights on those features empirically, defining politeness using some ground truth label—from a randomized treatment, or the listener, or a third party, or the speaker herself. We then use a supervised machine learning algorithm as a context-specific model of politeness: to classify unlabeled documents, and to characterize the nature of how politeness is expressed in the domain of interest.

Our software contributes to a rich ecosystem of general text analysis packages in R. This includes structuring raw text (e.g. **tidytext**, **tm**, **quanteda**, **coreNLP**, **spacyR**), sentiment analysis (e.g., **tidytext**, **SentimentAnalysis**, **syuzhet**), and topic modeling (**topicmodeling**, **stm**). We incorporate some of these packages in our own work. Our package is the first in R to study politeness specifically, and also one of the first to focus on linguistic pragmatics more broadly. By focusing on the turn-level syntactic structure in natural language, our work is complementary to (and distinct from) existing work that primarily focuses on semantic content, such as identifying trends in topics or emotions over large corpora.

In this paper we also work through two important applications of politeness detection in social science. First, we measure manipulated politeness as a treatment effect in an experiment in which writers were instructed to write offers for a phone on `craigslist.com`, in a style that was high (or low) in politeness. This was used to validate a psychometric construct across several studies. Second, we measure observed politeness as a context-specific and naturally-occurring construct in a dataset on speed-dating. This was used to understand the meaning of politeness from different perspectives.

Politeness workflow

The politeness package provides functions to identify politeness markers in natural language, graphically compare these to covariates of interest, develop a supervised model to detect politeness in new documents, and inspect high- and low-politeness documents. Table 1 summarizes the main functions of the politeness package. A full description of the functions is available in the package documentation.

Function	Description
`politeness()`	Detects linguistic markers of politeness in natural language. Takes an N-length vector of text documents and returning an N-row data.frame of feature counts. Some politeness features depend on grammatical parsing.
`politenessPlot()`	Plots the prevalence of politeness features over a set of documents. Highlights differences in politeness across covariate.
`politenessProjection()`	Training and projecting a regression model of politeness based on a binary or continuous variable. Supports both **glmnet** and **textir**, with LASSO as the default.
`findPoliteTexts()`	Finds examples of most or least polite texts in a corpus from a covariate identifying politeness scores of texts.

Table 1: Politeness functions.

These tools can be combined in a workflow that we believe will be useful to most researchers interested in linguistic politeness. First, we offer a function, `politeness()` that will calculate a set of linguistic features that have been identified in the past as relating to politeness. Second, we offer a function, `politenessPlot()` to visualize these counts, in comparison to a covariate related to politeness (e.g., treatment/control). If the researcher wants to generate a politeness classifier, they can do so using the `politenessProjection()` function, which creates a single mapping from the politeness features in the supplied text to some other measure of interest. In particular, if the researcher has some "ground truth" labels of politeness over a set of texts, they can use this function as a politeness classifier, and automatically assign politeness scores to many more new texts.

Politeness features

Broadly speaking, the space of politeness features is guided by a rich literature on the linguistics of politeness (Goffman, 1967; Lakoff, 1973; Brown and Levinson, 1987). In general, politeness in language is designed to pay face to the listener, so that they feel respected. And while the linguistic markers of politeness vary from situation to situation, there are two common themes in most polite speech: Positive Politeness, and Negative Politeness.

Positive politeness involves actively bolstering the listener's self-image (showing gratitude, identifying as an in-group member, paying complements) as well as not derogating that image (complaints, cursing, informal titles, and so on). Negative Politeness involves respecting the listener's autonomy. This involves a general softening of statements, using hedges and adverbs. Requests may also be tempered, using indirect subjunctive language, and apologizing. Alternatively, speakers may express low negative politeness by making bare commands and being contradictory.

These elements are all included in this package as part of the `politeness()` function, which tallies 36 separate politeness markers (summarized in Table 2, along with examples of each.). Many are translated directly from recent research on the computational linguistics of politeness (Danescu-Niculescu-Mizil et al., 2013; Voigt et al., 2017). We collected all of the features from these two papers, and removed a few that were very contextually specific (e.g., "keep your hands on the wheel" for drivers). However as we demonstrate below, many kinds of context-specific features can be helpful, and we show how to add them to the feature set manually.

The features in the politeness detector are summarized in in Table 2. We refer interested users to the original papers for details on the design of each feature. All features involve counting matches to a pre-defined list, which includes some combination of individual words, word stems, adjacency pairs, dependency pairs, part-of-speech-tags. Additionally, some features distinguish whether a match is found at the beginning of a sentence or not. Some (e.g., positive or negative emotion) include hundreds of possible matches; while others (e.g., "for you") are defined by a single phrase.

```
library(politeness)

data("feature_table")
```

```
feature_table
```

Feature Name	POS Tags	Description	Example
Hello	No	"hi", "hello", "hey"	"Hi, how are you today?"
Goodbye	No	"goodbye", "bye", "see you later"	"That's my best offer. Bye!"
Please Start	Yes	Please to start sentence	"Please let me know if that works"
Please	Both	Please mid-sentence	"Let me know if that works, please"
Gratitude	Both	"thank you", "i appreciate", etc.	"Thanks for your interest"
Apologies	Both	"sorry", "oops", "excuse me", etc.	"I'm sorry for being so blunt"
Formal Title	No	"sir", "madam", "mister", etc.	"Sir, that is quite an offer."
Informal Title	No	"buddy", "chief", "boss", etc.	"Dude, that is quite an offer."
Swearing	No	Vulgarity of all sorts	"The dang price is too high"
Subjunctive	No	Indirect request	"Could you lower the price?"
Indicative	No	Direct request	"Can you lower the price?"
Bare Command	Yes	Unconjugated verb to start sentence	"Lower the price for me"
Let Me Know	No	"let me know"	"Let me know if that works"
Affirmation	Yes	Direct agreement at start of sentence	"Cool, that works for me"
Conjunction Start	Yes	Begin sentence with conjunction	"And if that works for you"
Reasoning	No	Explicit reference to reasons	"I want to explain my offer price"
Reassurance	No	Minimizing other's problems	"Don't worry, we're still on track"
Ask Agency	No	Request an action for self	"Let me step back for a minute"
Give Agency	No	Suggest an action for other	"I want to let you come out ahead"
Hedges	No	Indicators of uncertainty	"I might take the deal"
Actually	Both	Indicators of certainty	"This is definitely a good idea."
Positive	No	Positive emotion words	"that is a great deal"
Negative	No	Negative emotion words	"that is a bad deal"
Negation	No	Contradiction words	"This cannot be your best offer"
Questions	No	Question words to start sentence	"Why did you settle on that value?"
By The Way	No	"by the way"	"By the way, my old offer stands"
Adverbial Just	Yes	modifying a quantity with "just"	"It is just enough to be worth it"
Filler Pause	No	Filler words and verbal pauses	"That would be, um, fine"
For Me	No	"for me"	"It would be great for me"
For You	No	"for you"	"It would be great for you"
First Person Plural	No	First-person plural pronouns	"it's a good deal for both of us"
First Person Single	Both	First-person singular mid-sentence	"It would benefit me, as well"
Second Person	Both	Second person mid-sentence	"It would benefit you, as well"
First Person Start	Yes	First-person singular to start sentence	"I would take that deal"
Second Person Start	Yes	Second-person to start sentence	"You should take that deal"
Impersonal Pronoun	No	Non-person referents	"That is a deal"

Table 2: Politeness features detected by `politeness()`. Features that have “No” in the POS Tags column require part-of-speech (POS) tagging; where as those with “Both” can be approximated with out POS tagging, but POS tagging is recommended.

Parsing grammar

The meaning of a sentence often depends not just on its constituent words, but also on its grammatical structure. This is useful for words that can have different meanings, such as the adverbial "just" in "it is just enough" compared to the adjectival use in "the decision was just". Politeness can also be expressed in the grammatical structure itself (e.g., the unconjugated verbs in bare commands like "give me that!"). This information is lost in in bag-of-words analyses that do not label the sentence structure, or ignore word order.

As of this writing, there was no available grammar parsing software available wholly within the R language. Instead, we build off one of the most powerful natural language processing tools available currently—the Python module SpaCy (Honnibal and Johnson, 2015). It is open-source, fast, accurate, and well-benchmarked. We use **spacyr** (Benoit and Matsuo, 2017) to connect to SpaCy and take advantage of their pretrained models to identify the grammatical structure of each document. The current SpaCy model en_core_web_sm is a Convolutional Neural Network trained on OntoNotes, a large corpus comprising of news, conversational telephone speech, weblogs, usenet newsgroups, broadcast, and talk shows texts. Prior to using **spacyr**, users must install SpaCy in Python. As of **spacyr** version 0.9.6, this spacy installation can be detected automatically on the user's computer by **spacyr** when it is first called.[1]

[1]For advanced users, including those who may have multiple Python installations, you may have to *initialize* the SpaCy engine first, so that it is ready for use during the session. That is done using a separate function

Many of the politeness features can be calculated without grammar parsing by setting parser="none". We recommend this as an initial first step for researchers, without having to install SpaCy. At this reduced setting, some features are dropped entirely (e.g., Bare Commands are a specific verb class). However, some features are approximated. For example, tags allow users to differentiate between a mid-sentence "please" and a "please" to start a sentence, while the tagless version of the function will collapse them into a single feature.

Detecting politeness features

The function politeness() takes in an *n*-length vector of texts, and returns an *n*-by-*f* data.frame, with *f* columns corresponding to the number of calculated features. We note that our package does not perform any spell- or grammar-checking on the text—though these kinds of errors can degrade the fidelity of the information in the text. Instead, we recommend that users do this on their own—either in R, using another package like **hunspell**, or else using other software or by hand.

There are 36 features in total (see Table 2), but some user options affect the number of features that are returned. For one, if a part-of-speech tagger is not used (by setting parser = "none") then some features that depend on these tags will not be calculated, as detailed in Table 2. Additionally, you may use the default setting drop_blank = TRUE which will remove features that were not detected in any of the texts.

For example, consider the following texts.

```
library(politeness)

texts <- c("Hello, you", "You")
```

politeness() with parser="none" (default) will identify that both texts contain the feature **Second.Person** and that only the first contains **Hello**.

```
df_politeness <- politeness(texts, parser="none", drop_blank = TRUE)
df_politeness

  Hello Second.Person
1     1             1
2     0             1
```

In order to get the full feature space we use parser="spacy". Now politeness() differentiates between the features **Second.Person.Start** (which captures the pronoun as a sentence subject) and **Second.Person** (which captures the pronoun as a sentence object) as shown below.

```
df_politeness <- politeness(texts, parser="spacy", drop_blank = TRUE)
df_politeness

  Hello Second.Person.Start Second.Person
1     1                   0             1
2     0                   1             0
```

The examples above are very short, in practice longer sentences will use features multiple times (e.g., positive words). Setting metric="count" will populate each cell with the raw count of each feature in the text. Alternatively, metric="binary" will return a binary indicator of the presence or absence of each feature. Finally, metric="average" will count the prevalence of features as a percentage of the word count of each document. This is useful as a robustness check when there is wide variance in document length. For example take the following two texts (this data, borrowed from (Jeong et al., 2018) is included in the package):

```
data("phone_offers")
texts <- phone_offers$message[c(21,25)]
texts
```

('spacyr::spacy_initialize(python_executable = PYTHON_PATH)'—make sure to substitute in your preferred Python path name) and is explained well in the **spacyr** documentation.

```
[1] "Hi I am very interested in your phone.  It is exactly what I have been looking for.
I would like to offer you 115 for it.  It would make me very happy to buy the phone today."

[2] "Hi, I hope your day is going well. I am very pleased to see the phone you are offering
for sale, as it is exactly what I need! I am on a very tight budget so I hope that you will
be willing for accept $115 for the phone. It is the most I can pay. Please know that I
would be so happy if I am able to buy this phone. I'm sorry that I can't offer more.
If you are willing to accept my offer, perhaps I can do you a small favor as well, like
mow your lawn or something. In any case if you accept my offer you would have my sincere
and heartfelt gratitude.   Whether you accept my offer or not, I hope that this
message finds you and yours well and happy. I hope you have a great day. :)"
```

Given a subset of features we observe the following counts.

```
features <- c("Positive.Emotion", "Impersonal.Pronoun","First.Person","Second.Person",
              "Negative.Emotion")

df_politeness <- politeness(texts,drop_blank = TRUE)
df_politeness[ , features]

  Positive.Emotion Impersonal.Pronoun First.Person Second.Person Negative.Emotion
1                2                  4            3             2                0
2               14                 10           17            12                1
```

Although both texts have similar politeness features, the longer text contains larger count values. The option `metric="binary"`, on the other hand, will return a simplified result with a 1 if the feature is present in the text as shown below.

```
df_politeness <- politeness(texts, metric="binary",drop_blank = TRUE)
df_politeness[ , features]

  Positive.Emotion Impersonal.Pronoun First.Person Second.Person Negative.Emotion
1                1                  1            1             1                0
2                1                  1            1             1                1
```

Plotting politeness features

The `politenessPlot()` function combines the politeness feature matrix with another measure of substantive interest that might covary with the prevalence of some of the politeness features—in particular, a ground-truth measure of politeness from annotation or assignment to treatment. This function generates a horizontal bar plot (with 95% confidence intervals) of feature prevalence among two groups of documents.

The function can handle any kind of politeness metric (counts, binary, or averaged), but that distinction must be made in the initial call to `politeness()`. Below, we plot the politeness features from a feature count matrix in Figure 1, and a binary feature matrix in Figure 2.

The order of the bars themselves are sorted automatically, and determined by calculating the variance-weighted log odds of each feature with respect to the binary covariate. Many covariates are continuous. By default, this package treats the top and bottom terciles of that distribution as binary categories, with the middle tercile dropped. Users can also create their own categories beforehand, and enter those labels in place of the covariate.

Often some features are not meaningful for further analysis – either because they are too rare in the data, or else because they do not meaningfully covary with the covariate of interest. Users have two options to exclude these from the plot. First, the `drop_blank` parameter can remove rare features – it takes a number between 0 and 1, which determines a cut-off based on prevalence. Specifically, all features which appear in less than this proportion of texts are excluded from the plot. To include all features, leave this value at 0.

Second, the `middle_out` parameter can remove features which do not vary meaningfully across the covariate. Each feature is evaluated using a two-sample t.test, and features are removed when the p-value of this test lies above the user's provided cut-off, a number between 0 and 1 (the default is 0.1). To include all features, simply set this value at 1.

```
df_politeness_count <- politeness(phone_offers$message,
                                  parser="none",
                                  drop_blank=FALSE)

politenessPlot(df_politeness_count,
               split=phone_offers$condition,
               split_levels = c("Tough","Warm"),
               split_name = "Condition")
```

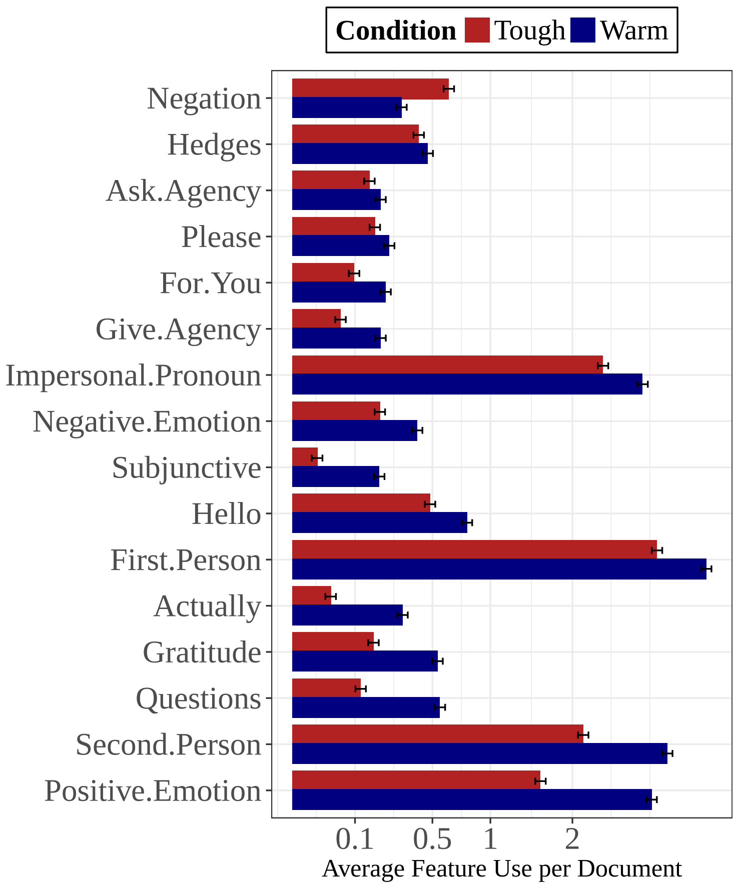


Figure 1: Politeness plot showing for each feature the average use per document in square root scale. Features are ordered by their variance-weighted log odds. This can accommodate any metric for the politeness features (count, binary, average) but that must be set in the call of `politeness()` before the data.frame is passed the plot function.

```
df_politeness_binary <- politeness(phone_offers$message,
                                   parser="none",
                                   metric="binary",
                                   drop_blank=FALSE)

politenessPlot(df_politeness_binary,
               split=phone_offers$condition,
               split_levels = c("Tough","Warm"),
               split_name = "Condition")
```

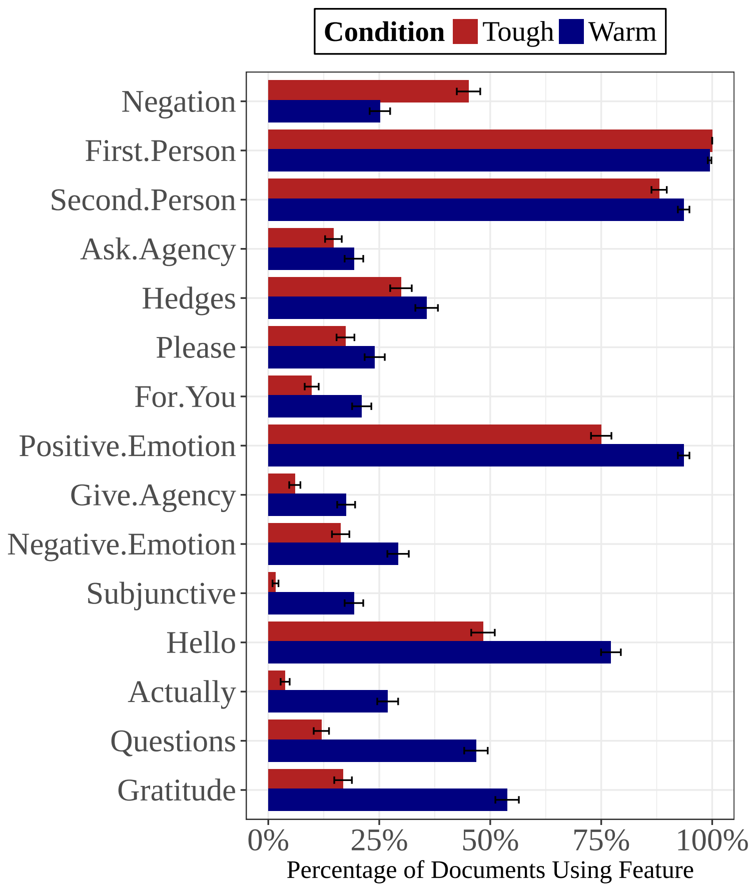


Figure 2: Politeness plot showing percentage of documents showing each feature. Features are ordered by their variance-weighted log odds. In order to plot the percentage of documents showing each feature `metric="binary"` must be set in the call of `politeness()`.

Projecting politeness features

Users can generate a politeness classifier with the `politenessProjection()` function. This creates a single mapping from the politeness features in the supplied text to the covariate of interest. This can then be used to predict the covariate itself in held-out texts. In particular, if the user has some "ground truth" labels of politeness over a set of texts, they can use this function as a politeness classifier, and automatically assign politeness scores to many more new texts. This ground truth can be labelled by annotators in observational data, or, as in our example below, generated from a randomized experiment in which the text itself is the outcome.

This function is a wrapper around supervised learning algorithms. The default uses `glmnet` (Friedman et al., 2010), the vanilla LASSO implementation in R. We also allow users to use a different algorithm, `textir` (Taddy, 2013), which implements a massively multinomial inverse regression. Intuitively, this model represents a more realistic causal structure to text-plus-covariate data – that is, the metadata is typically an *ex ante* property of the speaker that has a causal effect on the words they use, rather than the words having a causal effect on the speaker's metadata. Both packages have their merits, though for now we recommend using `glmnet` to start, especially if its use is familiar.

In addition to the `phone_offers` dataset, we have included a smaller `bowl_offers` dataset (also from (Jeong et al., 2018)). Participants in this study were given similar instructions (i.e., communicate in a warm or tough style) but for a different negotiation exercise. We use the `phone_offers` dataset to define the construct of interest, and use the `bowl_offers` dataset as held-out data in `politenessProjection()`. The results confirm that the manipulation in the held-out data induced more politeness in one condition than the other.

In addition to the projected labels for the new documents, `politenessProjection()` also returns the coefficients estimated in the model. This provides some transparency regarding the exact functional form of politeness being generated. However, we caution users that the coefficients are chosen to maximize the prediction accuracy of the model of the whole, rather than recovering the "true" coefficient for any particular feature. In particular, these models may not reflect the first-order relationships in the data, especially when many relevant features are colinear with each other.

```
df_polite_train <- politeness(phone_offers$message, drop_blank=FALSE)

df_polite_holdout <- politeness(bowl_offers$message, drop_blank=FALSE)

project <- politenessProjection(df_polite_train,
                                phone_offers$condition,
                                df_polite_holdout)


t.test(project$test_proj[bowl_offers$condition==0],
        project$test_proj[bowl_offers$condition==1])

t = -6.4515, df = 66.914, p-value = 1.439e-08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.4216472 -0.2223876
sample estimates:
mean of x mean of y
0.2217517 0.5437691

project$train_coefs

   Positive.Emotion     Negative.Emotion         Negation   Informal.Title
         0.49911207           0.38236159      -1.04272459       0.10637694
        Subjunctive              For.You      Give.Agency            Hello
         0.92618688           0.11704362       0.48516687       0.60900150
First.Person.Plural            Questions        Gratitude         Actually
        -0.52233935           1.32231435       0.69019865       1.34960715
             Please  First.Person.Single    Second.Person
         0.04522580           0.03072077       0.03484301
```

We consistently recommend that researchers build a context-specific model of politeness for their own datasets, using labeled examples from within-domain. However, some new users may want to try this workflow before deciding whether to hand-label new documents in their own domain. In that case, users can follow the example above to build a rudimentary out-of-the-box politeness classifier with the `phone_offers` dataset and the `politenessProjection()` function). This analysis assumes that the rules of politeness in the user's domain are identical to the rules of politeness in negotiations, which may or may not hold.

Finding examples of polite and impolite documents

Before users apply any output from the `politenessProjection()` function to other analyses, they should first be curious about examples of texts that best represent the distinction made by that projection (i.e., the most- or least-typically polite texts()). The `findPoliteTexts()` function replicates the analyses of `politenessProjection` but instead returns a selection of the texts that are the most extreme (i.e., high, or low, or both) along the projected dimension. The parameters `type` and `num_docs` allow users to specify the type and number of texts that are returned.

```
set.seed(111)

fpt_most <- findPoliteTexts(phone_offers$message,
                            df_polite_train,
                            phone_offers$condition,
                            type="most",
                            num_docs=2)
fpt_least <- findPoliteTexts(phone_offers$message,
                             df_polite_train,
                             phone_offers$condition,
                        type="least",
                        num_docs=2)

fpt_most$text

[1] Hello,   Oh my goodness, I'm so excited to see your listing. The phone is EXACTLY
what I have been searching for! And I can tell from your description and the photo
that it's just perfect, too. I can almost hear it ringing in my pocket right now!
What's that? Hello? So, happy to hear from you. Sorry, sorry. I'm getting ahead of
myself.
Soooo, I was wondering if there was any way that you might consider taking a little bit
less for this phone? It's absolutely everything I've been looking for but my bosses
&lt; grrrrr!!!&gt; will only give us $115 to get this phone. I know! Can you believe
it? Is there any way you could absolutely make my day and say ""yes"" to $115? I'd
be so, so grateful. Just let me know when you can. Thanks so much. :)

[2] Hello,  I am glad you're selling this phone. Is it still available? I would love to
purchase it. Would you consider $115 for it? I can buy it today if that price works.
Thank you.



fpt_least$text

[1] I am inquiring about your Iphone 6 plus that you had posted.  I am wanting to buy
and I have cash in hand.  The max amount I can offer is $115.  No more, no less.
Let me know.

[2] Come on. The price you are offering on a product that ISN'T NEW is unreasonable.
Now, I for one am very interested in getting this item. BUT, I will only pay $115.
I am not paying a penny more.
```

Execution time

In principle, these functions can handle an arbitrarily large set of documents. In practice, however, language data can be quite large, both in the amount of documents and the length of the documents. This can have a marked effect on execution time, especially on smaller computers.

In order to reduce computation time, we use **data.table** (Dowle and Srinivasan, 2017) and **quanteda** (Benoit et al., 2018) in the backend which have fast implementations of data manipulation. We also analyzed the execution time of intermediate steps within `politeness()` to identify any potential critical points.

To provide rough benchmarks, we ran the `politeness()` function with a range of document counts ($10^2, 10^3, 10^4$) and lengths (100, 200). The tests were performed using a 2012 Macbook Pro with a 2.5 GHz Intel Core i5. For each case we ran it three times, both with and without part-of-speech tagging, and the resulting execution times are plotted below.

Figure 3 shows that `politeness()` scales reasonably well for large sets of documents. For example, given 200-word documents, and using the spacy parser, we found that 100 and 1,000 length vectors of texts take an average of 7.3 and 70.6 seconds, respectively. We recommend that for larger corpora, researchers test the code on smaller subsets first.

A common and complex situation is when text is broken up into many documents per observation. One example of this would be a conversation transcript, in which people take many turns over a single conversation, but the covariates are measured at the level of the person, or conversation. In these cases, we recommend calculating the politeness markers in individual documents separately, and then aggregated into person-level counts afterwards, for plotting, analysis, and model-building. This will be more efficient and more accurate than concatenating each person's turns into one long document and calculating the politeness of the entire text at once.

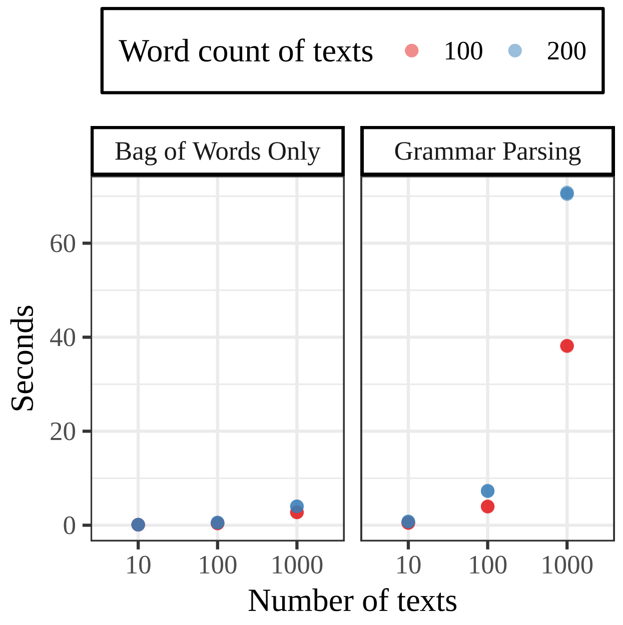


Figure 3: Execution time of `politeness()` with different number of texts, length of texts, and parsers.

Extended Example: Politeness in Speed Dating

We demonstrate the broader applications of our package in a new context: courtship, in face-to-face conversation with many turns per speaker. This example comes from the SpeedDate corpus (originally published in (Ranganath et al., 2009; McFarland et al., 2013). Here, politeness is not a treatment effect. Instead, we model the naturally-occurring variation in politeness (as rated by listeners) during a series of speed dating events. Furthermore, we highlight the context-specificity of politeness—we compare the linguistic markers of politeness in both female speakers (as rated by men) and male speakers (as rated by women), while holding constant the domain, the outcome measure, and the conversation itself.

The data were collected over three evenings, in which 110 heterosexual participants each met with 15-20 other potential partners for four minutes at a time. The main observations are gathered from each dater, from a survey immediately after their date. For privacy reasons, we cannot include the text of the dates themselves. However, we do include the matrix of politeness features as calculated by the `politeness()` function.

Note that the original function call treats each turn in the dataset as a separate document, to produce a one-row-per-turn matrix. So to analyze these data within our framework we need to condense the politeness feature matrix to the same one-row-per-person-date level. To do this here, we loop through each row of the per-person-date data, and add up the counts from the relevant section of the per-turn database.

```
load("speedDateDates.rdata")
load("speedDateTurns.rdata")

id.vars<c("selfid","otherid","turn","span","group")
polite.cols<-names(speedDateTurns)[!names(speedDateTurns) %in% id.vars]
speedDateDates[,polite.cols] <- NA
for (o in 1:nrow(speedDateDates)) {
  matched.rows<-(speedDateTurns$selfid==speedDateDates[o, "otherid"])
               & (speedDateTurns$otherid==speedDateDates[o, "selfid"])
  speedDateDates[o,polite.cols] <- colSums(speedDateTurns[matched.rows,polite.cols],
                                          na.rm=T)
}
```

We wanted to include extra domain-specific features, to capture elements of live face-to-face conversation. These were calculated separately, and merged into the politeness feature matrix. First, we counted moments of laughter, which were indicated by the transcribers. We also included repair questions (e.g., "pardon?") using a list from Ranganath et al. (2009). Finally, we included four kinds

	Followup.Qs	Switch.Qs	Intro.Qs	Mirror.Qs	Repair.Qs	Laughter
Spoken by Women	4.04	2.89	0.26	1.85	0.13	4.42
Spoken by Men	5.11	3.51	0.38	1.73	0.17	1.83

Table 3: Prevalence of features added to politeness detector, by gender.

of question types from Huang et al. (2017)—switch questions, which change the topic; follow-up questions, which expand on the current topic; introductory questions, which open a dialogue; and mirror questions, in which one person returns a question that they had just been asked. By merging these six extra features into the politeness feature matrix, downstream functions will treat them as though they were part of the original feature set.

```
extra_names<-c(paste0(c("Followup", "Switch", "Intro", "Mirror", "Repair"), ".Qs"),
               "Laughter")

extras<-rbind(round(colMeans(speedDateDates[(speedDateDates$sex==1),extra_names]), 2),
              round(colMeans(speedDateDates[(speedDateDates$sex==0),extra_names]), 2))
row.names(extras)<-c("Spoken by Women (to Men)", "Spoken by Men (to Women)")

extras
```

Immediately after each date, both participants evaluated their partner on a set of dimensions. The one most relevant for our purposes was the question "how courteous was your partner", which was rated on a scale from 1-10 (plotted in Figure 4). We use this as a ground truth measure of politeness. Because politeness was measured (not manipulated) we must dichotomize the continuous variable into two groups. We drop the middle third of medium-courtesy dates and compare the dates that were rated as most and least rude, to increase contrast. For clarity we do this manually in the example below, but the package also does this automatically if the provided labels are on a continuous scale.

```
speedDateDates$politeness <- ""
speedDateDates$politeness[speedDateDates$courtesy > 8] <- "Courteous"
speedDateDates$politeness[speedDateDates$courtesy < 6] <- "Rude"
speedDateDates<-speedDateDates[speedDateDates$politeness != "", ]
```

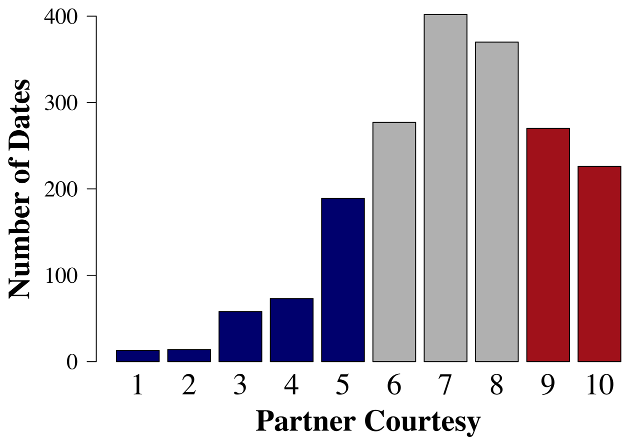


Figure 4: Distribution of rated courtesy in SpeedDate corpus.

Finally, we wanted to know whether the rules by which politeness is judged were different for men and for women. The average courtesy rating of men by women was slightly lower than that of the women by men. But these summary ratings do not tell us anything about how rudeness or courtesy is determined. Instead, we apply the `politenessPlot()` function among each gender separately to determine gender-specific models for how politeness is related to the linguistic choices of the speaker.

```
politenessPlot(speedDateDates[(speedDateDates$sex==1),
                c(polite.cols, extra_names)],
                split=speedDateDates$politeness[(speedDateDates$sex==1)],
                split_name="Ratings of Women (by Men)",
                top_title="",
                middle_out=0.1)


politenessPlot(speedDateDates[(speedDateDates$sex==0),
                c(polite.cols, extra_names)],
                split=speedDateDates$politeness[(speedDateDates$sex==0)],
                split_name="Ratings of Men (by Women)",
                top_title="",
                middle_out=0.1)
```

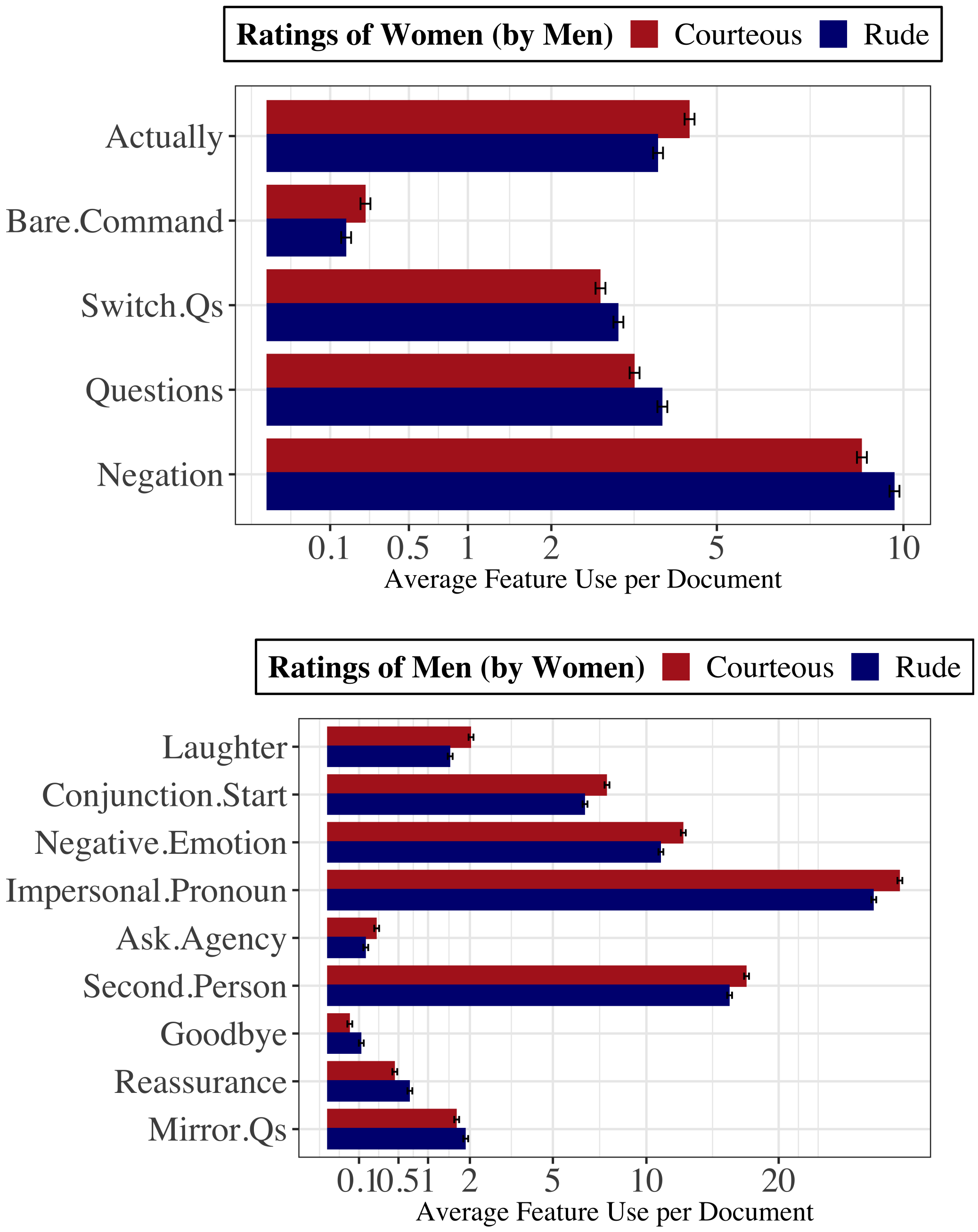


Figure 5: Politeness markers in speed-dating by speaker gender.

The results in Figure 5 demonstrate that the rules for what is considered polite behavior from a man or a woman can be quite different, even the same conversation. Men laugh less often than women, overall; but when they do laugh it is associated with courtesy by their partner. Men who are considered rude tend to give formal goodbyes as they are leaving, and more often respond to a question by mirroring it back to their partner. On the other hand, women are seen as rude when they contradict, or when they ask questions (in particular, topic-switching questions) but are seen as polite when they reveal more about themselves—the "actually" feature captures pivots to greater detail, such as "in fact" or "to be honest." Interpreting these features can be difficult without the text in hand, though, and users are encouraged to incorporate qualitative analyses of their own data, in addition to the analyses provided here.

We note that the differences in feature use between polite and impolite daters (of both genders) are not as stark in this example as in the negotiation study. This is for two reasons. First, the variation in politeness is naturally occurring, rather than induced by a randomized treatment. Contrasts will be greater when writers are instructed and able to adopt a particular communication style. Secondly, the speed daters communicated face-to-face, so there could be other sources of information that would affect the listener's rating of a speaker's politeness (such as intonation, body language, pacing, and so on). In datasets where they are measured, these other features could be added to the workflow in the same way as the extra text-based features above.

Conclusions

We detail an empirical model of politeness, as codified in the **politeness** package. We use simple examples to show the range of output from the `politeness()` and `politenessPlot()` functions. We also show how `politenessProjection()` can be used to develop a domain-specific supervised model for politeness, which can be used by `findPoliteTexts()` to identify and explore distinctive documents. We also work through two examples of consequential politeness—as manipulated amongst negotiators, and as observed amongst speed daters—that highlight many challenges that are common for all sorts of research in R that uses natural language data from social interactions.

The tools presented here should be useful for all researchers who study how people interact with one another, across a wide variety of contexts. In future work, we hope to expand this toolkit to handle a wider range of politeness markers that are not present in proper written english (including shorthand, slang, nonverbals, and other languages). Further research is also needed into the extent to which the markers of politeness can vary from one context to another, and the consequences of politeness on interpersonal relations.

Acknowledgements

We thank the Spencer Foundation, the Hewlett Foundation, Harvard's Institute for Quantitative Social Science, and the Harvard Vice Provost for Advances in Learning Research Group for their support.

Bibliography

K. Benoit and A. Matsuo. **spacyr**: *Wrapper to the 'spaCy' 'NLP' Library*, 2017. URL `https://CRAN.R-project.org/package=spacyr`. R package version 0.9.2. [p491]

K. Benoit, K. Watanabe, H. Wang, P. Nulty, A. Obeng, S. Müller, and A. Matsuo. quanteda: An r package for the quantitative analysis of textual data. *Journal of Open Source Software*, 3(30):774, 2018. doi: 10.21105/joss.00774. URL `https://quanteda.io`. [p497]

P. Brown and S. C. Levinson. Politeness: Some universals in language usage. *Studies in interactional sociolinguistics*, 4, 1987. URL `http://psycnet.apa.org/record/1987-97641-000`. [p489, 490]

C. Danescu-Niculescu-Mizil, M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts. A computational approach to politeness with application to social factors. *CoRR*, abs/1306.6078, 2013. URL `http://arxiv.org/abs/1306.6078`. [p489, 490]

M. Dowle and A. Srinivasan. *Data.table: Extension of 'data.frame'*, 2017. URL `https://CRAN.R-project.org/package=data.table`. R package version 1.10.4-3. [p497]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL `http://www.jstatsoft.org/v33/i01/`. [p495]

E. Goffman. On face-work. *Interaction Rituals*, pages 5–45, 1967. [p489, 490]

J. Grimmer and B. Stewart. Text as data: The promise and pitfalls of automatic content analysis. *Political Analysis*, 21(3):267–297, 2013. [p489]

M. Honnibal and M. Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, 2015. Association for Computational Linguistics. URL https://aclweb.org/anthology/D/D15/D15-1162. [p491]

K. Huang, M. Yeomans, A. W. Brooks, J. Minson, and F. Gino. It doesn't hurt to ask: Question-asking increases liking. *Journal of Personality and Social Psychology*, 113(3):430–452, 2017. [p499]

M. Jeong, J. Minson, M. Yeomans, and F. Gino. Communicating warmth in distributive negotiations is surprisingly counter-productive. *Working Paper*, 2018. [p492, 495]

D. Jurafsky and J. Martin. *Speech and Language Processing*. Pearson, 2014. [p489]

R. T. Lakoff. *The Logic of Politeness: Minding Your P's and Q's*. Chicago Linguistic Society, 1973. URL https://books.google.com/books?id=DfWfNAAACAAJ. [p489, 490]

C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999. [p489]

D. McFarland, D. Jurafsky, and R. Ranganath. Making the connection: Social bonding in courtship situations. *American Journal of Sociology*, 118(6):1596–1649, 2013. [p498]

R. Ranganath, D. Jurafsky, and D. McFarland. It's not you, it's me: Detecting flirting and its misperception in speed-dates. *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 1(1):334–342, 2009. [p498]

M. Taddy. Multinomial inverse regression for text analysis. *Journal of the American Statistical Association*, 108(503):755–770, 2013. doi: 10.1080/01621459.2012.734168. URL https://doi.org/10.1080/01621459.2012.734168. [p495]

R. Voigt, N. P. Camp, V. Prabhakaran, W. L. Hamilton, R. C. Hetey, C. M. Griffiths, D. Jurgens, D. Jurafsky, and J. L. Eberhardt. Language from police body camera footage shows racial disparities in officer respect. *Proceedings of the National Academy of Sciences*, 114(25):6521–6526, 2017. ISSN 0027-8424. URL https://doi.org/10.1073/pnas.1702413114. [p489, 490]

Michael Yeomans
Harvard University
Cambridge, MA 02138, USA
yeomans@fas.harvard.edu

Alejandro Kantor
Institute for Quantitative Social Science
Harvard University
Cambridge, MA 02138, USA
alejandrokantor@fas.harvard.edu

Dustin Tingley
Department of Government
Harvard University
Cambridge, MA 02138, USA
dtingley@gov.harvard.edu

ShinyItemAnalysis for Teaching Psychometrics and to Enforce Routine Analysis of Educational Tests

by Patrícia Martinková, Adéla Drabinová

Abstract This work introduces **ShinyItemAnalysis**, an R package and an online shiny application for psychometric analysis of educational tests and items. **ShinyItemAnalysis** covers a broad range of psychometric methods and offers data examples, model equations, parameter estimates, interpretation of results, together with a selected R code, and is therefore suitable for teaching psychometric concepts with R. Furthermore, the application aspires to be an easy-to-use tool for analysis of educational tests by allowing the users to upload and analyze their own data and to automatically generate analysis reports in PDF or HTML. We argue that psychometric analysis should be a routine part of test development in order to gather proofs of reliability and validity of the measurement, and we demonstrate how **ShinyItemAnalysis** may help enforce this goal.

Introduction

Assessments that are used to measure students' ability or knowledge need to produce valid, reliable and fair scores (Brennan 2006; Downing and Haladyna 2011; AERA, APA and NCME 2014). While many R packages have been developed to cover general psychometric concepts (e. g. **psych** (Revelle, 2018), **ltm** (Rizopoulos, 2006)) or specific psychometric topics (e. g. **difR** (Magis et al., 2010), **lavaan** (Rosseel, 2012), see also *Psychometrics*), stakeholders in this area are often non-programmers and thus, may find it difficult to overcome the initial burden of an R-based environment. Commercially available software provides an alternative but high prices and limited methodology may be an issue. Nevertheless, it is of high importance to enforce routine psychometric analysis in development and validation of educational tests of various types worldwide. Freely available software with user-friendly interface and interactive features may help this enforcement even in regions or scientific areas where understanding and usage of psychometric concepts is underdeveloped.

In this work we introduce **ShinyItemAnalysis** (Martinková et al., 2018a) - an R package and an online application based on **shiny** (Chang et al., 2018) which was initially created to support teaching of psychometric concepts and test development, and subsequently used to enforce routine validation of admission tests to Czech universities (Martinková et al., 2017). Its current mission is to support routine validation of educational and psychological measurements worldwide.

We first briefly explain the methodology and concepts in a step-by-step way, from the classical test theory (CTT) to the item response theory (IRT) models, including methods to detect differential item functioning (DIF). We then describe the implementation of **ShinyItemAnalysis** with practical examples coming from development and validation of the Homeostasis Concept Inventory (HCI, McFarland et al., 2017). We conclude with discussion of features helpful for teaching psychometrics, as well as features important for generation of PDF and HTML reports, enforcing routine analysis of admission and other educational tests.

Psychometric models for analysis of items and tests

Classical test theory

Traditional item analysis uses counts, proportions and correlations to evaluate properties of test items. Difficulty of items is estimated as the percentage of students who answered correctly to that item. Discrimination is usually described as the difference between the percent correct in the upper and lower third of the students tested (upper-lower index, ULI). As a rule of thumb, ULI should not be lower than 0.2, except for very easy or very difficult items (Ebel, 1954). ULI can be customized by determining the number of groups and by changing which groups should be compared: this is especially helpful for admission tests where a certain proportion of students (e. g. one fifth) are usually admitted (Martinková et al., 2017).

Other traditional statistics for a description of item discrimination include the point-biserial correlation, which is the Pearson correlation between responses to a particular item and scores on the total test. This correlation (R) is denoted RIT index if an item score (I) is correlated with the total score (T) of the whole test, or RIR if an item score (I) is correlated with the sum of the rest of the items (R).

In addition, difficulty and discrimination may be calculated for each response of the multiple choice item to evaluate distractors and diagnose possible issues, such as confusing wording. Respondents are divided into N groups (usually 3, 4 or 5) by their total score. Subsequently, the percentage of students in each group which selected a given answer (correct answer or distractor) is calculated and may be displayed in a distractor plot.

To gain empirical proofs of the construct validity of the whole instrument, correlation structure needs to be examined. Empirical proofs of validity may be provided by correlation with a criterion variable. For example, a correlation with subsequent university success or university GPA may be used to demonstrate predictive validity of admission scores.

To gain proofs of test reliability, internal consistency of items can be examined using Cronbach's alpha (Cronbach 1951, see also Zinbarg et al. 2005 for more discussion). Cronbach's alpha of test without a given item can be used to determine items not internally consistent with the rest of the test.

Regression models for description of item properties

Various regression models may be fitted to describe item properties in more detail. With binary data, logistic regression may be used to model the probability of a correct answer as a function of the total score by an S-shaped logistic curve. Parameter $b_0 \in R$ describes horizontal location of the fitted curve, parameter $b_1 \in R$ describes its slope.

$$\mathrm{P}\left(Y=1|X,b_0,b_1\right)=\frac{\exp\left(b_0+b_1X\right)}{1+\exp\left(b_0+b_1X\right)}. \tag{1}$$

A standardized total score may be used instead of a total score as an independent variable to model the properties of a given item. In such a case, the estimated curve remains the same, only the interpretation of item properties now focuses on improvement of 1 standard deviation rather than 1 point improvement. It is also helpful to re-parametrize the model using new parameters $a \in R$ and $b \in R$. Item difficulty parameter b is given by location of the inflection point, and item discrimination parameter a is given by the slope at the inflection point:

$$\mathrm{P}(Y=1|Z,a,b)=\frac{\exp(a(Z-b))}{1+\exp(a(Z-b))}. \tag{2}$$

Further, non-linear regression models allow us to account for guessing by providing non-zero left asymptote $c \in [0,1]$ and inattention by providing right asymptote $d \in [0,1]$:

$$\mathrm{P}\left(Y=1|Z,a,b,c,d\right)=c+\left(d-c\right)\frac{\exp\left(a\left(Z-b\right)\right)}{1+\exp\left(a\left(Z-b\right)\right)}. \tag{3}$$

Other regression models allow for further extensions or different data types: Ordinal regression allows for modelling Likert-scale and partial-credit items. To model responses to all given options (correct response and all distractors) in multiple-choice questions, a multinomial regression model can be used. Further complexities of the measurement data and item functioning may be accounted for by incorporating student characteristics with multiple regression models or clustering with hierarchical models.

A logistic regression model (1) for item description, and its re-parametrizations and extensions as illustrated in equations (2) and (3) may serve as a helpful introductory step for explaining and building IRT models which can be conceptualized as (logistic/non-linear/ordinal/multinomial) mixed effect regression models (Rijmen et al., 2003).

Item response theory models

IRT models assume respondent abilities being unobserved/latent scores θ which need to be estimated together with item parameters. 4-parameter logistic (4PL) IRT model corresponds to regression model (3) above

$$\mathrm{P}\left(Y=1|\theta,a,b,c,d\right)=c+\left(d-c\right)\frac{\exp\left(a\left(\theta-b\right)\right)}{1+\exp\left(a\left(\theta-b\right)\right)}. \tag{4}$$

Similarly, other submodels of 4PL model (4) may be obtained by fixing appropriate parameters. E. g. the 3PL IRT model is obtained by fixing inattention to $d=1$. The 2PL IRT model is obtained by further fixing pseudo-guessing to $c=0$, and the Rasch model by fixing discrimination $a=1$ in addition.

Other IRT models allow for further extensions or different data types: Modelling Likert-scale and partial-credit items can be done by modelling cumulative responses in a graded response model (GRM, Samejima, 1969). Alternatively, ordinal items may be analyzed by modelling adjacent categories logits using a generalized partial credit model (GPCM, Muraki, 1992), or its restricted version - partial credit model (PCM, Masters, 1982), and rating scale model (RSM, Andrich, 1978). To model distractor properties in multiple-choice items, Bock's nominal response model (NRM, Bock, 1972) is an IRT analogy of a multinomial regression model. This model is also a generalization of GPCM/PCM/RSM ordinal models. Many other IRT models have been used in the past to model item properties, including models accounting for multidimensional latent traits, hierarchical structure or response times (van der Linden, 2017).

A wide variety of estimation procedures has been proposed in last decades. Joint maximum likelihood estimation treats both ability and item parameters as unknown but fixed. Conditional maximum likelihood estimation takes an advantage of the fact that in exponential family models (such as in the Rasch model), total score is a sufficient statistics for an ability estimate and the ratio of correct answers is a sufficient statistics for a difficulty parameter. Finally, in a marginal maximum likelihood estimation used by the **mirt** (Chalmers, 2018) and **ltm** (Rizopoulos, 2006) package as well as in **ShinyItemAnalysis**, parameter θ is assumed to be a random variable following certain distribution (often standard normal) and is integrated out (see for example, Johnson, 2007). An EM algorithm with a fixed quadrature is used in latent scores and item parameters estimation. Besides MLE approaches, Bayesian methods with the Markov chain Monte Carlo are a good alternative, especially for multidimensional IRT models.

Differential item functioning

DIF occurs when respondents from different groups (e. g. such as defined by gender or ethnicity) with the same underlying true ability have a different probability of answering the item correctly. Differential distractor functioning (DDF) is a phenomenon when different distractors, or incorrect option choices, attract various groups with the same ability differentially. If an item functions differently for two groups, it is potentially unfair, thus detection of DIF and DDF should be routine analysis when developing and validating educational and psychological tests (Martinková et al., 2017).

Presence of DIF can be tested by many methods including Delta plot (Angoff and Ford, 1973), Mantel-Haenszel test based on contingency tables that are calculated for each level of a total score (Mantel and Haenszel, 1959), logistic regression models accounting for group membership (Swaminathan and Rogers, 1990), nonlinear regression (Drabinová and Martinková, 2017), and IRT based tests (Lord, 1980; Raju, 1988, 1990).

Implementation

The **ShinyItemAnalysis** package can be used either locally or online. The package uses several other R packages to provide a wide palette of psychometric tools to analyze data (see Table 1). The main function is called `startShinyItemAnalysis()`. It launches an interactive shiny application (Figure 1) which is further described below. Furthermore, function `gDiscrim()` calculates generalized coefficient ULI comparing the ratio of correct answers in predefined upper and lower groups of students (Martinková et al., 2017). Function `ItemAnalysis()` provides a complete traditional item analysis table with summary statistics and various difficulty and discrimination indices for all items. Function `DDplot()` plots difficulty and selected discrimination indices of the items ordered by their difficulty. Function `DistractorAnalysis()` calculates the proportions of choosing individual distractors for groups of respondents defined by their totals score. Graphical representation of distractor analysis is provided via function `plotDistractorAnalysis()`. Other functions include item - person maps for IRT models, `ggWrightMap()`, and plots for DIF analysis using IRT methods, `plotDIFirt()`, and logistic regression models, `plotDIFlogistic()`. These functions may be applied directly on data from an R console as shown in the provided R code. The package also includes training datasets Medical 100, Medical 100 graded (Martinková et al., 2017), and HCI (McFarland et al., 2017).

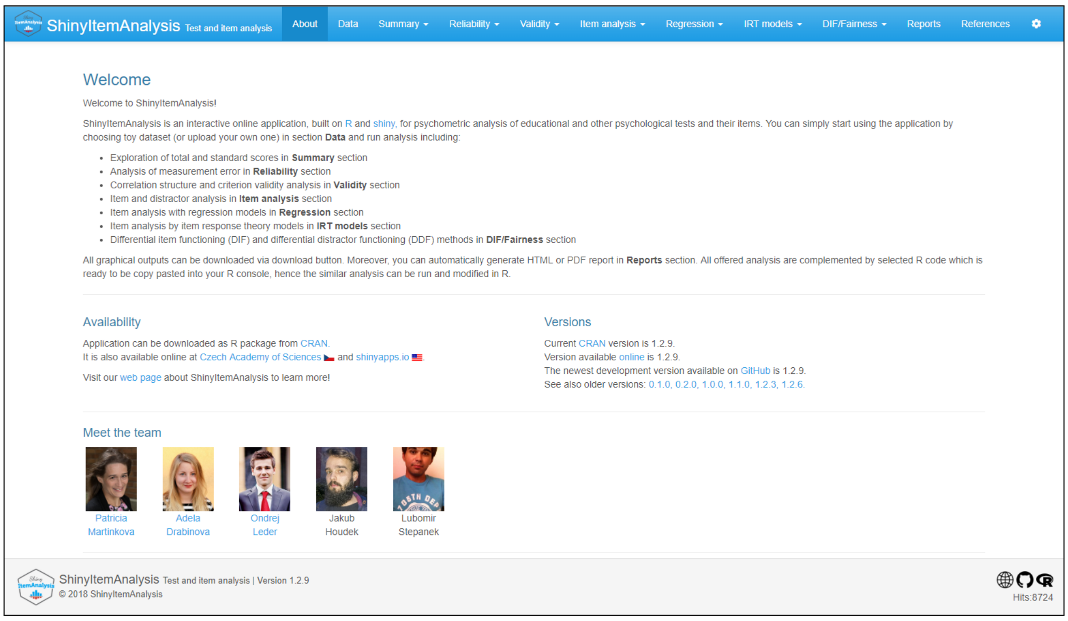
Examples

Running the application

The **ShinyItemAnalysis** application may be launched in R by calling `startShinyItemAnalysis()`, or more conveniently, directly from `https://shiny.cs.cas.cz/ShinyItemAnalysis`. The intro page (Figure 1) includes general information about the application. Various tools are included in separate tabs which correspond to separate sections.

R package	Citation	Title
corrplot	(Wei and Simko, 2017)	Visualization of a correlation matrix
cowplot	(Wilke, 2018)	Streamlined plot theme and plot annotations for 'ggplot2'
CTT	(Willse, 2018)	Classical test theory functions
data.table	(Dowle and Srinivasan, 2018)	Extension of `data.frame`
deltaPlotR	(Magis and Facon, 2014)	Identification of dichotomous differential item functioning using Angoff's delta plot method
difNLR	(Drabinová et al., 2018)	DIF and DDF detection by non-linear regression models
difR	(Magis et al., 2010)	Collection of methods to detect dichotomous differential item functioning
DT	(Xie et al., 2018)	A wrapper of the JavaScript library 'DataTables'
ggdendro	(de Vries and Ripley, 2016)	Create dendrograms and tree diagrams using 'ggplot2'
ggplot2	(Wickham, 2016)	Create elegant data visualisations using the grammar of graphics
gridExtra	(Auguie, 2017)	Miscellaneous functions for 'grid' graphics
knitr	(Xie, 2018)	A general-purpose package for dynamic report generation in R
latticeExtra	(Sarkar and Andrews, 2016)	Extra graphical utilities based on lattice
ltm	(Rizopoulos, 2006)	Latent trait models under IRT
mirt	(Chalmers, 2018)	Multidimensional item response theory
moments	(Komsta and Novomestky, 2015)	Moments, cumulants, skewness, kurtosis and related tests
msm	(Jackson, 2011)	Multi-state Markov and hidden Markov models in continuous time
nnet	(Venables and Ripley, 2002)	Feed-forward neural networks and multinomial log-linear models
plotly	(Sievert, 2018)	Create interactive web graphics via 'plotly.js'
psych	(Revelle, 2018)	Procedures for psychological, psychometric, and personality research
psychometric	(Fletcher, 2010)	Applied psychometric theory
reshape2	(Wickham, 2007)	Flexibly reshape data: A reboot of the reshape package
rmarkdown	(Allaire et al., 2018)	Dynamic documents for R
shiny	(Chang et al., 2018)	Web application framework for R
shinyBS	(Bailey, 2015)	Twitter bootstrap components for shiny
shinydashboard	(Chang and Borges Ribeiro, 2018)	Create dashboards with 'shiny'
shinyjs	(Attali, 2018)	Easily improve the user experience of your shiny apps in seconds
stringr	(Wickham, 2018)	Simple, consistent wrappers for common string operations
xtable	(Dahl et al., 2018)	Export tables to LaTeXor HTML

Table 1: R packages used for developing **ShinyItemAnalysis**.



Welcome

Welcome to ShinyItemAnalysis!

ShinyItemAnalysis is an interactive online application, built on R and shiny, for psychometric analysis of educational and other psychological tests and their items. You can simply start using the application by choosing toy dataset (or upload your own one) in section **Data** and run analysis including:

- Exploration of total and standard scores in **Summary** section
- Analysis of measurement error in **Reliability** section
- Correlation structure and criterion validity analysis in **Validity** section
- Item and distractor analysis in **Item analysis** section
- Item analysis with regression models in **Regression** section
- Item analysis by item response theory models in **IRT models** section
- Differential item functioning (DIF) and differential distractor functioning (DDF) methods in **DIF/Fairness** section

All graphical outputs can be downloaded via download button. Moreover, you can automatically generate HTML or PDF report in **Reports** section. All offered analysis are complemented by selected R code which is ready to be copy pasted into your R console, hence the similar analysis can be run and modified in R.

Availability

Application can be downloaded as R package from CRAN.
It is also available online at Czech Academy of Sciences 🇨🇿 and shinyapps.io 🇺🇸.

Visit our web page about ShinyItemAnalysis to learn more!

Versions

Current CRAN version is 1.2.9.
Version available online is 1.2.9.
The newest development version available on GitHub is 1.2.9.
See also older versions: 0.1.0, 0.2.0, 1.0.0, 1.1.0, 1.2.3, 1.2.6.

Meet the team

Patricia Martinkova, Adela Drabinova, Ondrej Leder, Jakub Houdek, Lubomir Stepanek

ShinyItemAnalysis Test and item analysis | Version 1.2.9
© 2018 ShinyItemAnalysis

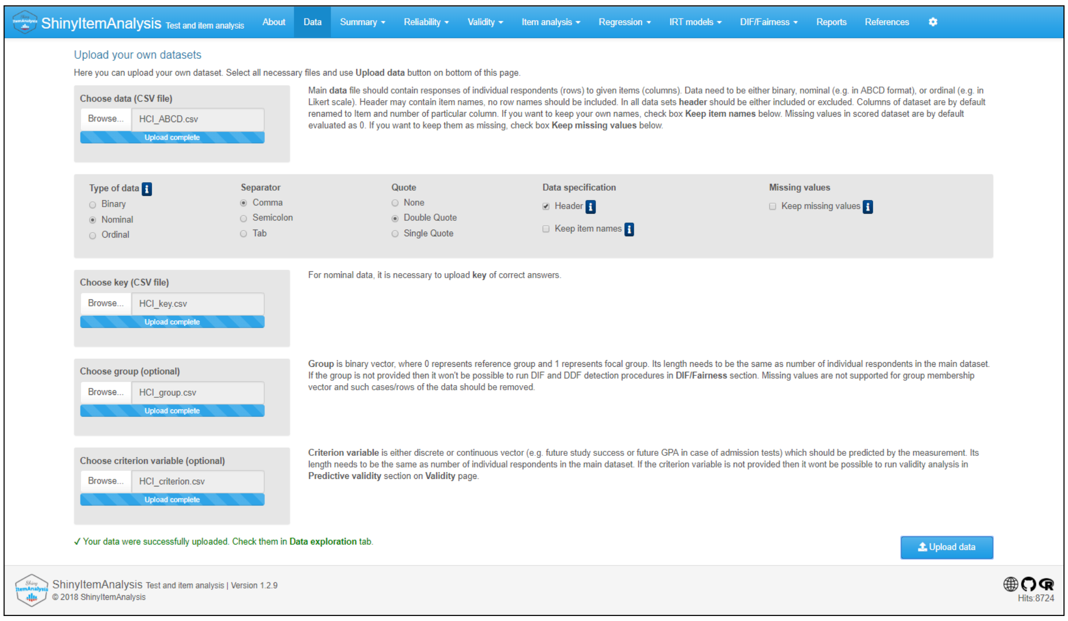
Hits:8724

Figure 1: Intro page.

Data selection and upload

Data selection is available in section **Data**. Six training datasets may be uploaded using the **Select dataset** button: Training datasets Medical 100, Medical 100 graded (Martinková et al., 2017) and HCI (McFarland et al., 2017) from the **ShinyItemAnalysis** package, and datasets GMAT (Martinková et al., 2017), GMAT2 and MSAT-B (Drabinová and Martinková, 2017) from the **difNLR** package (Drabinová et al., 2018).

Besides the provided toy datasets, users' own data may be uploaded as csv files and previewed in this section. To replicate examples involving the HCI dataset (McFarland et al., 2017), csv files are provided for upload in Supplemental Materials.



Upload your own datasets

Here you can upload your own dataset. Select all necessary files and use **Upload data** button on bottom of this page.

Choose data (CSV file)
Browse... HCI_ABCD.csv
Upload complete

Main **data** file should contain responses of individual respondents (rows) to given items (columns). Data need to be either binary, nominal (e.g. in ABCD format), or ordinal (e.g. in Likert scale). Header may contain item names, no row names should be included. In all data sets **header** should be either included or excluded. Columns of dataset are by default renamed to Item and number of particular column. If you want to keep your own names, check box **Keep item names** below. Missing values in scored dataset are by default evaluated as 0. If you want to keep them as missing, check box **Keep missing values** below.

Type of data: Binary [✓ Nominal] Ordinal

Separator: [✓ Comma] Semicolon Tab

Quote: None [✓ Double Quote] Single Quote

Data specification: [✓ Header] Keep item names

Missing values: Keep missing values

Choose key (CSV file)
Browse... HCI_key.csv
Upload complete

For nominal data, it is necessary to upload **key** of correct answers.

Choose group (optional)
Browse... HCI_group.csv
Upload complete

Group is binary vector, where 0 represents reference group and 1 represents focal group. Its length needs to be the same as number of individual respondents in the main dataset. If the group is not provided then it won't be possible to run DIF and DDF detection procedures in **DIF/Fairness** section. Missing values are not supported for group membership vector and such cases/rows of the data should be removed.

Choose criterion variable (optional)
Browse... HCI_criterion.csv
Upload complete

Criterion variable is either discrete or continuous vector (e.g. future study success or future GPA in case of admission tests) which should be predicted by the measurement. Its length needs to be the same as number of individual respondents in the main dataset. If the criterion variable is not provided then it wont be possible to run validity analysis in **Predictive validity** section on **Validity** page.

✓ Your data were successfully uploaded. Check them in **Data exploration** tab.

Upload data

ShinyItemAnalysis Test and item analysis | Version 1.2.9
© 2018 ShinyItemAnalysis

Hits:8724

Figure 2: Page to select or upload data.

Item analysis step-by-step

Further sections of the **ShinyItemAnalysis** application allow for step-by-step test and item analysis. The first four sections are devoted to traditional test and item analyses in a framework of classical test theory. Further sections are devoted to regression models, to IRT models and to DIF methods. A separate section is devoted to report generation and references are provided in the final section. The individual sections are described below in more detail.

Section **Summary** provides for histogram and summary statistics of the total scores as well as for various standard scores (Z scores, T scores), together with percentile and success rate for each level of the total score. Section **Reliability** offers internal consistency estimates with Cronbach's alpha.

Section **Validity** provides correlation structure (Figure 3, left) and checks of criterion validity (Figure 3, right). A correlation heat map displays selected type of correlation estimates between items. Polychoric correlation is the default correlation used for binary data. The plot can be reordered using hierarchical clustering while highlighting a specified number of clusters. Clusters of correlated items need to be checked for content and other similarities to see if they are intended. Criterion validity is analyzed with respect to selected variables (e. g. subsequent study success, major, or total score on a related test) and may be analyzed for the test as a whole or for individual items.

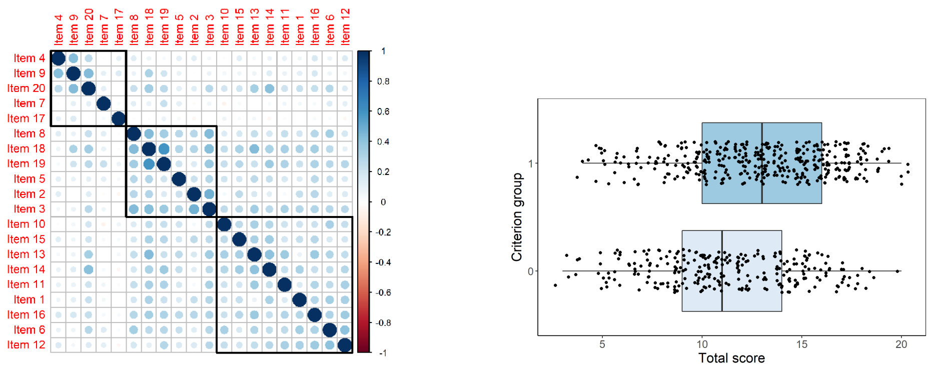


Figure 3: Validity plots for HCI data.

Section **Item analysis** offers traditional item analysis of the test as well as a more detailed distractor analysis. The so called DD plot (Figure 4) displays difficulty (red) and a selected discrimination index (blue) for all items. Items are ordered by difficulty. While lower discrimination may be expected for very easy or very hard items, all items with ULI discrimination lower than 0.2 (borderline in the plot) are worth further checks by content experts. The distractor plot (Figure 5) provides for detailed analysis of individual distractors by the respondents' total score. The correct answer should be selected more often by strong students than by students with a lower total score, i. e. the solid line in the distractor plot (see Figure 5) should be increasing. The distractors should work in an opposite direction, i. e. the dotted lines should be decreasing.

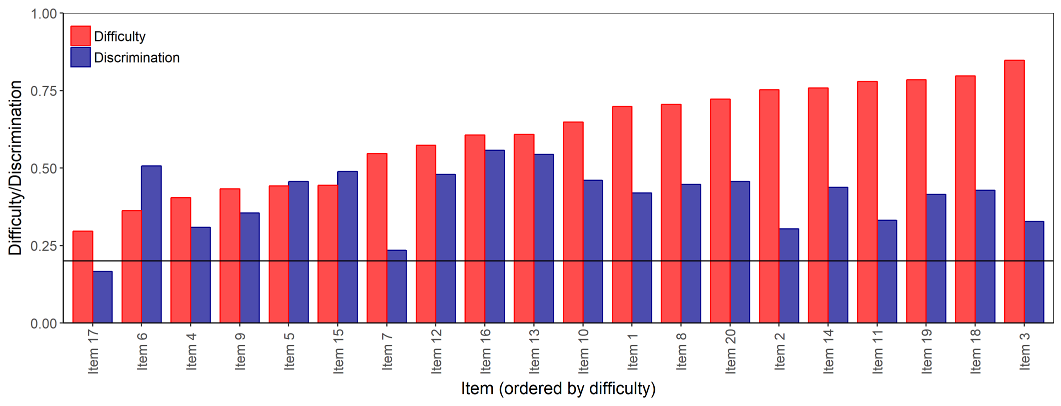


Figure 4: DD plot for HCI data.

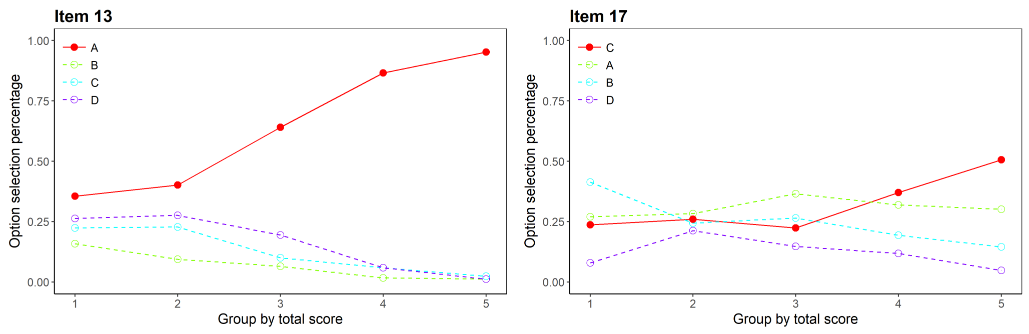


Figure 5: Distractor plots for items 13 and 17 in HCI data.

Section **Regression** allows for modelling of item properties with a logistic, non-linear or multinomial regression (see Figures 6). Probability of the selection of a given answer is modelled with respect to the total or standardized total score. Classical as well as IRT parametrization are provided for logistic and non-linear models. Model fit can be compared by Akaike's (Akaike, 1974) or Schwarz's Bayesian (Schwarz, 1978) information criteria and a likelihood-ratio test.

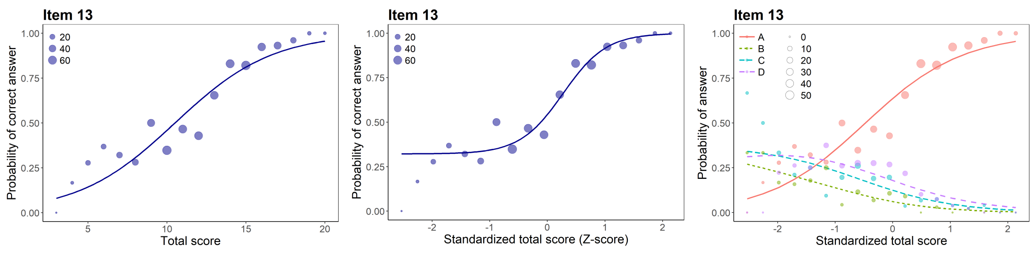


Figure 6: Regression plots for item 13 in HCI data.

Section **IRT models** provides for 1-4PL IRT models as well as Bock's nominal model, which may also be used for ordinal items. In IRT model specification, **ShinyItemAnalysis** uses default settings of the **mirt** package and for the Rasch model (Rasch, 1960) it fixes discrimination to $a = 1$, while variance of ability θ is freely estimated. Contrary to Rasch model, 1PL model allows any discrimination $a \in R$ (common to all items), while fixing variance of ability θ to 1. Similarly, other submodels of 4PL model

(4), e. g. 2PL and 3PL model, may be obtained by fixing appropriate parameters, while variance of ability θ is fixed to 1.

Interactive item characteristic curves (ICC), item information curves (IIC) and test information curves (TIC) are provided for all IRT models (see Figure 7). An item-person map is displayed for Rasch and 1PL models (Figure 7, bottom right). Table of item parameter estimates is completed by $S - X^2$ item fit statistics (Orlando and Thissen, 2000). Estimated latent abilities (factor scores) are also available for download. While fitting of IRT models is mainly implemented using the **mirt** package (Chalmers, 2018), sample R code is provided for work in both **mirt** and **ltm** (Rizopoulos, 2006).

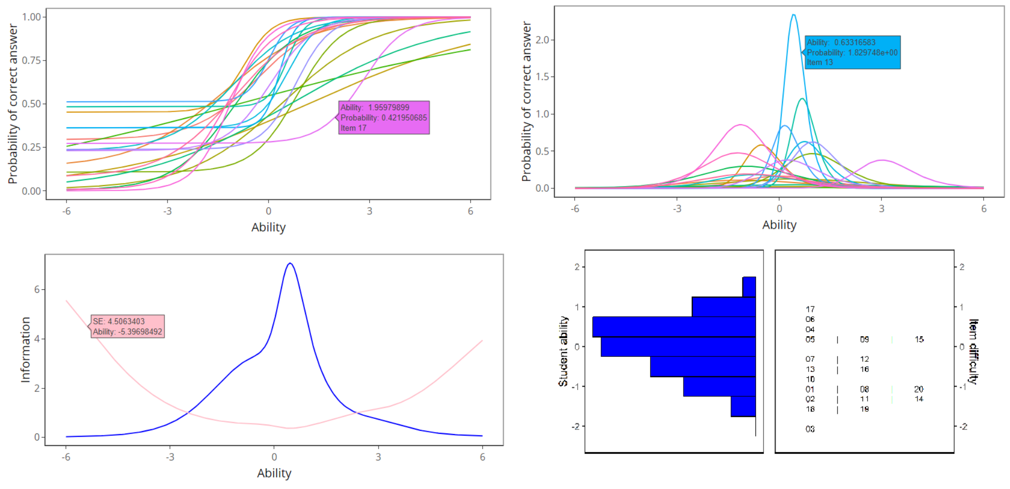


Figure 7: IRT plots for HCI data. From top left: Item characteristic curves, item information curves, test information curve with standard error of measurement, and item-person map.

Finally, section **DIF/Fairness** offers the most used tools for detection of DIF and DDF included in **deltaPlotR** (Magis and Facon, 2014), **difR** (Magis et al., 2010) and the **difNLR** package (Drabinová et al., 2018, see also Drabinová and Martinková, 2018 and Drabinová and Martinková, 2017).

Datasets GMAT and HCI provide valuable teaching examples, further discussed in Martinková et al. (2017). HCI is an example of a situation whereby the two groups differ significantly in their overall ability, yet no item is detected as DIF. Dataset GMAT was simulated to demonstrate that it is hypothetically possible that even though the distributions of the total scores for the two groups are identical, yet, there may be an item present that functions differently for each group (see Figure 8).

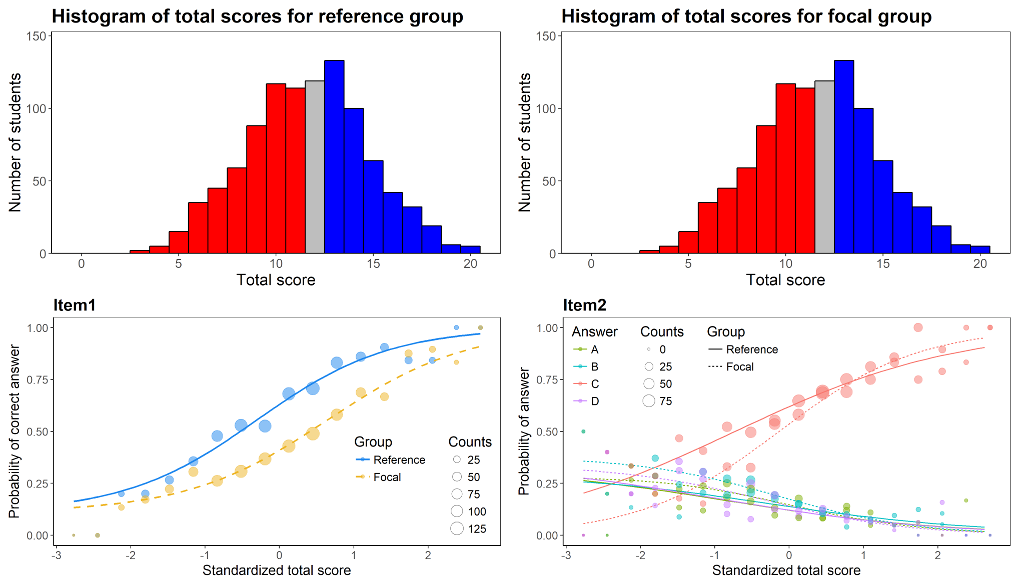


Figure 8: GMAT data simulated to show that hypothetically, two groups may have an identical distribution of total scores, yet there may be a DIF item present in the data.

Teaching with ShinyItemAnalysis

ShinyItemAnalysis is based on examples developed for a course of IRT models and psychometrics offered to graduate students at the University of Washington and at the Charles University. It has also been used at workshops for educators developing admission tests and other tests in various fields.

Besides the presence of a broad range of CTT as well as IRT methods, toy data examples, model equations, parameter estimates, and interactive interpretation of results, selected R code is also available, ready to be copy-pasted and run in R. The shiny application can thus serve as a bridge to users who do not feel secure in the R programming environment by providing examples which can be further modified or adopted to different datasets.

As an important teaching tool, an interactive training section is present, deploying item characteristic and item information curves for IRT models. For dichotomous models (Figure 9, left), the user can specify parameters a, b, c, and d of two toy items and latent ability θ of respondent. ICC is then provided interactively, displaying probability of a correct answer for any θ and highlighting this probability for selected θ. IIC compares the two items in the amount of information they provide about respondents of a given ability level.

For polytomous items (Figure 9, right), analogous interactive plots are available for GRM, (G)PCM as well as for NRM. Step functions are displayed for GRM, and category response functions are available for all three models. In addition, the expected item score is displayed for all the models.

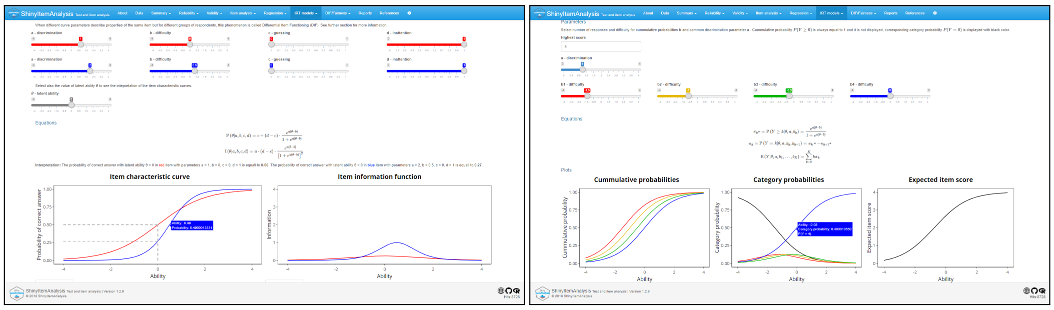

ShinyItemAnalysis
Item characteristic curve
Item information function
Cummulative probabilities
Category probabilities
Expected item score
Ability

Figure 9: Interactive training IRT section.

The training sections also contain interactive exercises where students may check their understanding of the IRT models. They are asked to sketch ICC and IIC functions of items with given parameters, and to answer questions, e. g. regarding probabilities of correct answers and the information these items provide for certain ability levels.

Automatic report generation

To support routine usage of psychometric methods in test development, **ShinyItemAnalysis** offers the possibility to upload your own data for analysis as csv files, and to generate PDF or HTML reports. A sample PDF report and the corresponding csv files used for its generation are provided in Supplemental Materials.

Report generation uses **rmarkdown** templates and **knitr** for compiling (see Figure 10). LaTeX is used for creating PDF reports. The latest version of LaTeX with properly set paths is needed to generate PDF reports locally.

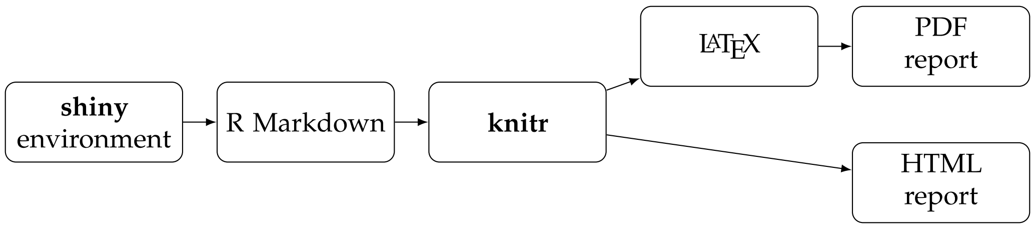
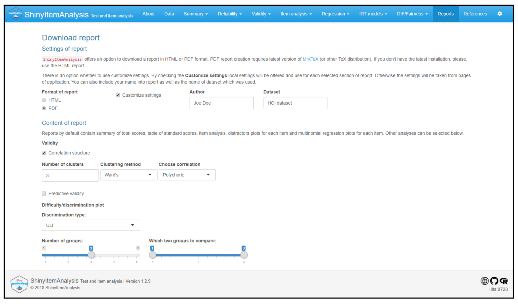


Figure 10: Report generation workflow.

Page with report setting allows user to specify a dataset name, the name of the person who generated the report, to select a method and to customize the settings (see Figure 11). The **Generate report** button at the bottom starts analyses needed for the report to be created. Subsequently, the **Download report** button initializes compiling the text, tables and figures into a PDF or an HTML file.



Download report

Settings of report

ShinyItemAnalysis offers an option to download a report in HTML or PDF format. PDF report creation requires latest version of MiKTeX (or other TeX distribution). If you don't have the latest installation, please, use the HTML report.

There is an option whether to use customize settings. By checking the **Customize settings** local settings will be offered and use for each selected section of report. Otherwise the settings will be taken from pages of application. You can also include your name into report as well as the name of dataset which was used.

Format of report: HTML [✓ PDF]

Customize settings: ✓

Author: Joe Doe

Dataset: HCI dataset

Content of report

Reports by default contain summary of total scores, table of standard scores, item analysis, distractors plots for each item and multinomial regression plots for each item. Other analyses can be selected below.

Validity

Correlation structure: ✓

Number of clusters: 3

Clustering method: Ward's

Choose correlation: Polychoric

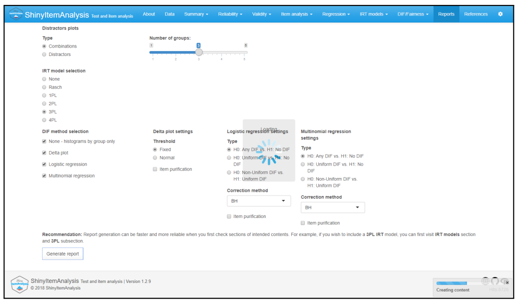
☐ Predictive validity

Difficulty/discrimination plot

Discrimination type: ULI

Number of groups: 3

Which two groups to compare: 1, 3



Distractors plots

Type: [✓ Combinations] Distractors

Number of groups: 3

IRT model selection: None Rasch 1PL 2PL [✓ 3PL] 4PL

DIF method selection: None - histograms by group only ✓; Delta plot ✓; Logistic regression ✓; Multinomial regression ✓

Delta plot settings

Threshold: [✓ Fixed] Normal

☐ Item purification

Logistic regression settings

Type: [✓ H0: Any DIF vs. H1: No DIF] H0: Uniform DIF vs. H1: No DIF; H0: Non-Uniform DIF vs. H1: Uniform DIF

Correction method: BH

☐ Item purification

Multinomial regression settings

Type: [✓ H0: Any DIF vs. H1: No DIF] H0: Uniform DIF vs. H1: No DIF; H0: Non-Uniform DIF vs. H1: Uniform DIF

Correction method: BH

☐ Item purification

Recommendation: Report generation can be faster and more reliable when you first check sections of intended contents. For example, if you wish to include a **3PL IRT** model, you can first visit **IRT models** section and **3PL** subsection.

Figure 11: Report settings for the HCI data analysis.

Sample pages of a PDF report on the HCI dataset are displayed in Figure 12. Reports provide a quick overview of test characteristics and may be a helpful material for test developers, item writers and institutional stakeholders.

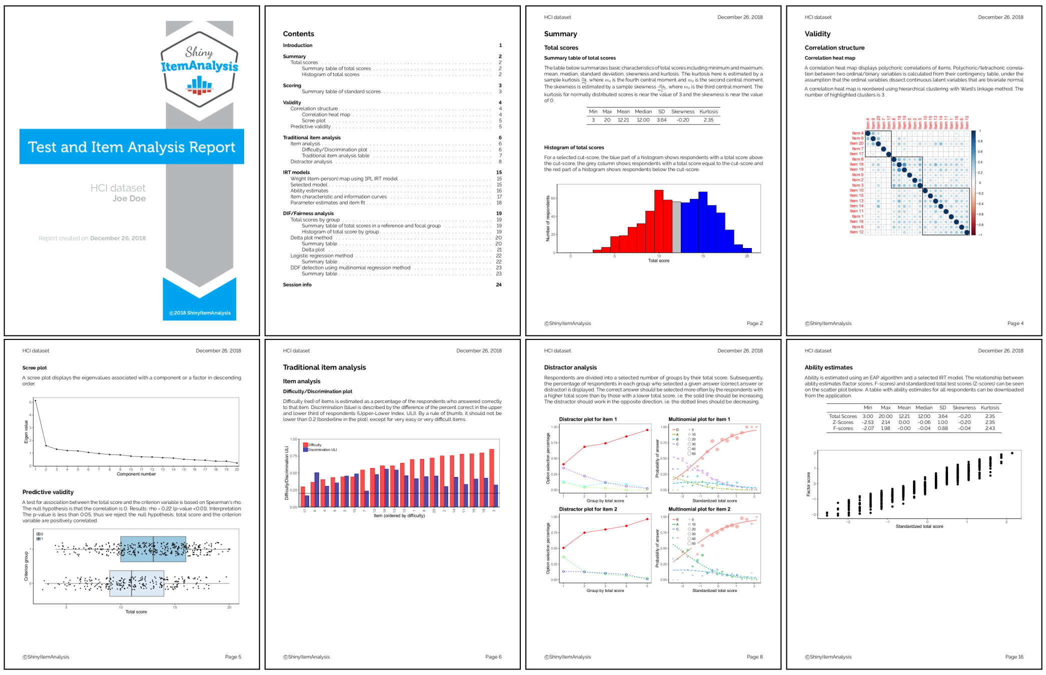


Figure 12: Selected pages of a report on the HCI data.

Discussion and conclusion

ShinyItemAnalysis is an R package (currently version 1.2.9) and an online shiny application for psychometric analysis of educational tests and items. It is suitable for teaching psychometric concepts and it aspires to be an easy-to-use tool for routine analysis of educational tests. For teaching psychometric concepts, a wide range of models and methods are provided, together with interactive plots, exercises, data examples, model equations, parameter estimates, interpretation of results, and selected R code to bring new users to R. For analysis of educational tests by educators, **ShinyItemAnalysis** interactive application allows users to upload and analyze their own data online or locally, and to automatically generate analysis reports in PDF or HTML.

Functionality of the **ShinyItemAnalysis** has been validated on three groups of users. As the first group, two university professors teaching psychometrics and test development provided their written feedback on using the application and suggested edits for wording used for interpretation of results provided by the shiny application. As the second group, over 20 participants of a graduate seminar on "Selected Topics in Psychometrics" at the Charles University in 2017/2018 used **ShinyItemAnalysis** throughout the year in practical exercises. Students prepared their final projects with **ShinyItemAnalysis** applied on their own datasets and provided closer feedback on their experience. The third group consisted of more than 20 university academics from different fields who participated in a short-term course on "Test Development and Validation" in 2018 at the Charles University. During the course, participants used the application on toy data embedded in the package. In addition, an online knowledge test was prepared in Google Docs, answered by participants, and subsequently analyzed in **ShinyItemAnalysis** during the same session. Participants provided their feedback and commented on usability of the package and shiny application. As a result, various features were improved (e. g. data upload format was extended, some cases of missing values are now handled).

Current developments of the **ShinyItemAnalysis** package comprise implementation of wider types of models, especially ordinal and multinomial models and models accounting for effect of the rater and a hierarchical structure. In reliability estimation, further sources of data variability are being implemented to provide estimation of model-based inter-rater reliability (Martinková et al., 2018b). Technical improvements include a more complex data upload or automatic testing of new versions of the application.

We argue that psychometric analysis should be a routine part of test development in order to gather proofs of reliability and validity of the measurement and we have demonstrated how **ShinyItemAnalysis** may enforce this goal. It may also serve as an example for other fields, demonstrating the ability of shiny applications to interactively present theoretical concepts and, when complemented with sample R code, to bring new users to R, or to serve as a bridge to those who have not yet discovered the beauties of R.

Acknowledgments

This work was initiated while P. Martinková was a Fulbright-Masaryk fellow with University of Washington. Research was partly supported by the Czech Science Foundation (grant number GJ15-15856Y). We gratefully thank Jenny McFarland for providing HCI data and David Magis, Hynek Cígler, Hana Ševčíková, Jon Kern and anonymous reviewers for helpful comments to previous versions of this manuscript. We also wish to acknowledge those who contributed to the **ShinyItemAnalysis** package or provided their valuable feedback by e-mail, on GitHub or through an online Google form at http://www.ShinyItemAnalysis.org/feedback.html.

Bibliography

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. URL https://doi.org/10.1109/TAC.1974.1100705. [p508]

J. Allaire, Y. Xie, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, and W. Chang. *rmarkdown: Dynamic Documents for R*, 2018. URL https://CRAN.R-project.org/package=rmarkdown. R package version 1.10. [p506]

American Educational Research Association (AERA), American Psychological Association (APA), National Council on Measurement in Education (NCME). *Standards for Educational and Psychological Testing*. American Educational Research Association, 2014. [p503]

D. Andrich. A rating formulation for ordered response categories. *Psychometrika*, 43(4):561–573, 1978. URL https://doi.org/10.1007/BF02293814. [p505]

W. H. Angoff and S. F. Ford. Item-race interaction on a test of scholastic aptitude. *Journal of Educational Measurement*, 10(2):95–105, 1973. URL https://doi.org/10.1002/j.2333-8504.1971.tb00812.x. [p505]

D. Attali. *shinyjs: Easily Improve the User Experience of Your shiny Apps in Seconds*, 2018. URL https://CRAN.R-project.org/package=shinyjs. R package version 1.0. [p506]

B. Auguie. *gridExtra: Miscellaneous Functions for 'Grid' Graphics*, 2017. URL https://CRAN.R-project.org/package=gridExtra. R package version 2.3. [p506]

E. Bailey. *shinyBS: Twitter Bootstrap Components for shiny*, 2015. URL https://CRAN.R-project.org/package=shinyBS. R package version 0.61. [p506]

R. D. Bock. Estimating item parameters and latent ability when responses are scored in two or more nominal categories. *Psychometrika*, 37(1):29–51, 1972. URL https://doi.org/10.1007/BF02291411. [p505]

R. L. Brennan. *Educational Measurement*. Praeger Publishers, 2006. [p503]

P. Chalmers. *mirt: Multidimensional Item Response Theory*, 2018. URL https://CRAN.R-project.org/package=mirt. R package version 1.29. [p505, 506, 509]

W. Chang and B. Borges Ribeiro. *shinydashboard: Create Dashboards with 'shiny'*, 2018. URL https://CRAN.R-project.org/package=shinydashboard. R package version 0.7.1. [p506]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2018. URL https://CRAN.R-project.org/package=shiny. R package version 1.2.0. [p503, 506]

L. J. Cronbach. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334, 1951. URL https://doi.org/10.1007/BF02310555. [p504]

D. B. Dahl, D. Scott, C. Roosen, A. Magnusson, and J. Swinton. *xtable: Export Tables to LaTeX or HTML*, 2018. URL https://CRAN.R-project.org/package=xtable. R package version 1.8-3. [p506]

A. de Vries and B. D. Ripley. *ggdendro: Create Dendrograms and Tree Diagrams Using 'ggplot2'*, 2016. URL https://CRAN.R-project.org/package=ggdendro. R package version 0.1-20. [p506]

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2018. URL https://CRAN.R-project.org/package=data.table. R package version 1.11.8. [p506]

S. M. Downing and T. M. Haladyna, editors. *Handbook of Test Development*. Lawrence Erlbaum Associates, Inc., 2011. [p503]

A. Drabinová and P. Martinková. Detection of differential item functioning with nonlinear regression: A non-IRT approach accounting for guessing. *Journal of Educational Measurement*, 54(4):498–517, 2017. URL https://doi.org/10.1111/jedm.12158. [p505, 506, 509]

A. Drabinová and P. Martinková. difNLR: Generalized logistic regression models for DIF and DDF detection. *The R Journal*, 2018. Submitted. [p509]

A. Drabinová, P. Martinková, and K. Zvára. *difNLR: DIF and DDF Detection by Non-Linear Regression Models*, 2018. URL https://CRAN.R-project.org/package=difNLR. R package version 1.2.2. [p506, 509]

R. L. Ebel. Procedures for the analysis of classroom tests. *Educational and Psychological Measurement*, 14 (2):352–364, 1954. URL https://doi.org/10.1177/001316445401400215. [p503]

T. D. Fletcher. *psychometric: Applied Psychometric Theory*, 2010. URL https://CRAN.R-project.org/package=psychometric. R package version 2.2. [p506]

C. H. Jackson. Multi-state models for panel data: The msm package for R. *Journal of Statistical Software*, 38(8):1–29, 2011. URL https://doi.org/10.18637/jss.v038.i08. [p506]

M. S. Johnson. Marginal maximum likelihood estimation of item response models in R. *Journal of Statistical Software*, 20(10):1–24, 2007. URL https://doi.org/10.18637/jss.v020.i10. [p505]

L. Komsta and F. Novomestky. *moments: Moments, Cumulants, Skewness, Kurtosis and Related Tests*, 2015. URL https://CRAN.R-project.org/package=moments. R package version 0.14. [p506]

F. M. Lord. *Applications of Item Response Theory to Practical Testing Problems*. Routledge, 1980. [p505]

D. Magis and B. Facon. deltaPlotR: An R package for differential item functioning analysis with Angoff's delta plot. *Journal of Statistical Software, Code Snippets*, 59(1):1–19, 2014. URL https://doi.org/10.18637/jss.v059.c01. [p506, 509]

D. Magis, S. Beland, F. Tuerlinckx, and P. De Boeck. A general framework and an R package for the detection of dichotomous differential item functioning. *Behavior Research Methods*, 42:847–862, 2010. URL https://doi.org/10.3758/BRM.42.3.847. [p503, 506, 509]

N. Mantel and W. Haenszel. Statistical aspects of the analysis of data from retrospective studies of disease. *Journal of the national cancer institute*, 22(4):719–748, 1959. URL https://doi.org/10.1093/jnci/22.4.719. [p505]

P. Martinková, A. Drabinová, and J. Houdek. ShinyItemAnalysis: Analýza přijímacích a jiných znalostních či psychologických testů [ShinyItemAnalysis: Analyzing admission and other educational and psychological tests]. *TESTFÓRUM*, 6(9):16–35, 2017. URL https://doi.org/10.5817/TF2017-9-129. [p503]

P. Martinková, L. Štěpánek, A. Drabinová, J. Houdek, M. Vejražka, and Č. Štuka. Semi-real-time analyses of item characteristics for medical school admission tests. In *Computer Science and Information Systems (FedCSIS), 2017 Federated Conference on*, pages 189–194. IEEE, 2017. URL https://doi.org/10.15439/2017F380. [p503, 505, 506]

P. Martinková, A. Drabinová, O. Leder, and J. Houdek. *ShinyItemAnalysis: Test and Item Analysis via shiny*, 2018a. URL https://CRAN.R-project.org/package=ShinyItemAnalysis. R package version 1.2.9. [p503]

P. Martinková, D. Goldhaber, and E. Erosheva. Disparities in ratings of internal and external applicants: A case for model-based inter-rater reliability. *PloS ONE*, 13(10):e0203002, 2018b. URL https://doi.org/10.1371/journal.pone.0203002. [p512]

P. Martinková, A. Drabinová, Y.-L. Liaw, E. A. Sanders, J. L. McFarland, and R. M. Price. Checking equity: Why differential item functioning analysis should be a routine part of developing conceptual assessments. *CBE-Life Sciences Education*, 16(2):rm2, 2017. URL https://doi.org/10.1187/cbe.16-10-0307. [p505, 506, 509]

G. N. Masters. A Rasch model for partial credit scoring. *Psychometrika*, 47(2):149–174, 1982. URL https://doi.org/10.1007/BF02296272. [p505]

J. L. McFarland, R. M. Price, M. P. Wenderoth, P. Martinková, W. Cliff, J. Michael, H. Modell, and A. Wright. Development and validation of the homeostasis concept inventory. *CBE-Life Sciences Education*, 16(2):ar35, 2017. URL https://doi.org/10.1187/cbe.16-10-0305. [p503, 505, 506, 507]

E. Muraki. A generalized partial credit model: Application of an EM algorithm. *ETS Research Report Series*, 1992(1), 1992. URL https://doi.org/10.1002/j.2333-8504.1992.tb01436.x. [p505]

M. Orlando and D. Thissen. Likelihood-based item fit indices for dichotomous item response theory models. *Applied Psychological Measurement*, 24(1):50–64, 2000. URL https://doi.org/10.1177/2F01466216000241003. [p509]

N. S. Raju. The area between two item characteristic curves. *Psychometrika*, 53(4):495–502, 1988. URL https://doi.org/10.1007/BF02294403. [p505]

N. S. Raju. Determining the significance of estimated signed and unsigned areas between two item response functions. *Applied Psychological Measurement*, 14(2):197–207, 1990. URL https://doi.org/10.1177/014662169001400208. [p505]

G. Rasch. *Studies in Mathematical Psychology: I. Probabilistic Models for Some Intelligence and Attainment Tests*. Nielsen & Lydiche, 1960. [p508]

W. Revelle. *psych: Procedures for Psychological, Psychometric, and Personality Research*, 2018. URL https://CRAN.R-project.org/package=psych. R package version 1.8.10. [p503, 506]

F. Rijmen, F. Tuerlinckx, P. De Boeck, and P. Kuppens. A nonlinear mixed model framework for item response theory. *Psychological methods*, 8(2):185, 2003. URL https://doi.org/10.1037/1082-989X.8.2.185. [p504]

D. Rizopoulos. ltm: An R package for latent variable modelling and item response theory analyses. *Journal of Statistical Software*, 17(5):1–25, 2006. URL https://doi.org/10.18637/jss.v017.i05. [p503, 505, 506, 509]

Y. Rosseel. lavaan: An R package for structural equation modeling. *Journal of Statistical Software*, 48(2): 1–36, 2012. URL https://doi.org/10.18637/jss.v048.i02. [p503]

F. Samejima. Estimation of latent ability using a response pattern of graded scores. *Psychometrika*, 34 (1):1–97, 1969. URL https://doi.org/10.1007%2FBF03372160. [p505]

D. Sarkar and F. Andrews. *latticeExtra: Extra Graphical Utilities Based on Lattice*, 2016. URL https://CRAN.R-project.org/package=latticeExtra. R package version 0.6-28. [p506]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. URL https://doi.org/10.1214/aos/1176344136. [p508]

C. Sievert. *plotly for R*, 2018. URL https://plotly-book.cpsievert.me. [p506]

H. Swaminathan and H. J. Rogers. Detecting differential item functioning using logistic regression procedures. *Journal of Educational measurement*, 27(4):361–370, 1990. URL https://doi.org/10.1111/j.1745-3984.1990.tb00754.x. [p505]

W. J. van der Linden. *Handbook of Item Response Theory*. Chapman and Hall/CRC, 2017. [p505]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, New York, 4th edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. [p506]

T. Wei and V. Simko. *corrplot: Visualization of a Correlation Matrix*, 2017. URL https://CRAN.R-project.org/package=corrplot. R package version 0.84. [p506]

H. Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. URL https://doi.org/10.18637/jss.v021.i12. [p506]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2016. URL http://ggplot2.org. [p506]

H. Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2018. URL https://CRAN.R-project.org/package=stringr. R package version 1.3.1. [p506]

C. O. Wilke. *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'*, 2018. URL https://CRAN.R-project.org/package=cowplot. R package version 0.9.3. [p506]

J. T. Willse. *CTT: Classical Test Theory Functions*, 2018. URL https://CRAN.R-project.org/package=CTT. R package version 2.3.3. [p506]

Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2018. URL https://CRAN.R-project.org/package=knitr. R package version 1.20. [p506]

Y. Xie, J. Cheng, and X. Tan. *DT: A Wrapper of the JavaScript Library 'DataTables'*, 2018. URL https://CRAN.R-project.org/package=DT. R package version 0.5. [p506]

R. E. Zinbarg, W. Revelle, I. Yovel, and W. Li. Cronbach's α, revelle's β, and mcdonald's ω_h: Their relations with each other and two alternative conceptualizations of reliability. *Psychometrika*, 70(1): 123–133, 2005. URL https://doi.org/10.1007/s11336-003-0974-7. [p504]

Patrícia Martinková
Department of Statistical Modelling, Institute of Computer Science, Czech Academy of Sciences
Pod Vodárenskou věží 271/2, Prague, 182 07
and
Institute for Research and Development of Education, Faculty of Education, Charles University
Myslíkova 7, Prague, 110 00
Czech Republic
ORCiD: 0000-0003-4754-8543
martinkova@cs.cas.cz

Adéla Drabinová
Department of Statistical Modelling, Institute of Computer Science, Czech Academy of Sciences
Pod Vodárenskou věží 271/2, Prague, 182 07
and
Department of Probability and Mathematical Statistics, Faculty of Mathematics and Physics, Charles University
Sokolovská 83, Prague, 186 75
Czech Republic
ORCiD: 0000-0002-9112-1208
drabinova@cs.cas.cz

RcppMsgPack: MessagePack Headers and Interface Functions for R

by Travers Ching and Dirk Eddelbuettel

Abstract MessagePack, or **MsgPack** for short, or when referring to the implementation, is an efficient binary serialization format for exchanging data between different programming languages. The **RcppMsgPack** package provides *R* with both the MessagePack *C++* header files, and the ability to access, create and alter MessagePack objects directly from *R*. The main driver functions of the R interface are two functions `msgpack_pack` and `msgpack_unpack`. The function `msgpack_pack` serializes *R* objects to a raw MessagePack message. The function `msgpack_unpack` de-serializes MessagePack messages back into *R* objects. Several helper functions are available to aid in processing and formatting data including `msgpack_simplify`, `msgpack_format` and `msgpack_map`.

Introduction

MessagePack (or **MsgPack** for short, or when referring to the actual implementation) is a binary serialization format made for exchanging data between different programming languages (Furuhashi, 2018). Unlike other related formats such as JSON, **MsgPack** is a binary format—which makes it more efficient in terms of (disk or memory) space, transfer speeds (which is increasingly important for large data sets across networks) and potentially also precision (as textual representation rarely goes to the length of binary precision). As shown on the project homepage at `https://msgpack.org`, several major projects including *Redis*, *Pinterest*, *Fluentd* and *Treasure Data* utilize **MsgPack** to transfer data or to represent internal data structures (Furuhashi, 2018). Other binary serialization formats similar to **MsgPack** include BSON (MongoDB, 2018) and ProtoBuf (Google, 2018) which have their own advantages and disadvantages, such as serialization speed, memory usage, compression and requirement of descriptive schemas (Hamida et al., 2015; Dawborn and Curran, 2014). R support for these formats is available via the packages **mongolite** (Ooms, 2014) and **RProtoBuf** (Eddelbuettel et al., 2016); Redis is also implemented in R through the **RcppRedis** package (Eddelbuettel, 2018). **RcppMsgPack** (Ching et al., 2018) brings support for the **MsgPack** specification to R.

The **MsgPack** specification describes a number of common data type: Booleans, Integers, Floats, Strings, Binary data, Arrays, Maps, and user-defined extension types, and has been implemented in most major programming languages. **RcppMsgPack** aims to provide an efficient, and easy to use implementation by relying on the official C++ **MsgPack** code and the **Rcpp** package (Eddelbuettel, 2013; Eddelbuettel et al., 2018). The package provides users with the **MsgPack** header files which can be used to more directly integrate **MsgPack** into R projects through C++ code. It also provides the ability to serialize and de-serialize data directly to and from R (*e.g.*, through pipes, file handlers, sockets or binary object files). These functionalities can be used to efficiently transfer data between various programming languages and between separate R instances.

In this manuscript, we describe the main interface functions used to serialize and deserialize **MsgPack** messages, and the conversion between R data types and **MsgPack** data types. We describe helper functions contained in **RcppMsgPack** and describe several use cases and examples of how **RcppMsgPack** can be used in practice, benchmarking several common approaches for transferring data between processes.

Interface functions

The functions `msgpack_pack` and `msgpack_unpack` allow serialization and de-serialization of R objects respectively (Figure 1). Here, `msgpack_pack` takes in any number of R objects and generates a single serialized message in the form of a raw vector. Conversely, `msgpack_unpack` takes a serialized message as input, and returns the R object(s) contained in the message. Moreover, `msgpack_format` is a helper function to properly format R objects for input, and `msgpack_simplify` is a helper function to simplify output from a **MsgPack** conversion. One of the main goals of **MsgPack** is the transfer of data across processes and/or hosts. Therefore, we also define two helper functions `msgpack_write` and `msgpack_read` which facilitate writing and reading of **MsgPack** objects to files, pipes or any connection object.

The data types in **MsgPack** do not directly map on to R data types, more so than other languages. For example, basic R "atomic" types such as integers and strings are inherently vectorized, which is not true in C++, Python or most other languages. *I.e.*, in R there is no distinction between a single integer and a vector of integers of length 1. R also has multiple non-value types, such as `NULL` or `NA`

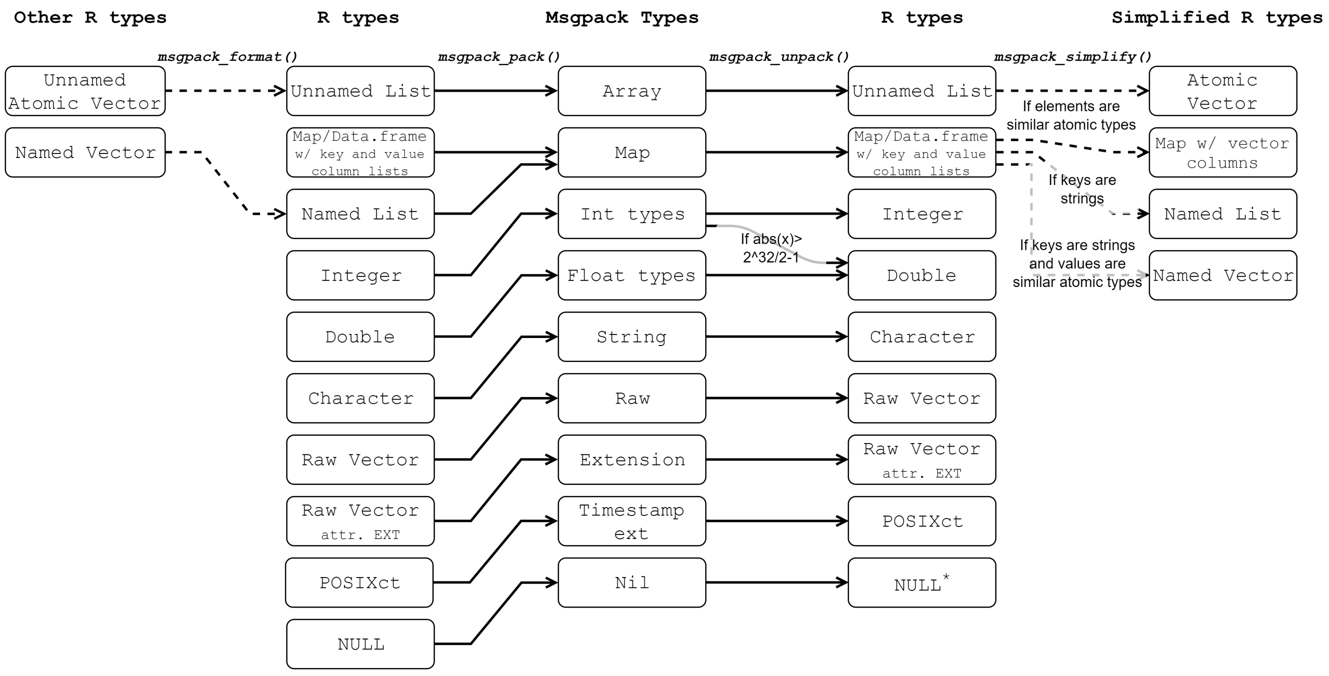


Figure 1: A flowchart of the conversion of R objects to **MsgPack** objects and vice versa.

for integer, string, numeric, etc. Because of these complexities, the conversion processes using these interface functions are described in detail below.

R integers are converted into **MsgPack** integers, which are automatically reduced in size, depending on the value of the integer. **MsgPack** integers are converted back into R integers. Because R does not natively support 64 bit integers, whereas **MsgPack** supports integers up to 64 unsigned bits in value, **MsgPack** integers exceeding signed 32 bits supported by R are coerced to R numeric values, with potential loss of precision. The integer NA value in R is represented by its bit value in C++ (0x80000000), and requires no special treatment.

R numeric (*i.e.*, doubles) variables conform to IEEE 754 double-precision standards (IEEE Standards Committee, 2008), and also require no special treatment. The numeric NA value is a special case of NaN values, and is serialized by its bit representation.

R strings (*i.e.*, objects of class character) are converted to **MsgPack** strings. Because C++ and **MsgPack** do not have missing values for strings, NA characters are converted into **MsgPack** Nil (similar to NULL in R).

R logical values are converted into **MsgPack** bool. Again, because NA logical values do not exist in C++ or **MsgPack**, NA logical values are converted into **MsgPack** Nil.

R raw vectors are converted into **MsgPack** bin. Raw vectors with the "EXT" integer attribute are converted into **MsgPack** extension types. The EXT attribute should be a positive integer, as negative values are reserved for official extensions.

Currently, the **MsgPack** specifications includes one official extension type: timestamps. Timestamps are a **MsgPack** extension type with extension value -1 and can be converted to and from R POSIXct objects using msgpack_timestamp_decode and msgpack_timestamp_encode respectively. **MsgPack** timestamps can encode nanosecond precision. R POSIXct objects rely on numeric, and therefore conversion may have some loss of precision (unless a package such as **nanotime** (Eddelbuettel and Silvestri, 2018) is used, which is left as a future extension).

MsgPack specifications define two container objects: arrays and maps. **MsgPack** arrays are a sequential container object. The length of the array is defined in its message header. Arrays can contain any other **MsgPack** types, including other arrays or maps.

MsgPack arrays are naturally analogous to R unnamed list objects. However, because lists have a large memory footprint, R atomic vectors (with length of 0 or greater than or equal to 2) are also allowed as input for serialization to arrays.

MsgPack maps are an ordered sequence of key and value pairs, where each key and value can be any **MsgPack** object. There is no requirement for unique keys. Maps do not have an analogous data type in R . Therefore, maps are implemented by creating an object of class map, which is also a data.frame with key and value columns. As input to serialization, these columns can also be lists, and can therefore contain any other R object, and not only a single type. The function msgpack_map is a simple helper function that takes two lists and returns a map which can be serialized into a **MsgPack** object with msgpack_pack.

In order to support as much generality as possible in serialization and deserialization, the use of lists to represent arrays and maps is necessary. However, it is often the case in R that one would want to deal with large vectors or matrices of a single type without the computational and memory overhead of lists. Two approaches are given to deal with this type of scenario. msgpack_simplify can be used after a call to msgpack_unpack to recursively simplify lists to vectors when only a single type is included within a list. (For lists of characters or logicals, this may also include NULLs.) Secondly, msgpack_unpack can be called with the simplify=TRUE parameter, which performs the same task as msgpack_simplify within C++, and is therefore much faster. The second approach can drastically improve speed and memory usage compared to the first approach.

Using MsgPack C++ headers through RcppMsgPack and Rcpp

Complex objects or data structures, such as trees, often do not fit into R data types because a tree data structure does not map nicely to an R vector, data.frame, matrix, etc. Storing such a complex object as a **MsgPack** message will be more performant in terms of serialization speed and memory usage.

The example below demonstrates how **MsgPack** headers can be integrated into a standard **Rcpp** workflow. In this example, a prefix tree is created for nucleotide sequences, and is serialized through **MsgPack** to create a persistant tree object in the form of a raw vector in R. The stored tree can be saved to disk, unpacked within R directly using msgpack_unpack or it can be reconstructed into the prefix tree within C++ using the **MsgPack** C++ interface. The code below defines a structure for storing the Prefix tree data and a function for constructing the tree using sequence data input from R and saving it as a **MsgPack** object:

```
struct Node {
  std::shared_ptr<Node> parent;
  std::set<int> sequence_idx;
  std::map< std::string, std::shared_ptr<Node> > children;
};

struct std::shared_ptr<Node>
NewNode(std::shared_ptr<Node> parent,
        std::string name,
        std::set<int> sequence_idx) {
  std::shared_ptr<Node> node = std::shared_ptr<Node>(new Node);
  node->sequence_idx = sequence_idx;
  if (parent) {
    parent->children.insert(std::pair<std::string,
                                      std::shared_ptr<Node> >(name, node));
  }
  return node;
}

void packTree(std::shared_ptr<Node> node,
              msgpack::packer<std::stringstream>& pkr) {
  pkr.pack_array(2);
  std::set<int> sequence_idx = node->sequence_idx;
  std::vector<int> vs(sequence_idx.begin(), sequence_idx.end());
  pkr.pack(vs);
  std::map< std::string, std::shared_ptr<Node> > children = node->children;
  pkr.pack_map(children.size());
  for (auto const& x : children) {
    pkr.pack(x.first);
    packTree(x.second, pkr);
  }
}

Rcpp::RawVector create_prefix_tree(std::vector<std::string> clone_sequences) {
  std::shared_ptr<Node> root;
  root = NewNode(std::shared_ptr<Node>(nullptr), "^", {});
  for(int i=0; i<clone_sequences.size(); i++) {
    std::shared_ptr<Node> current_node = root;
    for(int j=0; j < clone_sequences[i].size(); j++) {
      std::string nuc = clone_sequences[i].substr(j,1);
      if(current_node->children.count(nuc) == 1) {
        current_node = current_node->children[nuc];
        if(j == clone_sequences[i].size() - 1) {
          current_node->sequence_idx.insert(i);
        }
      } else {
        if(j == clone_sequences[i].size() - 1) {
          current_node = NewNode(current_node, nuc, {i});
        } else {
          current_node = NewNode(current_node, nuc, {});
        }
      }
    }
  }
  std::stringstream buffer;
  msgpack::packer<std::stringstream> pk(&buffer);
  packTree(root, pk);
  std::string bufstr = buffer.str();
  Rcpp::RawVector rawbuffer(bufstr.begin(), bufstr.end());
  return rawbuffer;
}
```

The create_prefix_tree function is an C++ function that returns a raw vector, which is a serialization of the prefix tree. From R, the prefix tree can be initialized and serialized through calling the **Rcpp** function.

```
tree <- create_prefix_tree(c("AGCT", "AGCC", "AGC", "ATG", "GACC", "GTCT"))
```

Because the resulting message is a standard **MsgPack** object, the tree can be unpacked directly in R using msgpack_unpack. Additionally, the following code reconstructs the prefix tree from the **MsgPack** message from within C++ using the C++ interface:

```
void print_node(msgpack::object & node_obj, std::string name) {
    std::vector<msgpack::object> node_obj_array;
    node_obj.convert(node_obj_array);
    msgpack::object_kv* p = node_obj_array[1].via.map.ptr;
    msgpack::object_kv* pend = node_obj_array[1].via.map.ptr +
                               node_obj_array[1].via.map.size;
    std::cout << name;
    if(p != pend) {
        std::cout << "(";
        for (; p < pend; ++p) {
            std::string name_child = p->key.as<std::string>();
            msgpack::object node_child_obj = p->val.as<msgpack::object>();
            print_node(node_child_obj, name_child);
            if(p < (pend-1)) std::cout << ",";
        }
        std::cout << ")";
    }
}
void print_newick_tree(std::vector<unsigned char> packed_tree) {
    std::string message(packed_tree.begin(), packed_tree.end());
    msgpack::object_handle oh;
    msgpack::unpack(oh, message.data(), message.size());
    msgpack::object obj = oh.get();
    print_node(obj, "Root");
}
```

Called from within R, the tree is printed to the console:

```
tree_nested_list <- msgpack_unpack(tree, simplify=F)
print_newick_tree(tree)

[1] Root(A(G(C(C,G,T),G)),G(A(C(C)),T(C(A,T))))
```

Writing MsgPack objects to disk and reading data from the internet

Following is an example of how one can create a **MsgPack** serialized message as a binary file, and read it from the internet into R. The first step would be to read in the data to R as normal, and serialize the data using the `msgpack_pack` function.

```
dat <- read.csv(paste0("https://raw.githubusercontent.com/vincentarelbundock/",
                       "Rdatasets/master/csv/mosaicData/Birthdays.csv"))
mp <- with(dat, msgpack_pack(X, state, year, month, day, date, wday, births))
```

The `msgpack_pack` function returns a raw vector, which can be written to disk using the `writeBin` function or the helper function `msgpack_write`.

```
msgpack_write(mp, file="birthdays_msgpack.mp")
```

A **MsgPack** object can be read directly from the internet, e.g., using the `GET` function from the **httr** package (Wickham, 2018). Subsequently, the `msgpack_unpack` function can be used to unpack the object to its original values.

```
md <- GET("http://travers.im/birthdays_msgpack.mp")
mp <- md$content
mu <- msgpack_unpack(mp)
```

Transferring large datasets from Python to R and back

To evaluate the performance of **MsgPack** serialization, we benchmarked the transfer of the MNIST (LeCun et al., 2010) and CIFAR-10 (Krizhevsky, 2009) datasets to and from Python and R. We compared this approach to writing and reading in CSV format, and to writing and reading using **feather** (Wickham et al., 2016), a cross-platform library and specification for efficiently handling tabular data. (The **feather** package is no longer actively developed, and will be superseded by Apache Arrow (Apache Arrow Developers, 2018), a more general cross-language development platform for working with in-memory data in a standardized language-independent columnar memory format. However, Arrow is not yet available for R.) In Python, serialization of the data was performed using the **msgpack** package and written to disk in binary format:

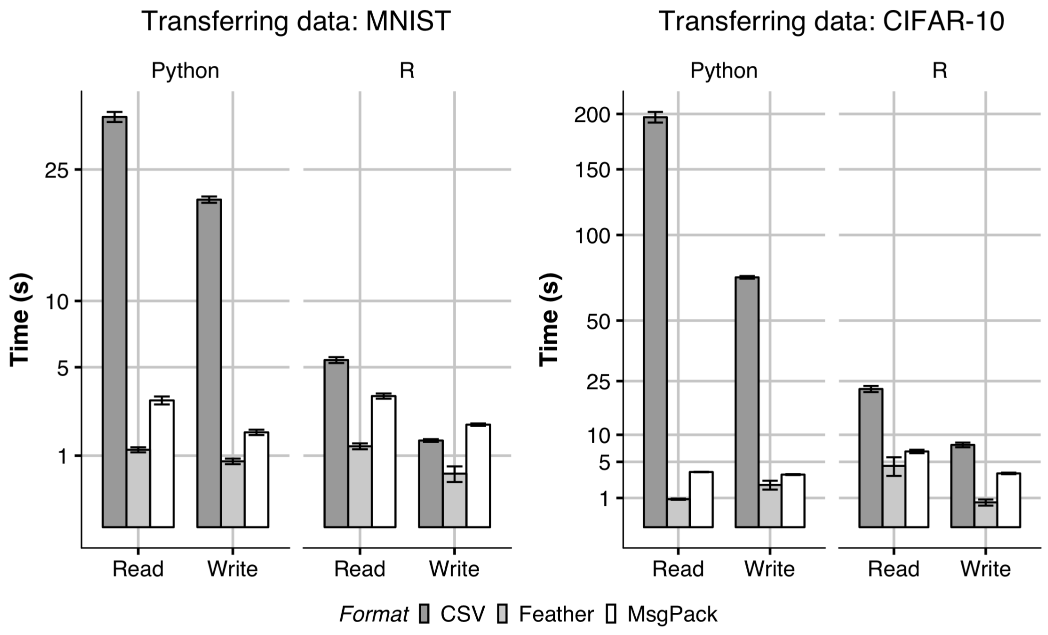


Figure 2: Transferring the MNIST dataset (left) and the CIFAR-10 dataset (right) to and from R and Python using either **MsgPack**, **feather** or CSV format.

```
xb = msgpack.packb(x)
with open("/tmp/dataset.mp", "wb") as file:
    file.write(xb)
```

Conversely, the unpackb function was used when benchmarking the transfer time from R to Python:

```
with open("/tmp/dataset.mp", "rb") as file:
    file.write(xb)
x = msgpack.unpackb(file.read())
```

In R, we used the readBin function followed by msgpack_unpack function to deserialize the dataset. Alternatively, one could use the helper function msgpack_read.

```
xb <- readBin(con = "/tmp/dataset.mp", "raw",
              n=file.info("/tmp/dataset.mp")$size)
x <- msgpack_unpack(xb, simplify=T)
```

Subsequently, the data can be re-serialized and written to disk using the msgpack_pack function followed by writeBin:

```
xb <- msgpack_pack(x)
writeBin(xb,"/tmp/dataset.mp", useBytes=T)
```

For CSV format, we used the fread and fwrite functions in the **data.table** package (Dowle et al., 2018) to write and read CSV tables in R. In Python, we used the package **numpy** savetxt and loadtxt functions (van der Walt et al., 2011). For the **feather** format, we used the write_feather and read_feather functions within the **feather** package in R and Python. All approaches were benchmarked after clearing the system cache and replicated 5 times (Figure 2).

Serialization and de-serialization of objects from **MsgPack** objects was similar between R and Python, with a slight speed advantage going to Python. Reading **MsgPack** objects was generally considerably faster than reading CSV format in both R and Python. Comparing **MsgPack** to **feather**, **feather** was generally faster. As **feather** is designed for columnar binary data, it does not use a header for every element and therefore has an advantage over the (more general) **MsgPack** in this context.

One surprising result is in writing of CSVs in R. Using **data.table**, writing the MNIST dataset was faster than **MsgPack**, although not quite as fast as **feather**. The MNIST dataset is formatted as a single floating point value in per pixel by **TensorFlow** (Abadi et al., 2016). When writing floating point data as CSV, both **numpy** and the **data.table** package truncate floating point numbers by default, which causes loss of precision.

MsgPack serialization memory usage

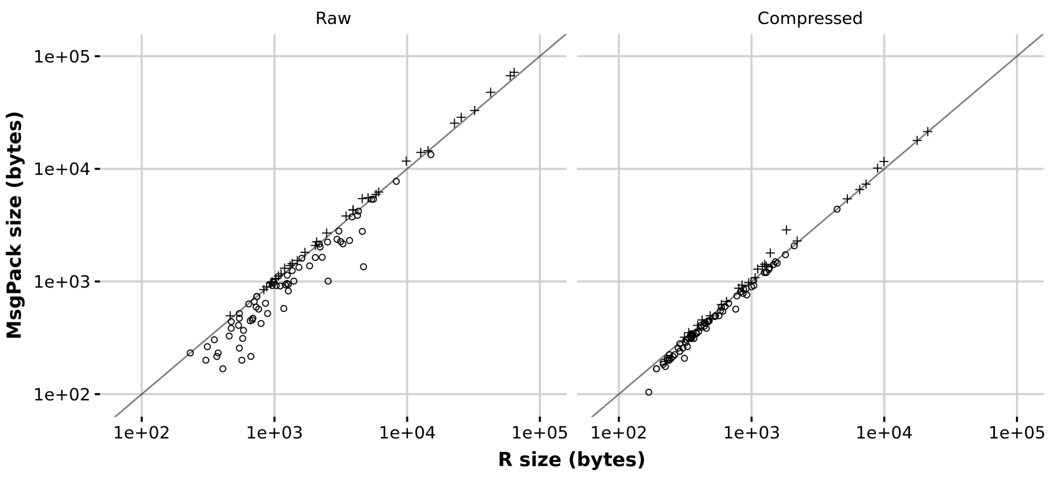


Figure 3: Serialization memory usage: **MsgPack** vs. R. Left: raw serialization size for datasets in **datasets**. Right: Gzip-compressed serialization size.

To compare the memory usage of **MsgPack** to native R serialization, we serialized the datasets found in the datasets R package (Figure 3). For R serialization, this was done by calling the serialize function for native R serialization or calling the msgpack_pack function for **MsgPack** serialization. For compression, the memCompress function was called on the results. Some datasets contained formulas, or other R-specific attributes that can't be stored in **MsgPack**; these attributes were removed prior to serialization and compression.

Memory usage was generally very similar between **MsgPack** and native R. Although there were some differences in raw serialization, the difference seems to be less apparent after compression. For some uncompressed serializations, **MsgPack** significantly outperformed R in memory usage. This is attributable to efficient integer storage: **MsgPack** stores variable size integers depending on the integer magnitude, and can be as low as 8 bits. Native R serialization uses the modified External Data Representation (XDR) standard (Srinivasan, 1995), which uses a fixed 32 bits for integers. Furthermore, attributes of each dataset are stored as lists internally, which can increase the relative overhead for small datasets, as R lists are not memory-efficient objects.

On the other hand, every **MsgPack** element has a short header of several bits, which increases its memory overhead, particularly for large vectors. This overhead is apparent for larger datasets, where R typically is more memory efficient.

Serialization of large lists

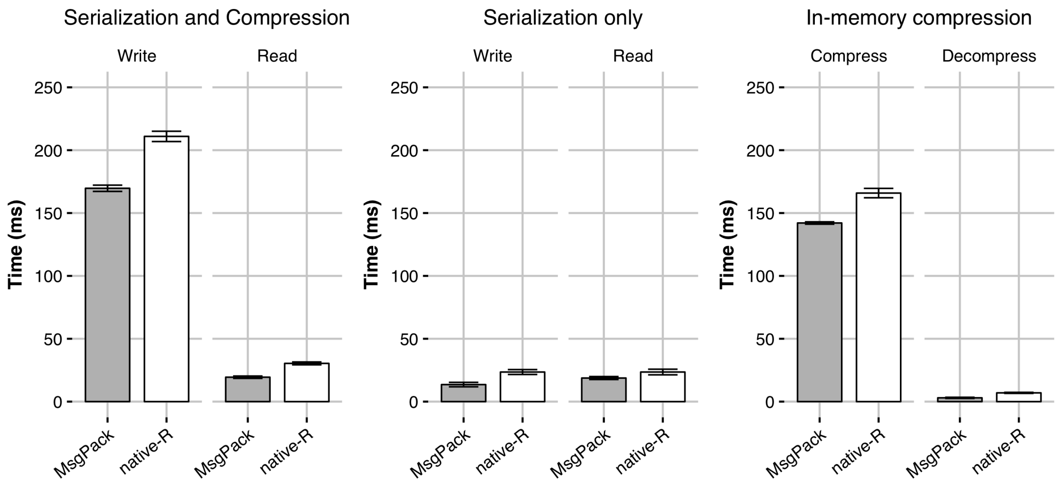


Figure 4: Serialization and compression time of a large list of DNA sequences.

As mentioned, one situation where **MsgPack** serialization is much more efficient than R serialization is how it handles list objects. Because R lists have a large memory overhead, both compression times and writing/reading to disk are faster using the **MsgPack** format. To show this, we generated a contrived list of DNA sequences as follows:

```
x <- replicate(10000, {
    replicate(sample(5,1), {
        paste(sample(c("G","C","A","T"), size=sample(50,1), replace=T), collapse="")
    })
})
```

If the data structure of the object is known, using the C++ headers directly can speed up serialization and de-serialization by avoiding the logic involved in serializing generic data structures, although the speed-up is not large. Compression and writing to disk can also be performed directly within C++, for example, by using the zlib C++ library (Gailly and Adler, 2018) to perform standard DEFLATE compression. The example below illustrates **MsgPack** serialization and subsequent compression using the **zlib** library:

```
void list_pack_gzip(List x, std::string file) {
  std::stringstream buffer;
  msgpack::packer<std::stringstream> pk(&buffer);
  pk.pack_array(x.size());
  for(int i=0; i<x.size(); i++) {
    CharacterVector xi = x[i];
    if(xi.size() == 1) {
      pk.pack(Rcpp::as<std::string>(xi[0]));
    } else {
      pk.pack_array(xi.size());
      for(int j=0; j<xi.size(); j++) {
        pk.pack(Rcpp::as<std::string>(xi[j]));
      }
    }
  }
  std::string bufstr = buffer.str();
  gzFile fi = gzopen(file.c_str(),"wb");
  gzwrite(fi, bufstr.c_str(), bufstr.size());
  gzclose(fi);
}
```

Conversely, uncompression and deserialization would need to be performed to recover the original R object. The following C++ function illustrates how this could be done:

```
  Rcpp::List list_unpack_gzip(std::string file) {
    gzFile fi = gzopen(file.c_str(),"rb");
    char buf[8192];
    std::vector<char> dat;
    uint b = gzread(fi, buf, 8192);
    while (b) {
      dat.insert(dat.end(), buf, buf + b);
      b = gzread(fi, buf, 8192);
    }
    gzclose(fi);
    std::string message = std::string(dat.data(), dat.size());
    msgpack::object_handle oh;
    msgpack::unpack(oh, message.data(), message.size());
    msgpack::object obj = oh.get();
    std::vector<msgpack::object> obj_vector;
    obj.convert(obj_vector);
    Rcpp::List L = Rcpp::List(obj_vector.size());
    std::vector< std::string > temp_str_vec;
    std::string temp_str;
    for (int i=0; i< obj_vector.size(); i++) {
      if (obj_vector[i].type == msgpack::type::ARRAY) {
        obj_vector[i].convert(temp_str_vec);
        L[i] = Rcpp::wrap(temp_str_vec);
      } else {
        obj_vector[i].convert(temp_str);
        L[i] = Rcpp::wrap(temp_str);
      }
    }
    return(L);
  }
```

To illustrate the potential benefit of this approach, we compared **MsgPack** serialization and **zlib** compression to standard R serialization using the `saveRDS` function (Figure 4). Using **MsgPack**, serialization is slightly faster for the example above. The timing of these two approaches can be broken down by process: both serialization and compression to disk (and vice versa) are faster using the **MsgPack** approach described above for certain types of data structures, such as lists. However, for most other types of data structures, native R serialization is usually faster. Most of the time-saving comes from the fact that the serialized **MsgPack** message is considerably smaller, leading to faster compresion time and/or faster reading and writing to disk.

Summary

The examples given above show how working with the R interface functions the C++ headers in **RcppMsgPack** can be useful in transferring data and integrated within data analysis workflows. The package allows fast and flexible serialization of generic data structures in R, and equivalent de-serialization of objects from other langauages. The **RcppMsgPack** package is hosted on CRAN, and can be installed via the standard `install.packages("RcppMsgPack")` command.

Bibliography

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and others. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. [p521]

Apache Arrow Developers. Apache Arrow: A cross-language development platform for in-memory data. https://arrow.apache.org/, 2018. [p520]

T. Ching, D. Eddelbuettel, the authors, and ontributors of MsgPack. *RcppMsgPack: 'MsgPack' C++ Header Files and Interface Functions for R*, 2018. URL https://cran.r-project.org/package=RcppMsgPack. R package version 0.2.3. [p516]

T. Dawborn and J. R. Curran. Docrep: A lightweight and efficient document representation framework. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 762–771, 2014. [p516]

M. Dowle, A. Srinivasan, J. Gorecki, M. Chirico, P. Stetsenko, T. Short, S. Lianoglou, E. Antonyan, M. Bonsch, and H. Parsonage. *data.table: Extension of 'data.frame'*, 2018. URL https://cran.r-project.org/package=data.table. R package version 1.11.8. [p521]

D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer-Verlag, New York, 2013. ISBN 978-1-4614-6867-7. [p516]

D. Eddelbuettel. *RcppRedis: Rcpp Bindings for Redis Using the Hiredis Library*, 2018. URL https://cran.r-project.org/package=RcppRedis. R package version 0.1.9. [p516]

D. Eddelbuettel and L. Silvestri. *Nanotime: Nanosecond-Resolution Time for R*, 2018. URL https://cran.r-project.org/package=nanotime. R package version 0.2.3. [p517]

D. Eddelbuettel, M. Stokely, and J. Ooms. RProtoBuf: Efficient cross-language data serialization in R. *Journal of Statistical Software*, 71(2):1–24, 2016. URL https://doi.org/10.18637/jss.v071.i02. [p516]

D. Eddelbuettel, R. François, J. Allaire, K. Ushey, Q. Kou, N. Russel, J. Chambers, and D. Bates. *Rcpp: Seamless R and C++ Integration*, 2018. URL https://cran.r-project.org/package=Rcpp. R package version 1.0.0. [p516]

S. Furuhashi. MessagePack. https://msgpack.org//, 2018. [p516]

J.-L. Gailly and M. Adler. Zlib compression library, 2018. URL https://zlib.net/. [p522]

Google. Protocol Buffers. http://code.google.com/apis/protocolbuffers/, 2018. [p516]

S. T.-B. Hamida, E. B. Hamida, and B. Ahmed. A new mhealth communication framework for use in wearable wbans and mobile technologies. *Sensors*, 15(2):3379–3408, 2015. [p516]

IEEE Standards Committee. 754-2008 IEEE standard for floating-point arithmetic. *IEEE Computer Society Std*, 2008, 2008. [p517]

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. [p520]

Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010. [p520]

MongoDB. Bson (binary json) serialization, 2018. [p516]

J. Ooms. The jsonlite package: A practical and consistent mapping between json data and r objects. *arXiv:1403.2805 [stat.CO]*, 2014. URL http://arxiv.org/abs/1403.2805. [p516]

R. Srinivasan. *XDR: External Data Representation Standard. No. RFC 1832*, 1995. URL http://www.rfc-editor.org/rfc/pdfrfc/rfc1832.txt.pdf. [p522]

S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. [p521]

H. Wickham. *Httr: Tools for Working with URLs and HTTP*, 2018. URL https://cran.r-project.org/package=httr. R package version 1.4.0. [p520]

H. Wickham, RStudio, Feather developers, Google, and LevelDB Authors. *Feather: R Bindings to the Feather API*, 2016. URL `https://cran.r-project.org/package=feather`. R package version 0.3.1. [p520]

Travers Ching
Unaffiliated
ORCiD: 0000-0002-5577-3516
`traversc@gmail.com`

Dirk Eddelbuettel
Department of Statistics
University of Illinois at Urbana-Champaign
725 S Wright St
Champaign, IL 61820
ORCiD: 0000-0001-6419-907X
`dirk@eddelbuettel.com`

BNSP: an R Package for Fitting Bayesian Semiparametric Regression Models and Variable Selection

by Georgios Papageorgiou

Abstract The R package **BNSP** provides a unified framework for semiparametric location-scale regression and stochastic search variable selection. The statistical methodology that the package is built upon utilizes basis function expansions to represent semiparametric covariate effects in the mean and variance functions, and spike-slab priors to perform selection and regularization of the estimated effects. In addition to the main function that performs posterior sampling, the package includes functions for assessing convergence of the sampler, summarizing model fits, visualizing covariate effects and obtaining predictions for new responses or their means given feature/covariate vectors.

Introduction

There are many approaches to non- and semi-parametric modeling. From a Bayesian perspective, Müller and Mitra (2013) provide a review that covers methods for density estimation, modeling of random effects distributions in mixed effects models, clustering, and modeling of unknown functions in regression models.

Our interest is on Bayesian methods for modeling unknown functions in regression models. In particular, we are interested in modeling both the mean and variance functions non-parametrically, as general functions of the covariates. There are multiple reasons why allowing the variance function to be a general function of the covariates may be important (Chan et al., 2006). Firstly, it can result in more realistic prediction intervals than those obtained by assuming constant error variance, or as Müller and Mitra (2013) put it, it can result in more honest representation of uncertainties. Secondly, it allows the practitioner to examine and understand which covariates drive the variance. Thirdly, it results in more efficient estimation of the mean function. Lastly, it produces more accurate standard errors of unknown parameters.

In the R (R Core Team, 2016) package **BNSP** (Papageorgiou, 2018) we implemented Bayesian regression models with Gaussian errors and with mean and log-variance functions that can be modeled as general functions of the covariates. Covariate effects may enter the mean and log-variance functions parametrically or non-parametrically, with the nonparametric effects represented as linear combinations of basis functions. The strategy that we follow in representing unknown functions is to utilize a large number of basis functions. This allows for flexible estimation and for capturing true effects that are locally adaptive. Potential problems associated with large numbers of basis functions, such as over-fitting, are avoided in our implementation, and efficient estimation is achieved, by utilizing spike-slab priors for variable selection. A review of variable selection methods is provided by O'Hara and Sillanpää (2009).

The methods described here belong to the general class of models known as generalized additive models for location, scale, and shape (GAMLSS) (Rigby and Stasinopoulos, 2005; Stasinopoulos and Rigby, 2007) or the Bayesian analogue termed as BAMLSS (Umlauf et al., 2018) and implemented in package **bamlss** (Umlauf et al., 2017). However, due to the nature of the spike-and-slab priors that we have implemented, in addition to flexible modeling of the mean and variance functions, the methods described here can also be utilized for selecting promising subsets of predictor variables in multiple regression models. The implemented methods fall in the general class of methods known as stochastic search variable selection (SSVS). SSVS has received considerable attention in the Bayesian literature and its applications range from investigating factors that affect individual's happiness (George and McCulloch, 1993), to constructing financial indexes (George and McCulloch, 1997), and to gene mapping (O'Hara and Sillanpää, 2009). These methods associate each regression coefficient, either a main effect or the coefficient of a basis function, with a latent binary variable that indicates whether the corresponding covariate is needed in the model or not. Hence, the joint posterior distribution of the vector of these binary variables can identify the models with the higher posterior probability.

R packages that are related to **BNSP** include **spikeSlabGAM** (Scheipl, 2016) that also utilizes SSVS methods (Scheipl, 2011). A major difference between the two packages, however, is that whereas **spikeSlabGAM** utilizes spike-and-slab priors for function selection, **BNSP** utilizes spike-and-slab priors for variable selection. In addition, Bayesian GAMLSS models, also refer to as distributional regression models, can also be fit with R package **brms** using normal priors (Bürkner, 2018). Further, the R package **gamboostLSS** (Hofner et al., 2018) includes frequentist GAMLSS implementation based on boosting that can handle high-dimensional data (Mayr et al., 2012). Lastly, the R package **mgcv** (Wood, 2018) can also fit generalized additive models with Gaussian errors and integrated smoothness estimation, with implementations that can handle large datasets.

In **BNSP** we have implemented functions for fitting such semi-parametric models, summarizing model fits, visualizing covariate effects and predicting new responses or their means. The main functions are `mvrm`, `mvrm2mcmc`, `print.mvrm`, `summary.mvrm`, `plot.mvrm`, and `predict.mvrm`. A quick description of these functions follows. The first one, `mvrm`, returns samples from the posterior distributions of the model parameters, and it is based on an efficient Markov chain Monte Carlo (MCMC) algorithm in which we integrate out the coefficients in the mean function, generate the variable selection indicators in blocks (Chan et al., 2006), and choose the MCMC tuning parameters adaptively (Roberts and Rosenthal, 2009). In order to minimize random-access memory utilization, posterior samples are not kept in memory, but instead written in files in a directory supplied by the user. The second function, `mvrm2mcmc`, reads-in the samples from the posterior of the model parameters and it creates an object of class `"mcmc"`. This enables users to easily utilize functions from package **coda** (Plummer et al., 2006), including its `plot` and `summary` methods for assessing convergence and for summarizing posterior distributions. Further, functions `print.mvrm` and `summary.mvrm` provide summaries of model fits, including models and priors specified, marginal posterior probabilities of term inclusion in the mean and variance models and models with the highest posterior probabilities. Function `plot.mvrm` creates plots of parametric and nonparametric terms that appear in the mean and variance models. The function can create two-dimensional plots by calling functions from the package **ggplot2** (Wickham, 2009). It can also create static or interactive three-dimensional plots by calling functions from the packages **plot3D** (Soetaert, 2016) and **threejs** (Lewis, 2016). Lastly, function `predict.mvrm` provides predictions either for new responses or their means given feature/covariate vectors.

We next provide a detailed model description followed by illustrations on the usage of the package and the options it provides. Technical details on the implementation of the MCMC algorithm are provided in the Appendix. The paper concludes with a brief summary.

Mean-variance nonparametric regression models

Let $\boldsymbol{y} = (y_1, \dots, y_n)^\top$ denote the vector of responses and let $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^\top$ and $\boldsymbol{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_n]^\top$ denote design matrices. The models that we consider express the vector of responses utilizing

$$\boldsymbol{Y} = \beta_0 \mathbf{1}_n + \boldsymbol{X}\boldsymbol{\beta}_1 + \boldsymbol{\epsilon},$$

where $\mathbf{1}_n$ is the usual n-dimensional vector of ones, β_0 is an intercept term, $\boldsymbol{\beta}_1$ is a vector of regression coefficients and $\boldsymbol{\epsilon} = (\epsilon_1, \dots, \epsilon_n)^\top$ is an n-dimensional vector of independent random errors. Each $\epsilon_i, i = 1, \dots, n$, is assumed to have a normal distribution, $\epsilon_i \sim N(0, \sigma_i^2)$, with variances that are modeled in terms of covariates. Let $\boldsymbol{\sigma}^2 = (\sigma_1^2, \dots, \sigma_n^2)^\top$. We model the vector of variances utilizing

$$\log(\boldsymbol{\sigma}^2) = \alpha_0 \mathbf{1}_n + \boldsymbol{Z}\boldsymbol{\alpha}_1,$$

where α_0 is an intercept term and $\boldsymbol{\alpha}_1$ is a vector of regression coefficients. Equivalently, the model for the variances can be expressed as

$$\sigma_i^2 = \sigma^2 \exp(\boldsymbol{z}_i^\top \boldsymbol{\alpha}_1), i = 1, \dots, n, \tag{1}$$

where $\sigma^2 = \exp(\alpha_0)$.

Let $\boldsymbol{D}(\boldsymbol{\alpha})$ denote an n-dimensional, diagonal matrix with elements $\exp(\boldsymbol{z}_i^\top \boldsymbol{\alpha}_1/2), i = 1, \dots, n$. Then, the model that we consider may be expressed more economically as

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{X}^* \boldsymbol{\beta} + \boldsymbol{\epsilon}, \\ \boldsymbol{\epsilon} &\sim N(\mathbf{0}, \sigma^2 \boldsymbol{D}^2(\boldsymbol{\alpha})), \end{aligned} \tag{2}$$

where $\boldsymbol{\beta} = (\beta_0, \boldsymbol{\beta}_1^\top)^\top$ and $\boldsymbol{X}^* = [\mathbf{1}_n, \boldsymbol{X}]$.

In the following subsections we describe how, within model (2), both parametric and nonparametric effects of explanatory variables on the mean and variance functions can be captured utilizing regression splines and variable selection methods. We begin by considering the special case where there is a single covariate entering the mean model and a single covariate entering the variance model.

Locally adaptive models with a single covariate

Suppose that the observed dataset consists of triplets $(y_i, u_i, w_i), i = 1, \ldots, n$, where explanatory variables u and w enter flexibly the mean and variance models, respectively. To model the nonparametric effects of u and w we consider the following formulations of the mean and variance models

$$\mu_i = \beta_0 + f_\mu(u_i) = \beta_0 + \sum_{j=1}^{q_1} \beta_j \phi_{1j}(u_i) = \beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta}_1, \tag{3}$$

$$\log(\sigma_i^2) = \alpha_0 + f_\sigma(w_i) = \alpha_0 + \sum_{j=1}^{q_2} \alpha_j \phi_{2j}(w_i) = \alpha_0 + \boldsymbol{z}_i^\top \boldsymbol{\alpha}_1. \tag{4}$$

In the preceding $\boldsymbol{x}_i = (\phi_{11}(u_i), \ldots, \phi_{1q_1}(u_i))^\top$ and $\boldsymbol{z}_i = (\phi_{21}(w_i), \ldots, \phi_{2q_2}(w_i))^\top$ are vectors of basis functions and $\boldsymbol{\beta}_1 = (\beta_1, \ldots, \beta_{q_1})^\top$ and $\boldsymbol{\alpha}_1 = (\alpha_1, \ldots, \alpha_{q_2})^\top$ are the corresponding coefficients.

In package **BNSP** we implemented two sets of basis functions. Firstly, radial basis functions

$$\begin{aligned}\mathcal{B}_1 = \{&\phi_1(u) = u, \phi_2(u) = ||u - \xi_1||^2 \log\left(||u - \xi_1||^2\right), \ldots, \\ &\phi_q(u) = ||u - \xi_{q-1}||^2 \log\left(||u - \xi_{q-1}||^2\right)\},\end{aligned} \tag{5}$$

where $||u||$ denotes the Euclidean norm of u and $\xi_1, \ldots, \xi_{q-1}$ are the knots that within package **BNSP** are chosen as the quantiles of the observed values of explanatory variable u, with $\xi_1 = \min(u_i)$, $\xi_{q-1} = \max(u_i)$ and the remaining knots chosen as equally spaced quantiles between ξ_1 and ξ_{q-1}.

Secondly, we implemented thin plate splines

$$\mathcal{B}_2 = \{\phi_1(u) = u, \phi_2(u) = (u - \xi_1)_+, \ldots, \phi_q(u) = (u - \xi_q)_+\},$$

where $(a)_+ = \max(a, 0)$ and the knots $\xi_1, \ldots, \xi_{q-1}$ are determined as above.

In addition, **BNSP** supports the smooth constructors from package **mgcv** (e.g., the low-rank thin plate splines, cubic regression splines, P-splines, their cyclic versions, and others). Examples on how these smooth terms are used within **BNSP** are provided later in this paper.

Locally adaptive models for the mean and variance functions are obtained utilizing the methodology developed by Chan et al. (2006). Considering the mean function, local adaptivity is achieved by utilizing a large number of basis functions, q_1. Over-fitting, and problems associated with it, is avoided by allowing positive prior probability that the regression coefficients are exactly zero. The latter is achieved by defining binary variables $\gamma_j, j = 1, \ldots, q_1$, that take value $\gamma_j = 1$ if $\beta_j \neq 0$ and $\gamma_j = 0$ if $\beta_j = 0$. Hence, vector $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_{q_1})^\top$ determines which terms enter the mean model. The vector of indicators $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_{q_2})^\top$ for the variance function is defined analogously.

Given vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$, the heteroscedastic, semiparametric model (2) can be written as

$$\begin{aligned}\boldsymbol{Y} &= \boldsymbol{X}_\gamma^* \boldsymbol{\beta}_\gamma + \boldsymbol{\epsilon}, \\ \boldsymbol{\epsilon} &\sim N(\boldsymbol{0}, \sigma^2 \boldsymbol{D}^2(\boldsymbol{\alpha}_\delta)),\end{aligned}$$

where $\boldsymbol{\beta}_\gamma$ consisting of all non-zero elements of $\boldsymbol{\beta}_1$ and $\boldsymbol{X}_\gamma^*$ consists of the corresponding columns of $\boldsymbol{X}^*$. Subvector $\boldsymbol{\alpha}_\delta$ is defined analogously.

We note that, as was suggested by Chan et al. (2006), we work with mean corrected columns in the design matrices $\boldsymbol{X}$ and $\boldsymbol{Z}$, both in this paper and in the **BNSP** implementation. We remove the mean from all columns in the design matrices except those that correspond to categorical variables.

Prior specification for models with a single covariate

Let $\tilde{\boldsymbol{X}} = \boldsymbol{D}(\boldsymbol{\alpha})^{-1}\boldsymbol{X}^*$. The prior for $\boldsymbol{\beta}_\gamma$ is specified as (Zellner, 1986)

$$\boldsymbol{\beta}_\gamma | c_\beta, \sigma^2, \boldsymbol{\gamma}, \boldsymbol{\alpha}, \boldsymbol{\delta} \sim N(\boldsymbol{0}, c_\beta \sigma^2 (\tilde{\boldsymbol{X}}_\gamma^\top \tilde{\boldsymbol{X}}_\gamma)^{-1}).$$

Further, the prior for $\boldsymbol{\alpha}_\delta$ is specified as

$$\boldsymbol{\alpha}_\delta | c_\alpha, \boldsymbol{\delta} \sim N(\boldsymbol{0}, c_\alpha \boldsymbol{I}).$$

Independent priors are specified for the indicators variables γ_j as $P(\gamma_j = 1|\pi_\mu) = \pi_\mu, j = 1, \ldots, q_1,$

from which the joint prior is obtained as

$$P(\gamma|\pi_\mu) = \pi_\mu^{N(\gamma)}(1-\pi_\mu)^{q_1-N(\gamma)},$$

where $N(\gamma) = \sum_{j=1}^{q_1} \gamma_j$.

Similarly, for the indicators δ_j we specify independent priors $P(\delta_j = 1|\pi_\sigma) = \pi_\sigma, j = 1, \dots, q_2$. It follows that the joint prior is

$$P(\delta|\pi_\sigma) = \pi_\sigma^{N(\delta)}(1-\pi_\sigma)^{q_2-N(\delta)},$$

where $N(\delta) = \sum_{j=1}^{q_2} \delta_j$.

We specify inverse Gamma priors for c_β and c_α and Beta priors for π_μ and π_σ

$$\begin{aligned} &c_\beta \sim \text{IG}(a_\beta, b_\beta), c_\alpha \sim \text{IG}(a_\alpha, b_\alpha), \\ &\pi_\mu \sim \text{Beta}(c_\mu, d_\mu), \pi_\sigma \sim \text{Beta}(c_\sigma, d_\sigma). \end{aligned} \tag{6}$$

Lastly, for σ^2 we consider inverse Gamma and half-normal priors

$$\sigma^2 \sim \text{IG}(a_\sigma, b_\sigma) \text{ and } |\sigma| \sim N(0, \phi_\sigma^2). \tag{7}$$

Extension to bivariate covariates

It is straightforward to extend the methodology described earlier to allow fitting of flexible mean and variance surfaces. In fact, the only modification required is in the basis functions and knots. For fitting surfaces, in package **BNSP** we implemented radial basis functions

$$\begin{aligned} \mathcal{B}_3 = \Big\{ &\phi_1(\boldsymbol{u}) = u_1, \phi_2(\boldsymbol{u}) = u_2, \phi_3(\boldsymbol{u}) = ||\boldsymbol{u} - \boldsymbol{\xi}_1||^2 \log\left(||\boldsymbol{u} - \boldsymbol{\xi}_1||^2\right), \dots, \\ &\phi_q(\boldsymbol{u}) = ||\boldsymbol{u} - \boldsymbol{\xi}_{q-2}||^2 \log\left(||\boldsymbol{u} - \boldsymbol{\xi}_{q-2}||^2\right) \Big\}. \end{aligned}$$

We note that the prior specification presented earlier for fitting flexible functions remains unchained for fitting flexible surfaces. Further, for fitting bivariate or higher order functions, **BNSP** also supports smooth constructors `s`, `te`, and `ti` from **mgcv**.

Extension to additive models

In the presence of multiple covariates, the effects of which may be modeled parametrically or semi-parametrically, the mean model in (3) is extended to the following

$$\mu_i = \beta_0 + \boldsymbol{u}_{ip}^\top \boldsymbol{\beta} + \sum_{k=1}^{K_1} f_{\mu,k}(u_{ik}), i = 1, \dots, n,$$

where $\boldsymbol{u}_{ip}$ includes the covariates the effects of which are modeled parametrically, $\boldsymbol{\beta}$ denotes the corresponding effects, and $f_{\mu,k}(u_{ik}), k = 1, \dots, K_1$, are flexible functions of one or more covariates expressed as

$$f_{\mu,k}(u_{ik}) = \sum_{j=1}^{q_{1k}} \beta_{kj}\phi_{1kj}(u_{ik}),$$

where $\phi_{1kj}, j = 1, \dots, q_{1k}$ are the basis functions used in the kth component, $k = 1, \dots, K_1$.

Similarly, the variance model (4), in the presence of multiple covariates, is expressed as

$$\log(\sigma_i^2) = \alpha_0 + \boldsymbol{w}_{ip}^\top \boldsymbol{\alpha} + \sum_{k=1}^{K_2} f_{\sigma,k}(w_{ik}), i = 1, \dots, n,$$

where

$$f_{\sigma,k}(w_{ik}) = \sum_{j=1}^{q_{2k}} \alpha_{kj}\phi_{2kj}(w_{ik}).$$

For additive models, local adaptivity is achieved using a similar strategy, as in the single covariate case. That is, we utilize a potentially large number of knots or basis functions in the flexible components that appear in the mean model, $f_{\mu,k}, k = 1, \dots, K_1$, and in the variance model, $f_{\sigma,k}, k = 1, \dots, K_2$. To avoid over-fitting, we allow removal of the unnecessary ones utilizing the usual indicator variables,

$\gamma_k = (\gamma_{k1}, \ldots, \gamma_{kq_{1k}})^\top, k = 1, \ldots, K_1$, and $\delta_k = (\delta_{k1}, \ldots, \delta_{kq_{2k}})^\top, k = 1, \ldots, K_2$. Here, vectors γ_k and δ_k determine which basis functions appear in $f_{\mu,k}$ and $f_{\sigma,k}$ respectively.

The model that we implemented in package **BNSP** specifies independent priors for the indicators variables γ_{kj} as $P(\gamma_{kj} = 1|\pi_{\mu_k}) = \pi_{\mu_k}, j = 1, \ldots, q_{1k}$. From these, the joint prior follows

$$P(\gamma_k|\pi_{\mu_k}) = \pi_{\mu_k}^{N(\gamma_k)}(1 - \pi_{\mu_k})^{q_{1k} - N(\gamma_k)},$$

where $N(\gamma_k) = \sum_{j=1}^{q_{1k}} \gamma_{kj}$.

Similarly, for the indicators δ_{kj} we specify independent priors $P(\delta_{kj} = 1|\pi_{\sigma_k}) = \pi_{\sigma_k}, j = 1, \ldots, q_{2k}$. It follows that the joint prior is

$$P(\delta_k|\pi_{\sigma_k}) = \pi_{\sigma_k}^{N(\delta_k)}(1 - \pi_{\sigma_k})^{q_{2k} - N(\delta_k)},$$

where $N(\delta_k) = \sum_{j=1}^{q_{2k}} \delta_{kj}$.

We specify the following independent priors for the inclusion probabilities.

$$\pi_{\mu_k} \sim \text{Beta}(c_{\mu_k}, d_{\mu_k}), k = 1, \ldots, K_1 \quad \pi_{\sigma_k} \sim \text{Beta}(c_{\sigma_k}, d_{\sigma_k}), k = 1, \ldots, K_2. \tag{8}$$

The rest of the priors are the same as those specified for the single covariate models.

Usage

In this section we provide results on simulation studies and real data analyses. The purpose is twofold: firstly we point out that the package works well and provides the expected results (in simulation studies) and secondly we illustrate the options that the users of **BNSP** have.

Simulation studies

Here we present results from three simulations studies, involving one, two, and multiple covariates. For the majority of these simulation studies, we utilize the same data-generating mechanisms as those presented by Chan et al. (2006).

Single covariate case

We consider two mechanisms that involve a single covariate that appears in both the mean and variance model. Denoting the covariate by u, the data-generating mechanisms are the normal model $Y \sim N\{\mu(u), \sigma^2(u)\}$ with the following mean and standard deviation functions:

1. $\mu(u) = 2u, \sigma(u) = 0.1 + u$,
2. $\mu(u) = \left\{N(u, \mu = 0.2, \sigma^2 = 0.004) + N(u, \mu = 0.6, \sigma^2 = 0.1)\right\}/4$,
   $\sigma(u) = \left\{N(u, \mu = 0.2, \sigma^2 = 0.004) + N(u, \mu = 0.6, \sigma^2 = 0.1)\right\}/6$.

We generate a single dataset of size $n = 500$ from each mechanism, where variable u is obtained from the uniform distribution, $u \sim \text{Unif}(0, 1)$. For instance, for obtaining a dataset from the first mechanism we use

```
> mu <- function(u) {2 * u}
> stdev <- function(u) {0.1 + u}
> set.seed(1)
> n <- 500
> u <- sort(runif(n))
> y <- rnorm(n, mu(u), stdev(u))
> data <- data.frame(y, u)
```

Above we specified the seed value to be one, and we do so in what follows, so that our results are replicable.

To the generated dataset we fit a special case of the model that we presented, where the mean and variance functions in (3) and (4) are specified as

$$\mu = \beta_0 + f_\mu(u) = \beta_0 + \sum_{j=1}^{q_1} \beta_j \phi_{1j}(u) \quad \text{and} \quad \log(\sigma^2) = \alpha_0 + f_\sigma(u) = \alpha_0 + \sum_{j=1}^{q_2} \alpha_j \phi_{2j}(u), \tag{9}$$

with ϕ denoting the radial basis functions presented in (5). Further, we choose $q_1 = q_2 = 21$ basis functions or, equivalently, 20 knots. Hence, we have $\phi_{1j}(u) = \phi_{2j}(u), j = 1, \ldots, 21$, which results in identical design matrices for the mean and variance models. In R, the two models are specified using

```
> model <- y ~ sm(u, k = 20, bs = "rd") | sm(u, k = 20, bs = "rd")
```

The above formula (Zeileis and Croissant, 2010) specifies the response, mean and variance models. Smooth terms are specified utilizing function sm, that takes as input the covariate u, the selected number of knots and the selected type of basis functions.

Next we specify the hyper-parameter values for the priors in (6) and (7). The default prior for c_β is inverse Gamma with $a_\beta = 0.5, b_\beta = n/2$ (Liang et al., 2008). For parameter c_α the default prior is a non-informative but proper inverse Gamma with $a_\alpha = b_\alpha = 1.1$. Concerning π_μ and π_σ, the default priors are uniform, obtained by setting $c_\mu = d_\mu = 1$ and $c_\sigma = d_\sigma = 1$. Lastly, the default prior for the error standard deviation is the half-normal with variance $\phi^2_\sigma = 2$, $|\sigma| \sim N(0,2)$.

We choose to run the MCMC sampler for 10,000 iterations and discard the first 5,000 as burn-in. Of the remaining 5,000 samples we retain 1 of every 2 samples, resulting in 2,500 posterior samples. Further, as mentioned above, we set the seed of the MCMC sampler equal to one. Obtaining posterior samples is achieved by a function call of the form

```
> DIR <- getwd()
> m1 <- mvrm(formula = model, data = data, sweeps = 10000, burn = 5000,
+       thin = 2, seed = 1, StorageDir = DIR,
+       c.betaPrior = "IG(0.5,0.5*n)", c.alphaPrior = "IG(1.1,1.1)",
+       pi.muPrior = "Beta(1,1)", pi.sigmaPrior = "Beta(1,1)", sigmaPrior = "HN(2)")
```

Samples from the posteriors of the model parameters $\{\boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\alpha}, \boldsymbol{\delta}, c_\beta, c_\alpha, \sigma^2\}$ are written in seven separate files which are stored in the directory specified by argument StorageDir. If a storage directory is not specified, then function mvrm returns an error message, as without these files there will be no output to process. Furthermore, the last two lines of the above function call show the specified priors, which are $c_\beta \sim \text{IG}(0.5, n/2)$, $c_\alpha \sim \text{IG}(1.1, 1.1)$, $\pi_\mu \sim \text{Beta}(1,1)$, $\pi_\sigma \sim \text{Beta}(1,1)$ and $|\sigma| \sim N(0,2)$, respectively. As we mentioned above, these priors are the default ones, and hence the same function call can be achieved without specifying the last two lines. Here we display the priors in order to describe how users can specify their own priors. For parameters c_β and c_α only inverse Gamma priors are available, with parameters that can be specified by the user in the intuitive way. For instance, the prior $c_\beta \sim \text{IG}(1.01, 1.01)$ can be specified in function mvrm by using c.betaPrior = "IG(1.01,1.01)". The second parameter of the prior for c_β can be a function of the sample size n (but only symbol n would work here), so for instance c.betaPrior = "IG(1,0.4*n)" is another acceptable specification. Further, Beta priors are available for parameters π_μ and π_σ with parameters that can be specified by the user again in the intuitive way. Lastly, two priors are available for the error variance. These are the default half-normal and the inverse Gamma. For instance, sigmaPrior = "HN(5)" defines $|\sigma| \sim N(0,5)$ as the prior while sigmaPrior = "IG(1.1,1.1)" defines $\sigma^2 \sim \text{IG}(1.1, 1.1)$ as the prior.

Function mvrm2mcmc reads in posterior samples from the files that the call to function mvrm generated and creates an object of class "mcmc". Hence, for summarizing posterior distributions and for assessing convergence, functions summary and plot from package **coda** can be used. As an example, here we read in the samples from the posterior of $\boldsymbol{\beta}$ and summarize the posterior using summary. For the sake of economizing space, only the part of the output that describes the posteriors of β_0, β_1, and β_2 is shown below:

```
> beta <- mvrm2mcmc(m1, "beta")

> summary(beta)

Iterations = 5001:9999
Thinning interval = 2
Number of chains = 1
Sample size per chain = 2500

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                  Mean       SD  Naive SE Time-series SE
(Intercept)  9.534e-01 0.004399 8.799e-05      0.0002534
u            1.864e+00 0.042045 8.409e-04      0.0010356
sm(u).1      3.842e-04 0.016421 3.284e-04      0.0003284
```

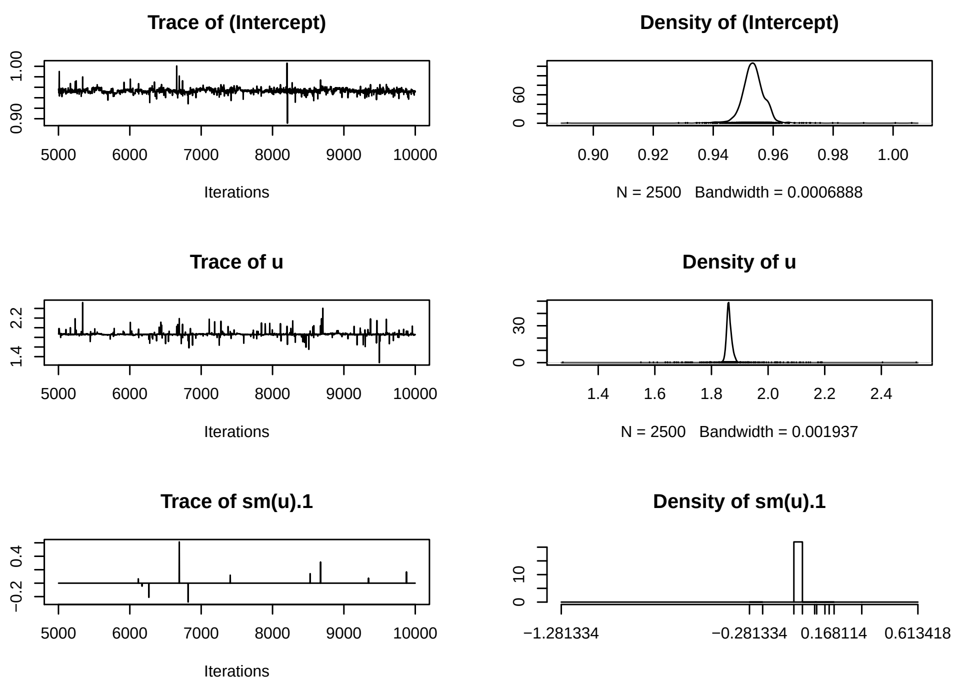


Figure 1: Trace and density plots for the regression coefficients β_0, β_1 and β_2 of the first simulated example. Parameters β_1 and β_2 are the coefficients of the first two basis functions, denoted by "u" and "sm(u).1". Plots for coefficients $\beta_3, \ldots, \beta_{21}$ are omitted as they follow a very similar pattern to that seen for β_2 (i.e., most of the time they take value zero but with random spikes away from zero).

```
2. Quantiles for each variable:

              2.5%    25%    50%    75%  97.5%
(Intercept) 0.946 0.9513 0.9533 0.9554 0.960
u           1.833 1.8565 1.8614 1.8682 1.923
sm(u).1     0.000 0.0000 0.0000 0.0000 0.000
```

Further, we may obtain a plot using

```
> plot(beta)
```

Figure 1 shows the first three of the plots created by function plot. These are the plots of the samples from the posteriors of coefficients β_0, β_1 and β_2. As we can see from both the summary and Figure 1, only the first two coefficients have posteriors that are not centered around zero.

Returning to the function mvrm2mcmc, it requires two inputs. These are an object of class "mvrm" and the name of the file to be read in R. For the parameters in the current model $\{\boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\alpha}, \boldsymbol{\delta}, c_\beta, c_\alpha, \sigma^2\}$ the corresponding file names are 'beta', 'gamma', 'alpha', 'delta', 'cbeta', 'calpha', and 'sigma2' respectively.

Summaries of mvrm fits may be obtained utilizing functions print.mvrm and summary.mvrm. The method print takes as input an object of class "mvrm". It returns basic information of the model fit, as shown below:

```
> print(m1)

Call:
mvrm(formula = model, data = data, sweeps = 10000, burn = 5000,
    thin = 2, seed = 1, StorageDir = DIR, c.betaPrior = "IG(0.5,0.5*n)",
    c.alphaPrior = "IG(1.1,1.1)", pi.muPrior = "Beta(1,1)",
    pi.sigmaPrior = "Beta(1,1)", sigmaPrior = "HN(2)")

2500 posterior samples

Mean model - marginal inclusion probabilities
```

```
      u  sm(u).1  sm(u).2  sm(u).3  sm(u).4  sm(u).5  sm(u).6  sm(u).7
  1.0000   0.0040   0.0036   0.0032   0.0084   0.0036   0.0044   0.0028
 sm(u).8  sm(u).9 sm(u).10 sm(u).11 sm(u).12 sm(u).13 sm(u).14 sm(u).15
  0.0060   0.0020   0.0060   0.0036   0.0056   0.0056   0.0036   0.0052
sm(u).16 sm(u).17 sm(u).18 sm(u).19 sm(u).20
  0.0060   0.0044   0.0056   0.0044   0.0052

Variance model - marginal inclusion probabilities
      u  sm(u).1  sm(u).2  sm(u).3  sm(u).4  sm(u).5  sm(u).6  sm(u).7
  1.0000   0.6072   0.5164   0.5808   0.5488   0.6760   0.5320   0.6336
 sm(u).8  sm(u).9 sm(u).10 sm(u).11 sm(u).12 sm(u).13 sm(u).14 sm(u).15
  0.6936   0.6708   0.5996   0.4816   0.4912   0.3728   0.6268   0.5688
sm(u).16 sm(u).17 sm(u).18 sm(u).19 sm(u).20
  0.5872   0.6528   0.4428   0.6900   0.5356
```

The function returns a matched call, the number of posterior samples obtained, and marginal inclusion probabilities of the terms in the mean and variance models.

Whereas the output of the `print` method focuses on marginal inclusion probabilities, the output of the `summary` method focuses on the most frequently visited models. It takes as input an object of class `"mvrm"` and the number of (most frequently visited) models to be displayed, which by default is set to `nModels = 5`. Here to economize space we set `nModels = 2`. The information returned by the `summary` method is shown below

```
> summary(m1, nModels = 2)

Specified model for the mean and variance:
y ~ sm(u, k = 20, bs = "rd") | sm(u, k = 20, bs = "rd")

Specified priors:
[1] c.beta = IG(0.5,0.5*n) c.alpha = IG(1.1,1.1)  pi.mu = Beta(1,1)
[4] pi.sigma = Beta(1,1)   sigma = HN(2)

Total posterior samples: 2500 ; burn-in: 5000 ; thinning: 2

Files stored in /home/papgeo/1/

Null deviance:           1299.292
Mean posterior deviance:  -88.691

Joint mean/variance model posterior probabilities:
  mean.u mean.sm.u..1 mean.sm.u..2 mean.sm.u..3 mean.sm.u..4 mean.sm.u..5
1      1            0            0            0            0            0
2      1            0            0            0            0            0
  mean.sm.u..6 mean.sm.u..7 mean.sm.u..8 mean.sm.u..9 mean.sm.u..10
1            0            0            0            0             0
2            0            0            0            0             0
  mean.sm.u..11 mean.sm.u..12 mean.sm.u..13 mean.sm.u..14 mean.sm.u..15
1             0             0             0             0             0
2             0             0             0             0             0
  mean.sm.u..16 mean.sm.u..17 mean.sm.u..18 mean.sm.u..19 mean.sm.u..20 var.u
1             0             0             0             0             0     1
2             0             0             0             0             0     1
  var.sm.u..1 var.sm.u..2 var.sm.u..3 var.sm.u..4 var.sm.u..5 var.sm.u..6
1           1           1           1           1           1           1
2           1           0           1           1           1           1
  var.sm.u..7 var.sm.u..8 var.sm.u..9 var.sm.u..10 var.sm.u..11 var.sm.u..12
1           1           1           1            0            1            0
2           1           1           1            1            1            1
  var.sm.u..13 var.sm.u..14 var.sm.u..15 var.sm.u..16 var.sm.u..17 var.sm.u..18
1            1            1            1            1            1            1
2            0            1            1            1            1            0
  var.sm.u..19 var.sm.u..20 freq prob cumulative
1            1            1  141 5.64       5.64
2            1            0  120 4.80      10.44
Displaying 2 models of the 916 visited
```

```
2 models account for 10.44% of the posterior mass
```

Firstly, the method provides the specified mean and variance models and the specified priors. This is followed by information about the MCMC chain and the directory where files have been stored. In addition, the function provides the null and the mean posterior deviance. Finally, the function provides the specification of the joint mean/variance models that were visited most often during MCMC sampling. This specification is in terms of a vector of indicators, each consisting of zeros and ones, that show which terms are in the mean and variance model. To make clear which terms pertain to the mean and which to the variance function, we have preceded the names of the model terms by "mean." or "var.". In the above output, we see that the most visited model specifies a linear mean model (only the linear term is included in the model) while the variance model includes twelve terms. See also Figure 2.

We next describe the function plot.mvrm which creates plots of terms in the mean and variance functions. Two calls to the plot method can be seen in the code below. Argument x expects an object of class "mvrm", as created by a call to the function mvrm. The model argument may take on one of three possible values: "mean", "stdev", or "both", specifying the model to be visualized. Further, the term argument determines the term to be plotted. In the current example there is only one term in each of the two models which leaves us with only one choice, term = "sm(u)". Equivalently, term may be specified as an integer, term = 1. If term is left unspecified, then, by default, the first term in the model is plotted. For creating two-dimensional plots, as in the current example, the plot method utilizes the package **ggplot2**. Users of **BNSP** may add their own options to plots via the argument plotOptions. The code below serves as an example.

```
> x1 <- seq(0, 1, length.out = 30)
> plotOptionsM <- list(geom_line(aes_string(x = x1, y = mu(x1)), col = 2, alpha = 0.5,
+                      lty = 2), geom_point(data = data, aes(x = u, y = y)))
> plot(x = m1, model = "mean", term = "sm(u)", plotOptions = plotOptionsM,
+      intercept = TRUE, quantiles = c(0.005, 0.995), grid = 30)
> plotOptionsV = list(geom_line(aes_string(x = x1, y = stdev(x1)), col = 2,
+                     alpha = 0.5, lty = 2))
> plot(x = m1, model = "stdev", term = "sm(u)", plotOptions = plotOptionsV,
+      intercept = TRUE, quantiles = c(0.05, 0.95), grid = 30)
```

The resulting plots can be seen in Figure 2, panels (a) and (b). Panel (a) displays the simulated dataset, showing the expected increase in both the mean and variance with increasing values of the covariate u. Further, we see the posterior mean of $\mu(u) = \beta_0 + f_\mu(u) = \beta_0 + \sum_{j=1}^{21} \beta_j \phi_{1j}(u)$ evaluated over a grid of 30 values of u, as specified by the (default) grid = 30 option for plot. For each sample $\boldsymbol{\beta}^{(s)}, s = 1, \ldots, S$, from the posterior of $\boldsymbol{\beta}$, and for each value of u over the grid of 30 values, $u_j, j = 1, \ldots, 30$, the plot method computes $\mu(u_j)^{(s)} = \beta_0^{(s)} + \sum_{j=1}^{21} \beta_j^{(s)} \phi_{1j}(u_j)$. The default option intercept = TRUE specifies that the intercept β_0 is included in the computation, but it may be removed by setting intercept = FALSE. The posterior means are computed by the usual $\bar{\mu}(u_j) = S^{-1} \sum_s \mu(u_j)^{(s)}$ and are plotted with solid (blue-color) line. By default, the function displays 80% point-wise credible intervals (CI). In Figure 2, panel (a) we have plotted 99% CIs, as specified by option quantiles = c(0.005,0.995). This option specifies that for each value $u_j, j = 1, \ldots, 30$, on the grid, 99% CIs for $\mu(u_j)$ are computed by the 0.5% and 99.5% quantiles of the samples $\mu(u_j)^{(s)}, s = 1, \ldots, S$. Plots without credible intervals may be obtained by setting quantiles = NULL.

Figure 2, panel (b) displays the posterior mean of the standard deviation function $\sigma(u) = \sigma \exp\{\sum_{j=1}^{q_2} \alpha_j \phi_{2j}(u)/2\}$. The details are the same as for the plot of the mean function, so here we briefly mention a difference: option intercept = TRUE specifies that σ is included in the calculation. It may be removed by setting intercept = FALSE, which will result in plots of $\sigma(u)^* = \exp\{\sum_{j=1}^{q_2} \alpha_j \phi_{2j}(u)/2\}$.

We use the second simulated dataset to show how the s constructor from the package **mgcv** may be used. In our example, we use s to specify the model:

```
> model <- y ~ s(u, k = 15, bs = "ps", absorb.cons=TRUE) |
+              s(u, k = 15, bs = "ps", absorb.cons=TRUE)
```

Function BNSP::s calls in turn mgcv::s and mgcv::smoothCon. All options of the last two functions may be passed to BNSP::s. In the example above we used options k, bs, and absorb.cons.

The remaining R code for the second simulated example is precisely the same as the one for the first example, and hence omitted. Results are shown in Figure 2, panels (c) and (d).

We conclude the current section by describing the function predict.mvrm. The function provides predictions and posterior credible or prediction intervals for given feature vectors. The two types of intervals differ in the associated level of uncertainty: prediction intervals attempt to capture a future response and are usually much wider than credible intervals that attempt to capture a mean response.

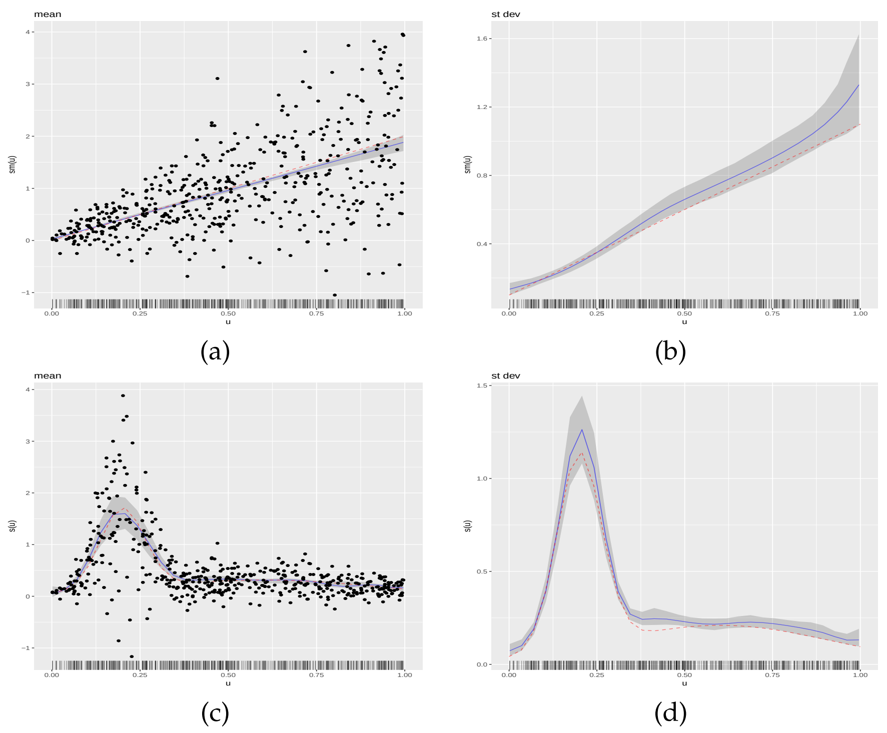


Figure 2: Results from the single covariate simulated examples. The column on the left-hand side displays the generated data points and posterior means of the estimated effect along with 99% CIs. The column on the right-hand side displays the posterior mean of the estimated standard deviation function along with 90% CIs. In all panels, the truth is represented by dashed (red color) lines, the posterior means by solid (blue color) lines, and the CIs by gray color.

The following code shows how credible and prediction intervals can be obtained for a sequence of covariate values stored in `x1`

```
> x1 <- seq(0, 1, length.out = 30)
> p1 <- predict(m1, newdata = data.frame(u = x1), interval = "credible")
> p2 <- predict(m1, newdata = data.frame(u = x1), interval = "prediction")
```

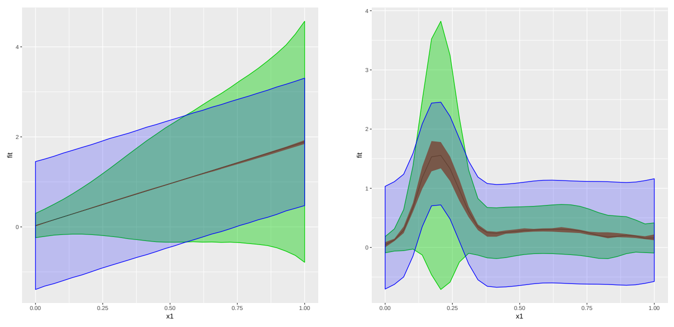
where the first argument in the `predict` method is a fitted `mvrm` model, the second one is a data frame containing the feature vectors at which predictions are to be obtained and the last one defines the type of interval to be created. We applied the `predict` method to the two simulated datasets. To each of those datasets we fitted two models: the first one is the one we saw earlier, where both the mean and variance are modeled in terms of covariates, while the second one ignores the dependence of the variance on the covariate. The latter model is specified in R using

```
> model <- y ~ sm(u, k = 20, bs = "rd") | 1
```

Results are displayed in Figure 3. Each of the two figures displays a credible interval and two prediction intervals. The figure emphasizes a point that was discussed in the introductory section, that modeling the variance in terms of covariates can result in more realistic prediction intervals. The same point was recently discussed by Mayr et al. (2012).

Bivariate covariate case

Interactions between two predictors can be modeled by appropriate specification of either the built-in `sm` function or the smooth constructors from **mgcv**. Function `sm` can take up to two covariates, both of which may be continuous or one continuous and one discrete. Next we consider an example that involves two continuous covariates. An example involving a continuous and a discrete covariate is shown later on, in the second application to a real dataset.



(a) (b)

Figure 3: Predictions results from the first two simulated datasets. Each panel displays a credible interval and two prediction intervals, one obtained using a model that recognizes the dependence of the variance on the covariate and one that ignores it.

Let $\boldsymbol{u} = (u_1, u_2)^\top$ denote a bivariate predictor. The data-generating mechanism that we consider is

$$
\begin{aligned}
&y(\boldsymbol{u}) \sim N\{\mu(\boldsymbol{u}), \sigma^2(\boldsymbol{u})\},\\
&\mu(\boldsymbol{u}) = 0.1 + N\left(\boldsymbol{u}, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right) + N\left(\boldsymbol{u}, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\right),\\
&\sigma^2(\boldsymbol{u}) = 0.1 + \left\{N\left(\boldsymbol{u}, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right) + N\left(\boldsymbol{u}, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2\right)\right\}/2,\\
&\boldsymbol{\mu}_1 = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix}, \boldsymbol{\Sigma}_1 = \begin{pmatrix} 0.03 & 0.01 \\ 0.01 & 0.03 \end{pmatrix}, \boldsymbol{\mu}_2 = \begin{pmatrix} 0.65 \\ 0.35 \end{pmatrix}, \boldsymbol{\Sigma}_2 = \begin{pmatrix} 0.09 & 0.01 \\ 0.01 & 0.09 \end{pmatrix}.
\end{aligned}
$$

As before, u_1 and u_2 are obtained independently from uniform distributions on the unit interval. Further, the sample size is set to $n = 500$.

In R, we simulate data from the above mechanism using

```
> mu1 <- matrix(c(0.25, 0.75))
> sigma1 <- matrix(c(0.03, 0.01, 0.01, 0.03), 2, 2)
> mu2 <- matrix(c(0.65, 0.35))
> sigma2 <- matrix(c(0.09, 0.01, 0.01, 0.09), 2, 2)
> mu <- function(x1, x2) {x <- cbind(x1, x2);
+       0.1 + dmvnorm(x, mu1, sigma1) + dmvnorm(x, mu2, sigma2)}
> Sigma <- function(x1, x2) {x <- cbind(x1, x2);
+          0.1 + (dmvnorm(x, mu1, sigma1) + dmvnorm(x, mu2, sigma2)) / 2}
> set.seed(1)
> n <- 500
> w1 <- runif(n)
> w2 <- runif(n)
> y <- vector()
> for (i in 1:n) y[i] <- rnorm(1, mean = mu(w1[i], w2[i]),
+                              sd = sqrt(Sigma(w1[i], w2[i])))
> data <- data.frame(y, w1, w2)
```

We fit a model with mean and variance functions specified as

$$
\mu(\boldsymbol{u}) = \beta_0 + \sum_{j_1=1}^{12}\sum_{j_2=1}^{12} \beta_{j_1,j_2}\phi_{1j_1,j_2}(\boldsymbol{u}), \quad \log(\sigma^2(\boldsymbol{u})) = \alpha_0 + \sum_{j_1=1}^{12}\sum_{j_2=1}^{12} \alpha_{j_1,j_2}\phi_{2j_1,j_2}(\boldsymbol{u}).
$$

The R code that fits the above model is

```
> Model <- y ~ sm(w1, w2, k = 10, bs = "rd") | sm(w1, w2, k = 10, bs = "rd")
> m2 <- mvrm(formula = Model, data = data, sweeps = 10000, burn = 5000, thin = 2,
+            seed = 1, StorageDir = DIR)
```

As in the univariate case, convergence assessment and univariate posterior summaries may be obtained by using function `mvrm2mcmc` in conjunction with functions `plot.mcmc` and `summary.mcmc`. Further, summaries of the `mvrm` fits may be obtained using functions `print.mvrm` and `summary.mvrm`. Plots of the bivariate effects may be obtained using function `plot.mvrm`. This is shown below, where argument `plotOptions` utilizes the package **colorspace** (Zeileis et al., 2009).

```
> plot(x = m2, model = "mean", term = "sm(w1,w2)", static = TRUE,
```

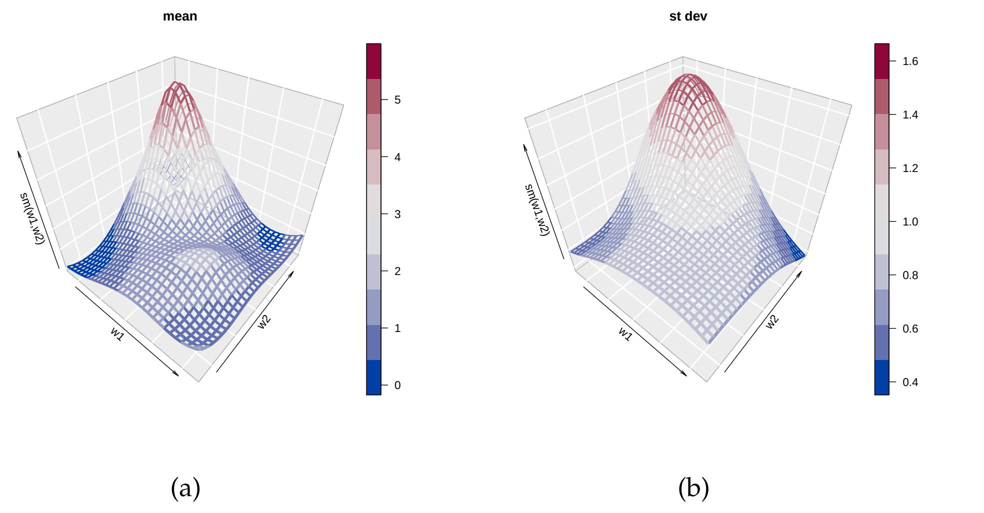


Figure 4: Bivariate simulation study results with two continuous covariates. Shown are posterior means of (a) the mean and (b) the standard deviation function.

```
+      plotOptions = list(col = diverge_hcl(n = 10)))
> plot(x = m2, model = "stdev", term = "sm(w1,w2)", static = TRUE,
+      plotOptions = list(col = diverge_hcl(n = 10)))
```

Results are shown in Figure 4. For bivariate predictors, function `plot.mvrm` calls function `ribbon3D` from the package **plot3D**. Dynamic plots, viewable in a browser, can be created by replacing the default '`static=TRUE`' by '`static=FALSE`'. When the latter option is specified, function `plot.mvrm` calls the function `scatterplot3js` from the package **threejs**. Users may pass their own options to `plot.mvrm` via the `plotOptions` argument.

Multiple covariate case

We consider fitting general additive models for the mean and variance functions in a simulated example with four independent continuous covariates. In this scenario, we set $n = 1000$. Further the covariates $\boldsymbol{w} = (w_1, w_2, w_3, w_4)^\top$ are simulated independently from a uniform distribution on the unit interval. The data-generating mechanism that we consider is

$$Y(\boldsymbol{w}) \sim N(\mu(\boldsymbol{w}), \sigma^2(\boldsymbol{w})),$$
$$\mu(\boldsymbol{w}) = \sum_{j=1}^{4} \mu_j(w_j) \quad \text{and} \quad \sigma(\boldsymbol{w}) = \prod_{j=1}^{4} \sigma_j(w_j),$$

where functions $\mu_j, \sigma_j, j = 1, \ldots, 4$, are specified below:

1. $\mu_1(w_1) = 1.5w_1, \sigma_1(w_1) = \left\{N(w_1, \mu = 0.2, \sigma^2 = 0.004) + N(w_1, \mu = 0.6, \sigma^2 = 0.1)\right\}/2,$
2. $\mu_2(w_2) = \left\{N(w_2, \mu = 0.2, \sigma^2 = 0.004) + N(w_2, \mu = 0.6, \sigma^2 = 0.1)\right\}/2,$
   $\sigma_2(w_2) = 0.6 + 0.5\sin(2\pi w_2),$
3. $\mu_3(w_3) = 1 + \sin(2\pi w_3), \sigma_3(w_3) = 1.1 - w_3,$
4. $\mu_4(w_4) = -w_4, \sigma_4(w_4) = 0.2 + 1.5w_4.$

To the generated dataset we fit a model with mean and variance functions modeled as

$$\mu(\boldsymbol{w}) = \beta_0 + \sum_{k=1}^{4}\sum_{j=1}^{16} \beta_{kj}\phi_{kj}(w_k) \quad \text{and} \quad \log\{\sigma^2(\boldsymbol{w})\} = \alpha_0 + \sum_{k=1}^{4}\sum_{j=1}^{16} \alpha_{kj}\phi_{kj}(w_k).$$

Fitting the above model to the simulated data is achieved by the following R code

```
> Model <- y ~ sm(w1, k = 15, bs = "rd") + sm(w2, k = 15, bs = "rd") +
+              sm(w3, k = 15, bs = "rd") + sm(w4, k = 15, bs = "rd") |
+              sm(w1, k = 15, bs = "rd") + sm(w2, k = 15, bs = "rd") +
+              sm(w3, k = 15, bs = "rd") + sm(w4, k = 15, bs = "rd")
```

```
> m3 <- mvrm(formula = Model, data = data, sweeps = 50000, burn = 25000,
+            thin = 5, seed = 1, StorageDir = DIR)
```

By default the function sm utilizes the radial basis functions, hence there is no need to specify bs = "rd", as we did earlier, if radial basis functions are preferred over thin plate splines. Further, we have selected k = 15 for all smooth functions. However, there is no restriction to the number of knots and certainly one can select a different number of knots for each smooth function.

As discussed previously, for each term that appears in the right-hand side of the mean and variance functions, the model incorporates indicator variables that specify which basis functions are to be included and which are to be excluded from the model. For the current example, the indicator variables are denoted by γ_{kj} and $\delta_{kj}, k = 1,2,3,4, j = 1,\ldots,16$. The prior probabilities that variables are included were specified in (8) and they are specific to each term, $\pi_{\mu_k} \sim \text{Beta}(c_{\mu_k}, d_{\mu_k})$, $\pi_{\sigma_k} \sim \text{Beta}(c_{\sigma_k}, d_{\sigma_k}), k = 1,2,3,4$. The default option pi.muPrior = "Beta(1,1)" specifies that $\pi_{\mu_k} \sim \text{Beta}(1,1), k = 1,2,3,4$. Further, by setting, for example, pi.muPrior = "Beta(2,1)" we specify that $\pi_{\mu_k} \sim \text{Beta}(2,1), k = 1,2,3,4$. To specify a different Beta prior for each of the four terms, pi.muPrior will have to be specified as a vector of length four, as an example, pi.muPrior = c("Beta(1,1)","Beta(2,1)","Beta(3,1)","Beta(4,1)"). Specification of the priors for π_{σ_k} is done in a similar way, via argument pi.sigmaPrior.

We conclude this section by presenting plots of the four terms in the mean and variance models. The plots are presented in Figure 5. We provide a few details on how the plot method works in the presence of multiple terms, and how the comparison between true and estimated effects is made. Starting with the mean function, to create the relevant plots, that appear on the left panels of Figure 5, the plot method considers only the part of the mean function $\mu(\boldsymbol{u})$ that is related to the chosen term while leaving all other terms out. For instance, in the code below we choose term = "sm(u1)" and hence we plot the posterior mean and a posterior credible interval for $\sum_{j=1}^{16} \beta_{1j}\phi_{1j}(u_1)$, where the intercept β_0 is left out by option intercept = FALSE. Further, comparison is made with a centered version of the true curve, represented by the dashed (red color) line and obtained by the first three lines of code below.

```
> x1 <- seq(0, 1, length.out = 30)
> y1 <- mu1(x1)
> y1 <- y1 - mean(y1)
> PlotOptions <- list(geom_line(aes_string(x = x1, y = y1),
+                     col = 2, alpha = 0.5, lty = 2))
> plot(x = m3, model = "mean", term = "sm(w1)", plotOptions = PlotOptions,
+      intercept = FALSE, centreEffects = FALSE, quantiles = c(0.005, 1 - 0.005))
```

The plots of the four standard deviation terms are shown in the right panels of Figure 5. Again, these are created by considering only the part of the model for $\sigma(\boldsymbol{u})$ that is related to the chosen term. For instance, below we choose term = "sm(u1)". Hence, in this case the plot will present the posterior mean and a posterior credible interval for $\exp\{\sum_{j=1}^{16} \alpha_{1j}\phi_{1j}(u_1)/2\}$, where the intercept α_0 is left out by option intercept = FALSE. Option centreEffects = TRUE scales the posterior realizations of $\exp\{\sum_{j=1}^{16} \alpha_{1j}\phi_{1j}(u_1)/2\}$ before plotting them, where the scaling is done in such a way that the realized function has mean one over the range of the predictor. Further, the comparison is made with a scaled version of the true curve, where again the scaling is done to achieve mean one. This is shown below and it is in the spirit of Chan et al. (2006) who discuss the differences between the data generating mechanism and the fitted model.

```
> y1 <- stdev1(x1) / mean(stdev1(x1))
> PlotOptions <- list(geom_line(aes_string(x = x1, y = y1),
+                     col = 2, alpha = 0.5, lty = 2))
> plot(x = m3, model = "stdev", term = "sm(w1)", plotOptions = PlotOptions,
+      intercept = FALSE, centreEffects = TRUE, quantiles = c(0.025, 1 - 0.025))
```

Data analyses

In this section we present four empirical applications.

Wage and age

In the first empirical application, we analyse a dataset from Pagan and Ullah (1999) that is available in the R package **np** (Hayfield and Racine, 2008). The dataset consists of $n = 205$ observations on dependent variable logwage, the logarithm of the individual's wage, and covariate age, the individual's

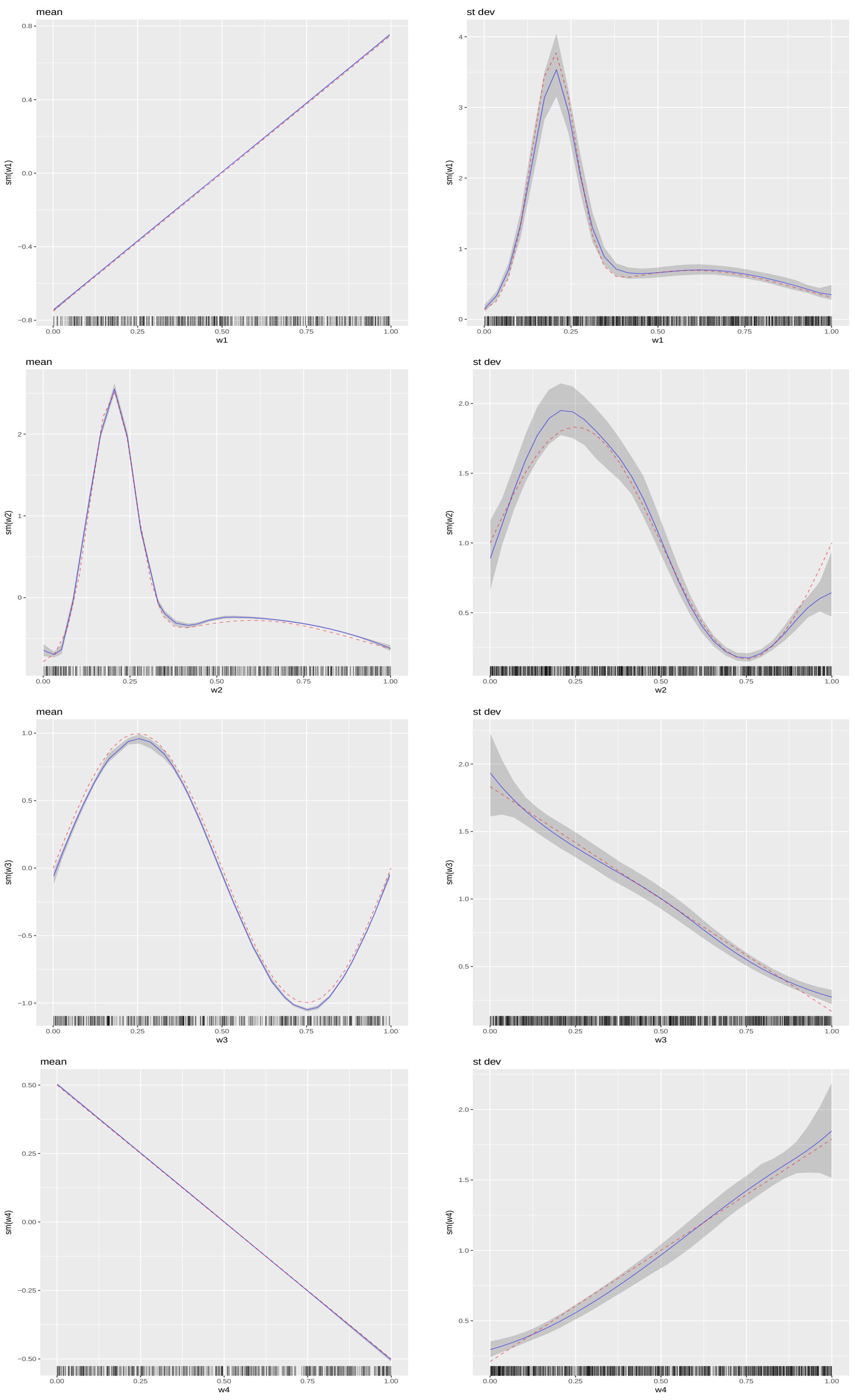


Figure 5: Multiple covariate simulation study results. The column on the left-hand side presents the true and estimated mean functions, along with 99% credible intervals. The column on the right-hand side presents the true and estimated standard deviation functions, along with 95% credible intervals. In all panels, the truth is represented by dashed (red color) lines, the estimated functions by solid (blue color) lines, and the credible intervals by gray color.

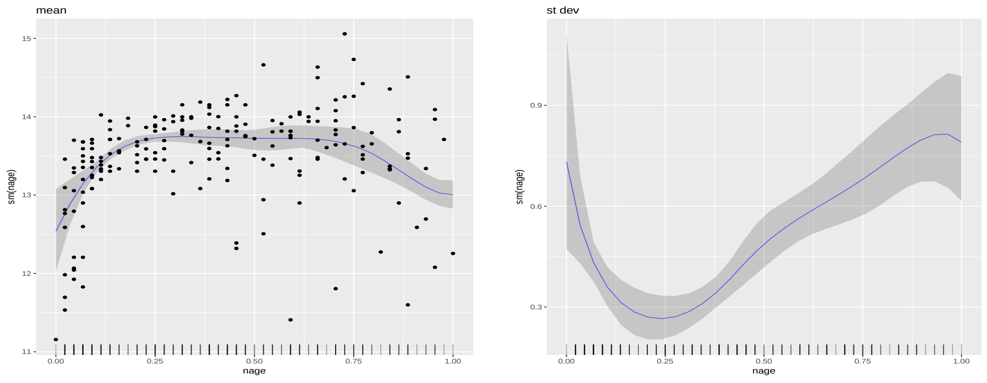


Figure 6: Results from the data analysis on the relationship between age and the logarithm of wage. Panel (a) shows the posterior mean, an 80% credible interval of the mean function, and the observed data-points. Panel (b) shows the posterior mean and an 80% credible interval of the standard deviation function.

age. The dataset comes from the 1971 Census of Canada Public Use Sample Tapes and the sampling units it involves are males of common education. Hence, the investigation of the relationship between age and the logarithm of wage is carried out controlling for the two potentially important covariates education and gender.

We utilize the following R code to specify flexible models for the mean and variance functions, and to obtain $5,000$ posterior samples, after a burn-in period of $25,000$ samples and a thinning period of 5.

```
> data(cps71)
> DIR <- getwd()
> model <- logwage ~ sm(age, k = 30, bs = "rd") | sm(age, k = 30, bs = "rd")
> m4 <- mvrm(formula = model, data = cps71, sweeps = 50000,
+            burn = 25000, thin = 5, seed = 1, StorageDir = DIR)
```

After checking convergence, we use the following code to create the plots that appear in Figure 6.

```
> wagePlotOptions <- list(geom_point(data = cps71, aes(x = age, y = logwage)))
> plot(x = m4, model = "mean", term = "sm(age)", plotOptions = wagePlotOptions)
> plot(x = m4, model = "stdev", term = "sm(age)")
```

Figure 6 (a) shows the posterior mean and an 80% credible interval for the mean function and it suggests a quadratic relationship between age and `logwage`. Figure 6 (b) shows the posterior mean and an 80% credible interval for the standard deviation function. It suggest a complex relationship between age and the variability in `logwage`. The relationship suggested by Figure 6 (b) is also suggested by the spread of the data-points around the estimated mean in Figure 6 (a). At ages around 20 years the variability in `logwage` is high. It then reduces until about age 30, to start increasing again until about age 45. From age 45 to 60 it remains stable but high, while for ages above 60, Figure 6 (b) suggests further increase in the variability, but the wide credible interval suggests high uncertainty over this age range.

Wage and multiple covariates

In the second empirical application, we analyse a dataset from Wooldridge (2008) that is also available in the R package **np**. The response variable here is the logarithm of the individual's hourly wage (`lwage`) while the covariates include the years of education (`educ`), the years of experience (`exper`), the years with the current employer (`tenure`), the individual's gender (named as `female` within the dataset, with levels `Female` and `Male`), and marital status (named as `married` with levels `Married` and `Notmarried`). The dataset consists of $n = 526$ independent observations. We analyse the first three covariates as continuous and the last two as discrete.

As the variance function is modeled in terms of an exponential, see (1), to avoid potential numerical problems, we transform the three continuous variables to have range in the interval $[0, 1]$, using

```
> data(wage1)
> wage1$ntenure <- wage1$tenure / max(wage1$tenure)
> wage1$nexper <- wage1$exper / max(wage1$exper)
> wage1$neduc <- wage1$educ / max(wage1$educ)
```

We choose to fit the following mean and variance models to the data

$$\mu_i = \beta_0 + \beta_1\, \texttt{married}_i + f_1(\texttt{ntenure}_i) + f_2(\texttt{neduc}_i) + f_3(\texttt{nexper}_i, \texttt{female}_i),$$
$$\log(\sigma_i^2) = \alpha_0 + f_4(\texttt{nexper}_i).$$

We note that, as it turns out, an interaction between variables nexper and female is not necessary for the current data analysis. However, we choose to add this term in the mean model in order to illustrate how interaction terms can be specified. We illustrate further options below.

```
> knots1 <- seq(min(wage1$nexper), max(wage1$nexper), length.out = 30)
> knots2 <- c(0, 1)
> knotsD <- expand.grid(knots1, knots2)
> model <- lwage ~ fmarried + sm(ntenure) + sm(neduc, knots=data.frame(knots =
+ seq(min(wage1$neduc), max(wage1$neduc), length.out = 15))) +
+ sm(nexper, ffemale, knots = knotsD) | sm(nexper, knots=data.frame(knots =
+ seq(min(wage1$nexper), max(wage1$nexper), length.out=15)))
```

The first three lines of the R code above specify the matrix of (potential) knots to be used for representing f_3(nexper,female). Knots may be left unspecified, in which case the defaults in function sm will be used. Furthermore, in the specification of the mean model we use sm(ntenure). By this, we chose to represent f_1 utilizing the default 10 knots and the radial basis functions. Further, the specification of f_2 in the mean model illustrates how users can specify their own knots for univariate functions. In the current example, we select 15 knots uniformly spread over the range of neduc. Fifteen knots are also used to represent f_4 within the variance model.

The following code is used to obtain samples from the posteriors of the model parameters.

```
> DIR <- getwd()
> m5 <- mvrm(formula = model, data = wage1, sweeps = 100000,
+            burn = 25000, thin = 5, seed = 1, StorageDir = DIR))
```

After summarizing results and checking convergence, we create plots of posterior means, along with 95% credible intervals, for functions $f_1, \ldots, f_4$. These are displayed in Figure 7. As it turns out, variable married does not have an effect on the mean of lwage. For this reason, we do not provide further results on the posterior of the coefficient of covariate married, β_1. However, in the code below we show how graphical summaries on β_1 can be obtained, if needed.

```
> PlotOptionsT <- list(geom_point(data = wage1, aes(x = ntenure, y = lwage)))
> plot(x = m5, model = "mean", term="sm(ntenure)", quantiles = c(0.025, 0.975),
+      plotOptions = PlotOptionsT)
> PlotOptionsEdu <- list(geom_point(data = wage1, aes(x = neduc, y = lwage)))
> plot(x = m5, model = "mean", term = "sm(neduc)", quantiles = c(0.025, 0.975),
+      plotOptions = PlotOptionsEdu)
> pchs <- as.numeric(wage1$female)
> pchs[pchs == 1] <- 17; pchs[pchs == 2] <- 19
> cols <- as.numeric(wage1$female)
> cols[cols == 2] <- 3; cols[cols == 1] <- 2
> PlotOptionsE <- list(geom_point(data = wage1, aes(x = nexper, y = lwage),
+                      col = cols, pch = pchs, group = wage1$female))
> plot(x = m5, model = "mean", term="sm(nexper,female)", quantiles = c(0.025, 0.975),
+      plotOptions = PlotOptionsE)
> plot(x = m5, model = "stdev", term = "sm(nexper)", quantiles = c(0.025, 0.975))
> PlotOptionsF <- list(geom_boxplot(fill = 2, color = 1))
> plot(x = m5, model = "mean", term = "married", quantiles = c(0.025, 0.975),
+      plotOptions = PlotOptionsF)
```

Figure 7, panels (a) and (b) show the posterior means and 95% credible intervals for f_1(ntenure) and f_2(neduc). It can be seen that expected wages increase with tenure and education, although there is high uncertainty over a large part of the range of both covariates. Panel (c) displays the posterior mean and a 95% credible interval for f_3. We can see that although the forms of the two functions are similar, (i.e. the interaction term is not needed), males have higher expected wages than females. Lastly, panel (d) displays posterior summaries of the standard deviation function, $\sigma_i = \sigma \exp(f_4/2)$. It can be seen that variability first increases and then decreases as experience increases.

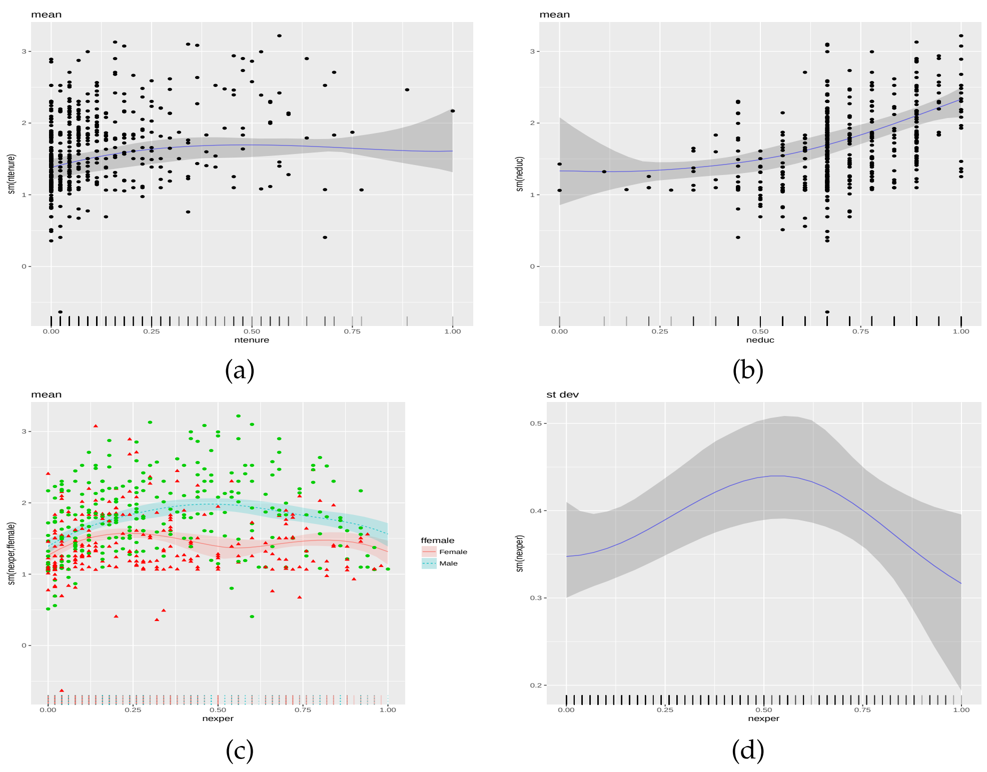


Figure 7: Results from the data analysis on the relationship between covariates gender, marital status, experience, education, and tenure, and response variable logarithm of hourly wage. Posterior means and 95% credible intervals for (a) f_1(`ntenure`), (b) f_2(`neduc`), (c) f_3(`nexper`, `female`), and (d) the standard deviation function $\sigma_i = \sigma \exp[f_4(\texttt{nexper})/2]$.

Lastly, we show how to obtain predictions and credible intervals for the levels `"Married"` and `"Notmarried"` of variable `fmaried` and the levels `"Female"` and `"Male"` of variable `fffemale`, with variables `ntenure`, `nedc`, and `nexper` fixed at their mid-range.

```
> p1 <- predict(m5, newdata = data.frame(fmarried = rep(c("Married", "Notmarried"), 2),
+        ntenure = rep(0.5, 4), neduc = rep(0.5, 4), nexper = rep(0.5, 4),
+        ffemale = rep(c("Female", "Male"), each = 2)), interval = "credible")

> p1
       fit       lwr       upr
1 1.321802 1.119508 1.506574
2 1.320400 1.119000 1.505272
3 1.913341 1.794035 2.036255
4 1.911939 1.791578 2.034832
```

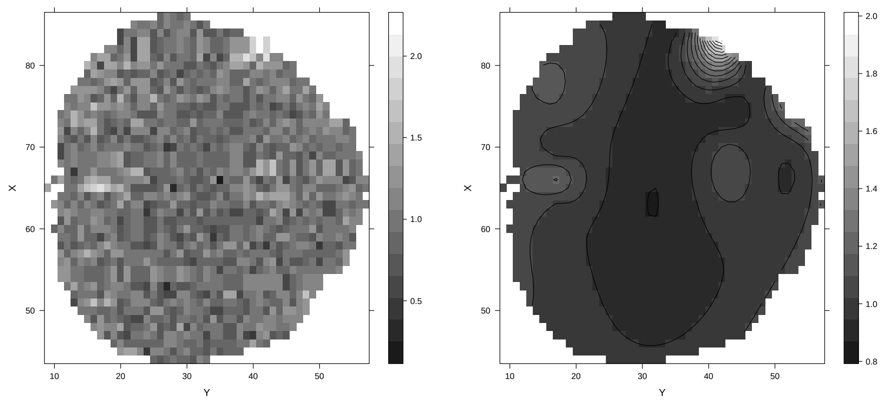
The predictions are suggestive of no "marriage" effect and of "gender" effect.

Brain activity

Here we analyse brain activity level data obtained by functional magnetic resonance imaging. The dataset is available in package **gamair** (Wood, 2006) and it was previously analysed by Landau et al. (2003). We are interested in three of the columns in the dataset. These are the response variable, `medFPQ`, which is calculated as the median over three measurements of "Fundamental Power Quotient" and the two covariates, `X` and `Y`, which show the location of each voxel.

The following R code loads the relevant data frame, removes two outliers and transforms the response variable, as was suggested by Wood (2006). In addition, it plots the brain activity data using function `levelplot` from the package **lattice** (Sarkar, 2008).

```
> data(brain)
> brain <- brain[brain$medFPQ > 5e-5, ]
> brain$medFPQ <- (brain$medFPQ) ^ 0.25
```



(a) (b)

Figure 8: Results from the brain activity level data analysis. Panel (a) shows the observed data and panel (b) the model-based smooth surface.

```
> levelplot(medFPQ ~ Y * X, data = brain, xlab = "Y", ylab = "X",
+           col.regions = gray(10 : 100 / 100))
```

The plot of the observed data is shown in Figure 8, panel (a). Its distinctive feature is the noise level, which makes it difficult to decipher any latent pattern. Hence, the goal of the current data analysis is to obtain a smooth surface of brain activity level from the noisy data. It was argued by Wood (2006) that for achieving this goal a spatial error term is not needed in the model. Thus, we analyse the brain activity level data using a model of the form

$$\text{medFPQ}_i \overset{\text{ind}}{\sim} N(\mu_i, \sigma^2), \text{ where } \mu_i = \beta_0 + \sum_{j_1=1}^{10}\sum_{j_2=1}^{10} \beta_{j_1,j_2}\phi_{1j_1,j_2}(X_i, Y_i), i = 1, \ldots, n,$$

where $n = 1565$ is the number of voxels.

The R code that fits the above model is

```
> Model <- medFPQ ~ sm(Y, X, k = 10, bs = "rd") | 1
> m6 <- mvrm(formula = Model, data = brain, sweeps = 50000, burn = 20000, thin = 2,
+            seed = 1, StorageDir = DIR)
```

From the fitted model we obtain a smooth brain activity level surface using function predict. The function estimates the average activity at each voxel of the brain. Further, we plot the estimated surface using function levelplot.

```
> p1 <- predict(m6)
> levelplot(p1[, 1] ~ Y * X, data = brain , xlab = "Y", ylab = "X",
+           col.regions = gray(10 : 100 / 100), contour = TRUE)
```

Results are shown in Figure 8, panel(b). The smooth surface makes it much easier to see and understand which parts of the brain have higher activity.

Cars

In the fourth and final application we use the function mvrm to identify the best subset of predictors in a regression setting. Usually stepwise model selection is performed, using functions step from base R and stepAIC from the **MASS** package. Here we show how mvrm can be used as an alternative to those two functions. The data frame that we apply mvrm on is mtcars, where the response variable is mpg and the explanatory variables that we consider are disp, hp, wt, and qsec. The code below loads the data frame, specifies the model and obtains samples from the posteriors of the model parameters.

```
> data(mtcars)
> Model <- mpg ~ disp + hp + wt + qsec | 1
> m7 <- mvrm(formula = Model, data = mtcars, sweeps = 50000, burn = 25000, thin = 2,
+            seed = 1, StorageDir = DIR)
```

The following is an excerpt of the output that the summary method produces showing the three models with the highest posterior probability.

```
> summary(m7, nModels = 3)

Joint mean/variance model posterior probabilities:
mean.disp mean.hp mean.wt mean.qsec freq  prob cumulative
1       0       1       1         0 1085 43.40      43.40
2       0       0       1         1 1040 41.60      85.00
3       0       0       1         0  128  5.12      90.12
Displaying 3 models of the 11 visited
3 models account for 90.12% of the posterior mass
```

The model with the highest posterior probability (43.4%) is the one that includes explanatory variables hp and wt. The model that includes wt and qsec has almost equal posterior probability, 41.6%. These two models account for 85% of the posterior mass. The third most promising model is the one that includes only wt as predictor, but its posterior probability is much lower, 5.12%.

Appendix: MCMC algorithm

In this section we present the technical details of how the MCMC algorithm is designed for the case where there is a single covariate in the mean and variance models. We first note that to improve mixing of the sampler, we integrate out vector $\boldsymbol{\beta}$ from the likelihood of $\boldsymbol{y}$, as was done by Chan et al. (2006):

$$f(\boldsymbol{y}|\boldsymbol{\alpha}, c_\beta, \gamma, \delta, \sigma^2) \propto |\sigma^2 D^2(\boldsymbol{\alpha}_\delta)|^{-\frac{1}{2}}(c_\beta+1)^{-\frac{N(\gamma)+1}{2}} \exp(-S/2\sigma^2), \tag{10}$$

where, with$\tilde{\boldsymbol{y}} = \boldsymbol{D}^{-1}(\boldsymbol{\alpha}_\delta)\boldsymbol{y}$, we have $S = S(\boldsymbol{y}, \boldsymbol{\alpha}, c_\beta, \gamma, \delta) = \tilde{\boldsymbol{y}}^\top\tilde{\boldsymbol{y}} - \frac{c_\beta}{1+c_\beta}\tilde{\boldsymbol{y}}^\top\tilde{\boldsymbol{X}}_\gamma(\tilde{\boldsymbol{X}}_\gamma^\top\tilde{\boldsymbol{X}}_\gamma)^{-1}\tilde{\boldsymbol{X}}_\gamma^\top\tilde{\boldsymbol{y}}$.

The six steps of the MCMC sampler are as follows

1. Similar to Chan et al. (2006), the elements of γ are updated in blocks of randomly chosen elements. The block size is chosen based on probabilities that can be supplied by the user or be left at their default values. Let γ_B be a block of random size of randomly chosen elements from γ. The proposed value for γ_B is obtained from its prior with the remaining elements of γ, denoted by γ_{B^C}, kept at their current value. The proposal pmf is obtained from the Bernoulli prior with π_μ integrated out

   $$p(\gamma_B|\gamma_{B^C}) = \frac{p(\gamma)}{p(\gamma_{B^C})} = \frac{\text{Beta}(c_\mu + N(\gamma), d_\mu + q_1 - N(\gamma))}{\text{Beta}(c_\mu + N(\gamma_{B^C}), d_\mu + q_1 - L(\gamma_B) - N(\gamma_{B^C}))},$$

   where $L(\gamma_B)$ denotes the length of γ_B (i.e., the size of the block). For this proposal pmf, the acceptance probability of the Metropolis-Hastings move reduces to the ratio of the likelihoods in (10)

   $$\min\left\{1, \frac{(c_\beta+1)^{-\frac{N(\boldsymbol{\gamma}^P)+1}{2}}\exp(-S^P/2\sigma^2)}{(c_\beta+1)^{-\frac{N(\boldsymbol{\gamma}^C)+1}{2}}\exp(-S^C/2\sigma^2)}\right\},$$

   where superscripts P and C denote proposed and currents values respectively.

2. Vectors $\boldsymbol{\alpha}$ and δ are updated simultaneously. Similarly to the updating of γ, the elements of δ are updated in random order in blocks of random size. Let δ_B denote a block. Blocks δ_B and the whole vector $\boldsymbol{\alpha}$ are generated simultaneously. As was mentioned by Chan et al. (2006), generating the whole vector $\boldsymbol{\alpha}$, instead of subvector $\boldsymbol{\alpha}_B$, is necessary in order to make the proposed value of $\boldsymbol{\alpha}$ consistent with the proposed value of δ.

   Generating the proposed value for δ_B is done in a similar way as was done for γ_B in the previous step. Let δ^P denote the proposed value of δ. Next, we describe how the proposed vale for $\boldsymbol{\alpha}_{\delta^P}$ is obtained. The development is in the spirit of Chan et al. (2006) who built on the work of Gamerman (1997).

   Let $\hat{\boldsymbol{\beta}}_\gamma^C = \{c_\beta/(1+c_\beta)\}(\tilde{\boldsymbol{X}}_\gamma^\top\tilde{\boldsymbol{X}}_\gamma)^{-1}\tilde{\boldsymbol{X}}_\gamma^\top\tilde{\boldsymbol{y}}$ denote the current value of the posterior mean of $\boldsymbol{\beta}_\gamma$. Define the current squared residuals

   $$e_i^C = (y_i - (\boldsymbol{x}_{i\gamma}^*)^\top\hat{\boldsymbol{\beta}}_\gamma^C)^2,$$

   $i = 1, \ldots, n$. These will have an approximate $\sigma_i^2\chi_1^2$ distribution, where $\sigma_i^2 = \sigma^2\exp(\boldsymbol{z}_i^\top\boldsymbol{\alpha})$. The latter defines a Gamma generalized linear model (GLM) for the squared residuals with mean $E(\sigma_i^2\chi_1^2) = \sigma_i^2 = \sigma^2\exp(z_i^\top\boldsymbol{\alpha})$, which, utilizing a log-link, can be thought of as Gamma GLM with an offset term: $\log(\sigma_i^2) = \log(\sigma^2) + z_i^\top\boldsymbol{\alpha}$. Given the proposed value of δ, denoted by δ^P, the proposal density for $\boldsymbol{\alpha}_{\delta^P}^P$ is derived utilizing the one step iteratively re-weighted least squares algorithm. This proceeds as follows. First define the transformed observations

$$d_i^C(\boldsymbol{\alpha}^C) = \log(\sigma^2) + z_i^\top\boldsymbol{\alpha}^C + \frac{e_i^C - (\sigma_i^2)^C}{(\sigma_i^2)^C},$$

where superscript C denotes current values. Further, let $\boldsymbol{d}^C$ denote the vector of d_i^C.
Next we define

$$\boldsymbol{\Delta}(\delta^P) = (c_\alpha^{-1}\boldsymbol{I} + \boldsymbol{Z}_{\delta^P}^\top\boldsymbol{Z}_{\delta^P})^{-1} \text{ and} \hat{\boldsymbol{\alpha}}(\delta^P, \boldsymbol{\alpha}^C) = \boldsymbol{\Delta}_{\delta^P}\boldsymbol{Z}_{\delta^P}^\top\boldsymbol{d}^C,$$

where $\boldsymbol{Z}$ is the design matrix. The proposed value $\boldsymbol{\alpha}_{\delta^P}^P$ is obtained from a multivariate normal distribution with mean$\hat{\boldsymbol{\alpha}}(\delta^P, \boldsymbol{\alpha}^C)$ and covariance $h\boldsymbol{\Delta}(\delta^P)$, denoted as $N(\boldsymbol{\alpha}_{\delta^P}^P; \hat{\boldsymbol{\alpha}}(\delta^P, \boldsymbol{\alpha}^C), h\boldsymbol{\Delta}(\delta^P))$, where h is a free parameter that we introduce and select its value adaptively (Roberts and Rosenthal, 2009) in order to achieve an acceptance probability of $20\% - 25\%$ (Roberts and Rosenthal, 2001).
Let $N(\boldsymbol{\alpha}_{\delta^C}^C; \hat{\boldsymbol{\alpha}}(\delta^C, \boldsymbol{\alpha}^P), h\boldsymbol{\Delta}(\delta^C))$ denote the proposal density for taking a step in the reverse direction, from model δ^P to δ^C. Then the acceptance probability of the pair $(\delta^P, \boldsymbol{\alpha}_{\delta^P}^P)$ is the minimum between 1 and

$$\frac{|D^2(\boldsymbol{\alpha}_{\delta^P}^P)|^{-\frac{1}{2}}\exp(-S^P/2\sigma^2)}{|D^2(\boldsymbol{\alpha}_{\delta^C}^C)|^{-\frac{1}{2}}\exp(-S^C/2\sigma^2)} \frac{(2\pi c_\alpha)^{-\frac{N(\delta^P)}{2}}\exp(-\frac{1}{2c_\alpha}(\boldsymbol{\alpha}_{\delta^P}^P)^\top\boldsymbol{\alpha}_{\delta^P}^P)}{(2\pi c_\alpha)^{-\frac{N(\delta^C)}{2}}\exp(-\frac{1}{2c_\alpha}(\boldsymbol{\alpha}_{\delta^C}^C)^\top\boldsymbol{\alpha}_{\delta^C}^C)} \frac{N(\boldsymbol{\alpha}_{\delta^C}^C; \hat{\boldsymbol{\alpha}}_{\delta^C}, h\boldsymbol{\Delta}_{\delta^C})}{N(\boldsymbol{\alpha}_{\delta^P}^P; \hat{\boldsymbol{\alpha}}_{\delta^P}, h\boldsymbol{\Delta}_{\delta^P})}.$$

We note that the determinants that appear in the above ratio are equal to one when utilizing centred explanatory variables in the variance model and hence can be left out of the calculation of the acceptance probability.

3. We update σ^2 utilizing the marginal (10) and the two priors in (7). The full conditional corresponding to the $\mathrm{IG}(a_\sigma, b_\sigma)$ prior is

$$f(\sigma^2|\dots) \propto (\sigma^2)^{-\frac{n}{2}-a_\sigma-1}\exp\{-(S/2+b_\sigma)/\sigma^2\},$$

which is recognized as another inverse gamma $\mathrm{IG}(n/2+a_\sigma, S/2+b_\sigma)$ distribution.
The full conditional corresponding to the normal prior $|\sigma| \sim N(0, \phi_\sigma^2)$ is

$$f(\sigma^2|\dots) \propto (\sigma^2)^{-\frac{n}{2}}\exp(-S/2\sigma^2)\exp(-\sigma^2/2\phi_\sigma^2).$$

Proposed values are obtained from $\sigma_p^2 \sim N(\sigma_c^2, f^2)$ where σ_c^2 denotes the current value. Proposed values are accepted with probability $f(\sigma_p^2|\dots)/f(\sigma_c^2|\dots)$. We treat f^2 as a tuning parameter and we select its value adaptively (Roberts and Rosenthal, 2009) in order to achieve an acceptance probability of $20\% - 25\%$ (Roberts and Rosenthal, 2001).

4. Parameter c_β is updated from the marginal (10) and the $\mathrm{IG}(a_\beta, b_\beta)$ prior

$$f(c_\beta|\dots) \propto (c_\beta+1)^{-\frac{N(\gamma)+1}{2}}\exp(-S/2\sigma^2)(c_\beta)^{-a_\beta-1}\exp(-b_\beta/c_\beta).$$

To sample from the above, we utilize a normal approximation to it. Let $\ell(c_\beta) = \log\{f(c_\beta|\dots)\}$. We utilize a normal proposal density $N(\hat{c}_\beta, -g^2/\ell''(\hat{c}_\beta))$, where $\hat{c}_\beta$ is the mode of $\ell(c_\beta)$, found using a Newton-Raphson algorithm, $\ell''(\hat{c}_\beta)$ is the second derivative of $\ell(c_\beta)$ evaluated at the mode, and g^2 is a tuning variance parameter the value of which is chosen adaptively (Roberts and Rosenthal, 2009). At iteration $u+1$ the acceptance probability is the minimum between one and

$$\frac{f(c_\beta^{(u+1)}|\dots)}{f(c_\beta^{(u)}|\dots)} \frac{N(c_\beta^{(u)}; \hat{c}_\beta, -g^2/\ell''(\hat{c}_\beta))}{N(c_\beta^{(u+1)}; \hat{c}_\beta, -g^2/\ell''(\hat{c}_\beta))}.$$

5. Parameter c_α is updated from the inverse Gamma density $\mathrm{IG}(a_\alpha + N(\delta)/2, b_\alpha + \boldsymbol{\alpha}_\delta^\top\boldsymbol{\alpha}_\delta/2)$.

6. The sampler utilizes the marginal in (10) to improve mixing. However, if samples are required from the posterior of $\boldsymbol{\beta}$, they can be generated from

$$\boldsymbol{\beta}_\gamma|\dots \sim N(\frac{c_\beta}{1+c_\beta}(\tilde{\boldsymbol{X}}_\gamma^\top\tilde{\boldsymbol{X}}_\gamma)^{-1}\tilde{\boldsymbol{X}}_\gamma^\top\tilde{\boldsymbol{y}}, \frac{\sigma^2 c_\beta}{1+c_\beta}(\tilde{\boldsymbol{X}}_\gamma^\top\tilde{\boldsymbol{X}}_\gamma)^{-1}),$$

where $\boldsymbol{\beta}_\gamma$ is the non-zero part of $\boldsymbol{\beta}$.

Summary

We have presented a tutorial on several functions from the R package **BNSP**. These functions are used for specifying, fitting and summarizing results from regression models with Gaussian errors and with mean and variance functions that can be modeled nonparametrically. Function `sm` is utilized to specify smooth terms in the mean and variance functions of the model. Function `mvrm` calls an MCMC algorithm that obtains samples from the posteriors of the model parameters. Samples are converted into an object of class `"mcmc"` by the function `mvrm2mcmc` which facilitates the use of multiple functions from package **coda**. Functions `print.mvrm` and `summary.mvrm` provide summaries of fitted `"mvrm"` objects. Further, function `plot.mvrm` provides graphical summaries of parametric and nonparametric terms that enter the mean or variance function. Lastly, function `predict.mvrm` provides predictions for a future response or a mean response along with the corresponding prediction/credible intervals.

Bibliography

P.-C. Bürkner. **brms**: *Bayesian Regression Models Using 'Stan'*, 2018. URL `https://CRAN.R-project.org/package=brms`. R package version 2.3.1. [p526]

D. Chan, R. Kohn, D. Nott, and C. Kirby. Locally adaptive semiparametric estimation of the mean and variance functions in regression models. *Journal of Computational and Graphical Statistics*, 15(4): 915–936, 2006. ISSN 10618600. URL `https://doi.org/10.1198/106186006X157441`. [p526, 527, 528, 530, 538, 544]

D. Gamerman. Sampling from the posterior distribution in generalized linear mixed models. *Statistics and Computing*, 7(1):57–68, 1997. ISSN 1573-1375. URL `https://doi.org/10.1023/A:1018509429360`. [p544]

E. I. George and R. E. McCulloch. Variable selection via Gibbs sampling. *Journal of the American Statistical Association*, 88(423):881–889, 1993. URL `https://doi.org/10.1080/01621459.1993.10476353`. [p526]

E. I. George and R. E. McCulloch. Approaches for Bayesian variable selection. *Statistica Sinica*, 7(2): 339–373, 1997. [p526]

T. Hayfield and J. S. Racine. Nonparametric econometrics: The np package. *Journal of Statistical Software*, 27(5), 2008. URL `http://dx.doi.org/10.18637/jss.v027.i05`. [p538]

B. Hofner, A. Mayr, N. Fenske, and M. Schmid. *gamboostLSS: Boosting Methods for GAMLSS Models*, 2018. URL `https://CRAN.R-project.org/package=gamboostLSS`. R package version 2.0-1. [p526]

S. Landau, I. C. Ellison-Wright, and E. T. Bullmore. Tests for a difference in timing of physiological response between two brain regions measured by using functional magnetic resonance imaging. *Journal of the Royal Statistical Society C*, 53(1):63–82, 2003. URL `https://doi.org/10.1111/j.0035-9254.2003.04844.x`. [p542]

B. W. Lewis. *Threejs: Interactive 3D Scatter Plots, Networks and Globes*, 2016. URL `https://CRAN.R-project.org/package=threejs`. R package version 0.2.2. [p527]

F. Liang, R. Paulo, G. Molina, M. A. Clyde, and J. O. Berger. Mixtures of g Priors for Bayesian variable selection. *Journal of the American Statistical Association*, 103(481):410–423, 2008. ISSN 0162-1459, 1537-274X. URL `https://doi.org/10.1198/016214507000001337`. [p531]

A. Mayr, N. Fenske, B. Hofner, T. Kneib, and M. Schmid. Generalized additive models for location, scale and shape for high dimensional data—a flexible approach based on boosting. *Journal of the Royal Statistical Society C*, 61(3):403–427, 2012. URL `https://doi.org/10.1111/j.1467-9876.2011.01033.x`. [p527, 535]

P. Müller and R. Mitra. Bayesian nonparametric inference — why and how. *Bayesian Analysis*, 8(2): 269–302, 2013. URL `https://doi.org/10.1214/13-BA811`. [p526]

R. B. O'Hara and M. J. Sillanpää. A review of Bayesian variable selection methods: What, how and which. *Bayesian Analysis*, 4(1):85–117, 2009. URL `https://doi.org/10.1214/09-BA403`. [p526]

A. Pagan and A. Ullah. *Nonparametric Econometrics*. Cambridge University Press, 1999. URL https://doi.org/10.1017/CBO9780511612503. [p538]

G. Papageorgiou. *BNSP: Bayesian Non- And Semi-Parametric Model Fitting*, 2018. URL https://CRAN.R-project.org/package=BNSP. R package version 2.0.7. [p526]

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 http://CRAN.R-project.org/doc/Rnews/Rnews_2006-1.pdf. [p527]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL https://www.R-project.org/. ISBN 3-900051-07-0. [p526]

R. A. Rigby and D. M. Stasinopoulos. Generalized additive models for location, scale and shape, (with discussion). *Journal of the Royal Statistical Society C*, 54(3):507–554, 2005. URL https://doi.org/10.1111/j.1467-9876.2005.00510.x. [p526]

G. O. Roberts and J. S. Rosenthal. Optimal scaling for various Metropolis-Hastings algorithms. *Statistical Science*, 16(4):351–367, 2001. URL https://doi.org/10.1214/ss/1015346320. [p545]

G. O. Roberts and J. S. Rosenthal. Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009. ISSN 1061-8600, 1537-2715. URL https://doi.org/10.1198/jcgs.2009.06134. [p527, 545]

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York, 2008. URL http://lmdvr.r-forge.r-project.org. ISBN 978-0-387-75968-5. [p542]

F. Scheipl. spikeSlabGAM: Bayesian variable selection, model choice and regularization for generalized additive mixed models in R. *Journal of Statistical Software*, 43(14):1–24, 2011. URL http://dx.doi.org/10.18637/jss.v043.i14. [p526]

F. Scheipl. *spikeSlabGAM: Bayesian Variable Selection and Model Choice for Generalized Additive Mixed Models*, 2016. URL https://CRAN.R-project.org/package=spikeSlabGAM. R package version 1.1-11. [p526]

K. Soetaert. *plot3D: Plotting Multi-Dimensional Data*, 2016. URL https://CRAN.R-project.org/package=plot3D. R package version 1.1. [p527]

D. M. Stasinopoulos and R. A. Rigby. Generalized additive models for location scale and shape (GAMLSS) in R. *Journal of Statistical Software*, 23(7):1–46, 2007. URL http://dx.doi.org/10.18637/jss.v023.i07. [p526]

N. Umlauf, N. Klein, A. Zeileis, and M. Koehler. *bamlss: Bayesian Additive Models for Location Scale and Shape (and Beyond)*, 2017. URL https://CRAN.R-project.org/package=bamlss. R package version 0.1-2. [p526]

N. Umlauf, N. Klein, and A. Zeileis. BAMLSS: Bayesian additive models for location, scale and shape (and beyond). *Journal of Computational and Graphical Statistics*, 2018. URL http://dx.doi.org/10.1080/10618600.2017.1407325. [p526]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. URL http://ggplot2.org. [p527]

S. Wood. *Mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation*, 2018. URL https://CRAN.R-project.org/package=mgcv. R package version 1.8-23. [p527]

S. N. Wood. *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC, Boca Raton, Florida, 2006. ISBN 1-58488-474-6. [p542, 543]

J. Wooldridge. *Introductory Econometrics: A Modern Approach*. South Western College, 2008. [p540]

A. Zeileis and Y. Croissant. Extended model formulas in R: Multiple parts and multiple responses. *Journal of Statistical Software*, 34(1):1–13, 2010. URL http://dx.doi.org/10.18637/jss.v034.i01. [p531]

A. Zeileis, K. Hornik, and P. Murrell. Escaping RGBland: Selecting colors for statistical graphics. *Computational Statistics & Data Analysis*, 53(9):3259–3270, 2009. URL https://doi.org/10.1016/j.csda.2008.11.033. [p536]

A. Zellner. On assessing prior distributions and Bayesian regression analysis with g-prior distributions. In P. Goel and Zellner, editors, *Bayesian Inference and Decision Techniques: Essays in Honor of Bruno De Finetti*, pages 233–243. Elsevier Science Publishers, 1986. [p528]

Georgios Papageorgiou
Department of Economics, Mathematics and Statistics
Birkbeck, University of London
g.papageorgiou@bbk.ac.uk

Measurement Errors in R

by Iñaki Ucar, Edzer Pebesma, Arturo Azcorra

Abstract This paper presents an R package to handle and represent measurements with errors in a very simple way. We briefly introduce the main concepts of metrology and propagation of uncertainty, and discuss related R packages. Building upon this, we introduce the **errors** package, which provides a class for associating uncertainty metadata, automated propagation and reporting. Working with **errors** enables transparent, lightweight, less error-prone handling and convenient representation of measurements with errors. Finally, we discuss the advantages, limitations and future work of computing with errors.

Introduction

The International Vocabulary of Metrology (VIM) defines a *quantity* as "a property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference", where most typically the number is a *quantity value*, attributed to a *measurand* and experimentally obtained via some measurement procedure, and the reference is a *measurement unit* (BIPM et al., 2012).

Additionally, any quantity value must accommodate some indication about the quality of the measurement, a quantifiable attribute known as *uncertainty*. The Guide to the Expression of Uncertainty in Measurement (GUM) defines *uncertainty* as "a parameter, associated with the result of a measurement, that characterises the dispersion of the values that could reasonably be attributed to the measurand" (BIPM et al., 2008). Uncertainty can be mainly classified into *standard uncertainty*, which is the result of a direct measurement (e.g., electrical voltage measured with a voltmeter, or current measured with a amperemeter), and *combined standard uncertainty*, which is the result of an indirect measurement (i.e., the standard uncertainty when the result is derived from a number of other quantities by the means of some mathematical relationship; e.g., electrical power as a product of voltage and current). Therefore, provided a set of quantities with known uncertainties, the process of obtaining the uncertainty of a derived measurement is called *propagation of uncertainty*.

Traditionally, computational systems have treated these three components (quantity values, measurement units and uncertainty) separately. Data consisted of bare numbers, and mathematical operations applied to them solely. Units were just metadata, and uncertainty propagation was an unpleasant task requiring additional effort and complex operations. Nowadays though, many software libraries have formalised *quantity calculus* as method of including units within the scope of mathematical operations, thus preserving dimensional correctness and protecting us from computing nonsensical combinations of quantities. However, these libraries rarely integrate uncertainty handling and propagation (Flater, 2018).

Within the R environment, the **units** package (Pebesma and Mailund, 2017; Pebesma et al., 2016) defines a class for associating unit metadata to numeric vectors, which enables transparent quantity derivation, simplification and conversion. This approach is a very comfortable way of managing units with the added advantage of eliminating an entire class of potential programming mistakes. Unfortunately, neither **units** nor any other package address the integration of uncertainties into quantity calculus.

This article presents **errors** (Ucar, 2018), a package that defines a framework for associating uncertainty metadata to R vectors, matrices and arrays, thus providing transparent, lightweight and automated propagation of uncertainty. This implementation also enables ongoing developments for integrating units and uncertainty handling into a complete solution.

Propagation of uncertainty

There are two main methods for propagation of uncertainty: the *Taylor series method* (TSM) and the *Monte Carlo method* (MCM). The TSM, also called the *delta method*, is based on a Taylor expansion of the mathematical expression that produces the output variables. As for the MCM, it can deal with generalised input distributions and propagates the uncertainty by Monte Carlo simulation.

Taylor series method

The TSM is a flexible and simple method of propagation of uncertainty that offers a good degree of approximation in most cases. In the following, we will provide a brief description. A full derivation,

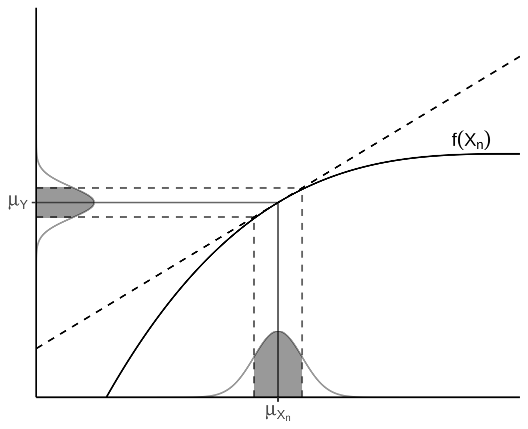


Figure 1: Illustration of linearity in an interval ± one standard deviation around the mean.

discussion and examples can be found in Arras (1998).

Mathematically, an indirect measurement is obtained as a function of n direct or indirect measurements, $Y = f(X_1, ..., X_n)$, where the distribution of X_n is unknown *a priori*. Usually, the sources of random variability are many, independent and probably unknown as well. Thus, the central limit theorem establishes that an addition of a sufficiently large number of random variables tends to a normal distribution. As a result, the **first assumption** states that X_n are normally distributed.

The **second assumption** presumes linearity, i.e., that f can be approximated by a first-order Taylor series expansion around μ_{X_n} (see Figure 1). Then, given a set of n input variables X and a set of m output variables Y, the first-order *uncertainty propagation law* establishes that

$$\Sigma_Y = J_X \Sigma_X J_X^T \tag{1}$$

where Σ is the covariance matrix and J is the Jacobian operator.

In practice, as recommended in the GUM (BIPM et al., 2008), this first-order approximation is good even if f is non-linear, provided that the non-linearity is negligible compared to the magnitude of the uncertainty, i.e., $\mathbb{E}[f(X)] \approx f(\mathbb{E}[X])$. Also, this weaker condition is distribution-free: no assumptions are needed on the probability density functions (PDF) of X_n, although they must be reasonably symmetric.

If we consider Equation (1) for pairwise computations, i.e., $Y = f(X_1, X_2)$, we can write the propagation of the uncertainty σ_Y as follows:

$$\sigma_Y^2 = \left(\frac{\partial^2 f}{\partial X_1^2}\right)^2 \sigma_{X_1}^2 + \left(\frac{\partial^2 f}{\partial X_2^2}\right)^2 \sigma_{X_2}^2 + 2\frac{\partial f \partial f}{\partial X_1 \partial X_2}\sigma_{X_1 X_2} \tag{2}$$

The cross-covariances for the output Y and any other variable Z can be simplified as follows:

$$\sigma_{YZ} = \frac{\partial f}{\partial X_1}\sigma_{X_1 Z} + \frac{\partial f}{\partial X_2}\sigma_{X_2 Z} \tag{3}$$

where, notably, if $Z = X_i$, one of the covariances above results in $\sigma_{X_i}^2$. Finally, and for the sake of completeness, the correlation coefficient can be obtained as $r_{YZ} = \sigma_{YZ}/(\sigma_Y \sigma_Z)$.

Monte Carlo method

The MCM is based on the same principles underlying the TSM. It is based on the propagation of the PDFs of the input variables X_n by performing random sampling and evaluating them under the model considered. Thus, this method is not constrained by the TSM assumptions, and explicitly determines a PDF for the output quantity Y, which makes it a more general approach that applies to a broader set of problems. For further details on this method, as well as a comparison with the TSM and some discussion on the applicability of both methods, the reader may refer to the Supplement 1 of the GUM (BIPM et al., 2008).

Reporting uncertainty

The GUM (BIPM et al., 2008) defines four ways of reporting standard uncertainty and combined standard uncertainty. For instance, if the reported quantity is assumed to be a mass m_S of nominal value 100 g:

1. $m_S = 100.02147$ g with (a combined standard uncertainty) u_c = 0.35 mg.
2. $m_S = 100.02147(35)$ g, where the number in parentheses is the numerical value of (the combined standard uncertainty) u_c referred to the corresponding last digits of the quoted result.
3. $m_S = 100.02147(0.00035)$ g, where the number in parentheses is the numerical value of (the combined standard uncertainty) u_c expressed in the unit of the quoted result.
4. $m_S = (100.02147 \pm 0.00035)$ g, where the number following the symbol $\pm$ is the numerical value of (the combined standard uncertainty) u_c and not a confidence interval.

Schemes (2, 3) and (4) would be referred to as *parenthesis* notation and *plus-minus* notation respectively throughout this document. Although (4) is a very extended notation, the GUM explicitly discourages its use to prevent confusion with confidence intervals.

Related work

Several R packages are devoted to or provide methods for propagation of uncertainties. The **car** (Fox and Weisberg, 2016, 2011) and **msm** (Jackson, 2016, 2011) packages provide the functions `deltaMethod()` and `deltamethod()` respectively. Both of them implement a first-order TSM with a similar syntax, requiring a formula, a vector of values and a covariance matrix, thus being able to deal with dependency across variables.

```
library(msm)

vals <- c(5, 1)
err <- c(0.01, 0.01)
deltamethod(~ x1 / x2, vals, diag(err**2))

#> [1] 0.0509902
```

The **metRology** (Ellison, 2017) and **propagate** (Spiess, 2014) packages stand out as very comprehensive sets of tools specifically focused on this topic, including both TSM and MCM. The **metRology** package implements TSM using algebraic or numeric differentiation, with support for correlation. It also provides a function for assessing the statistical performance of GUM uncertainty (TSM) using attained coverage probability. The **propagate** package implements TSM up to second order. It provides a unified interface for both TSM and MCM through the `propagate()` function, which requires an expression and a data frame or matrix as input. In the following example, the uncertainty of x/y is computed, where $x = 5.00(1)$ and $y = 1.00(1)$. The first result corresponds to the TSM method and the second one, to the MCM.

```
library(propagate); set.seed(42)

x <- c(5, 0.01)
y <- c(1, 0.01)
propagate(expression(x/y), data=cbind(x, y), type="stat", do.sim=TRUE)

#>    Mean.1    Mean.2      sd.1      sd.2      2.5%     97.5%
#> 5.0000000 5.0005000 0.0509902 0.0509952 4.9005511 5.1004489
#>       Mean         sd     Median        MAD       2.5%      97.5%
#> 5.00042580 0.05101762 4.99991050 0.05103023 4.90184577 5.10187805
```

Unfortunately, as in the case of **car** and **msm**, these packages are limited to work only with expressions, which does not solve the issue of requiring a separate workflow to deal with uncertainties.

The **spup** package (Sawicka and Heuvelink, 2017) focuses on uncertainty analysis in spatial environmental modelling, where the spatial cross-correlation between variables becomes important. The uncertainty is described with probability distributions and propagated using MCM.

Finally, the **distr** package (Kohl, 2017; Ruckdeschel et al., 2006) takes this idea one step further by providing an S4-based object-oriented implementation of probability distributions, with which one can operate arithmetically or apply mathematical functions. It implements all kinds of probability distributions and has more methods for computing the distribution of derived quantities. Also, **distr** is the base of a whole family of packages, such as **distrEllipse**, **distrEx**, **distrMod**, **distrRmetrics**, **distrSim** and **distrTeach**.

All these packages provide excellent tools for uncertainty analysis and propagation. However, none of them addresses the issue of an integrated workflow, as **units** does for unit metadata by assigning units directly to R vectors, matrices and arrays. As a result, units can be added to any existing R computation in a very straightforward way. On the other hand, existing tools for uncertainty propagation require building specific expressions or data structures, and then some more work to extract the results out and to report them properly, with an appropriate number of significant digits.

Automated uncertainty handling in R: the errors package

The **errors** package aims to provide easy and lightweight handling of measurement with errors, including uncertainty propagation using the first-order TSM presented in the previous section and a formally sound representation.

Package description and usage

Standard uncertainties, can be assigned to numeric vectors, matrices and arrays, and then all the mathematical and arithmetic operations are transparently applied to both the values and the associated uncertainties:

```
library(errors)

x <- 1:5 + rnorm(5, sd = 0.01)
y <- 1:5 + rnorm(5, sd = 0.02)
errors(x) <- 0.01
errors(y) <- 0.02
x; y

#> Errors: 0.01 0.01 0.01 0.01 0.01
#> [1] 0.990148 1.993813 2.995589 3.995865 5.011481

#> Errors: 0.02 0.02 0.02 0.02 0.02
#> [1] 1.014570 1.996277 2.959307 3.992147 5.013118

(z <- x / y)

#> Errors: 0.021616206 0.011190137 0.007630253 0.005605339 0.004459269
#> [1] 0.9759291 0.9987660 1.0122601 1.0009312 0.9996735
```

The `errors()` method assigns or retrieves a vector of uncertainties, which is stored as an attribute of the class `errors`, along with a unique object identifier:

```
str(x)

#>  'errors' num [1:5] 0.99(1) 1.99(1) 3.00(1) 4.00(1) 5.01(1)
#>  - attr(*, "id")=<environment: 0x55e111773cd8>
#>  - attr(*, "errors")= num [1:5] 0.01 0.01 0.01 0.01 0.01
```

Correlations (and thus covariances) between pairs of variables can be set and retrieved using the `correl()` and `covar()` methods. These correlations are stored in an internal hash table indexed by the unique object identifier assigned to each `errors` object. If an object is removed, its associated correlations are cleaned up automatically.

```
correl(x, x) # one, cannot be changed

#> [1] 1 1 1 1 1

correl(x, y) # NULL, not defined yet

#> NULL

correl(x, y) <- runif(length(x), -1, 1)
correl(x, y)
```

```
#> [1]  0.39668415 -0.03191595  0.91062883 -0.37620892  0.25118328

covar(x, y)

#> [1]  7.933683e-05 -6.383190e-06  1.821258e-04 -7.524178e-05  5.023666e-05
```

Internally, **errors** provides S3 methods for the generics belonging to the groups `Math` and `Ops`, which propagate the uncertainty and the covariance using Equations (2) and (3) respectively.

```
z # previous computation without correlations

#> Errors: 0.021616206 0.011190137 0.007630253 0.005605339 0.004459269
#> [1] 0.9759291 0.9987660 1.0122601 1.0009312 0.9996735

(z_correl <- x / y)

#> Errors: 0.017799486 0.011332199 0.004014688 0.006393033 0.003986033
#> [1] 0.9759291 0.9987660 1.0122601 1.0009312 0.9996735
```

Other many S3 methods are also provided, such as generics belonging to the `Summary` group, subsetting operators (`[`, `[<-`, `[[`, `[[<-`), concatenation (`c()`), differentiation (`diff`), row and column binding (`rbind`, `cbind`), coercion to list, data frame and matrix, and more. Such methods mutate the `errors` object, and thus return a new one with no correlations associated. There are also *setters* defined as an alternative to the assignment methods (`set_*()` instead of `errors<-`, `correl<-` and `covar<-`), primarily intended for their use in conjunction with the pipe operator (`%>%`) from the **magrittr** (Bache and Wickham, 2014) package.

Additionally, other useful summaries are provided, namely, the mean, the weighted mean and the median. The uncertainty of any measurement of central tendency cannot be smaller than the uncertainty of the individual measurements. Therefore, the uncertainty assigned to the mean is computed as the maximum between the standard deviation of the mean and the mean of the individual uncertainties (weighted, in the case of the weighted mean). As for the median, its uncertainty is computed as $\sqrt{\pi/2} \approx 1.253$ times the standard deviation of the mean, where this constant comes from the asymptotic variance formula (Hampel et al., 2011).

It is worth noting that both values and uncertainties are stored with all the digits. However, when a single measurement or a column of measurements in a data frame are printed, there are S3 methods for `format()` and `print()` defined to properly format the output and display a single significant digit for the uncertainty. This default representation can be overridden using the `digits` option, and it is globally controlled with the option `errors.digits`.

```
# the elementary charge
e <- set_errors(1.6021766208e-19, 0.0000000098e-19)
print(e, digits = 2)

#> 1.6021766208(98)e-19
```

The *parenthesis* notation, in which *the number in parentheses is the uncertainty referred to the corresponding last digits of the quantity value* (scheme 2 from the GUM, widely used in physics due to its compactness), is used by default, but this can be overridden through the appropriate option in order to use the *plus-minus* notation instead.

```
options(errors.notation = "plus-minus")
print(e, digits = 2)

#> (1.6021766208 ± 0.0000000098)e-19

options(errors.notation = "parenthesis")
```

Finally, **errors** also facilitates plotting of error bars. In the following, we first assign a 2% of uncertainty to all the numeric variables in the `iris` data set and then we plot it using base graphics and **ggplot2** (Wickham and Chang, 2016; Wickham, 2009). The result is shown in Figure 2.

```
library(dplyr)

iris.e <- iris %>%
  mutate_at(vars(-Species), funs(set_errors(., .*0.02)))

head(iris.e, 5)
```

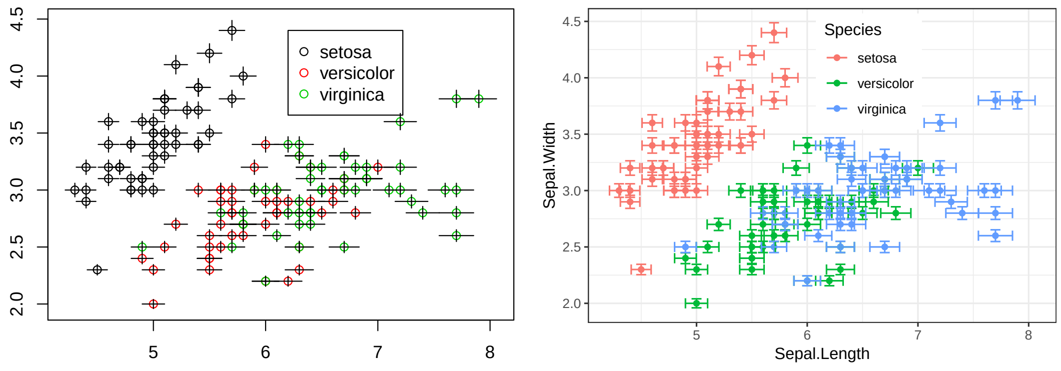


Figure 2: Base plot with error bars (left) and **ggplot2**'s version (right).

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1       5.1(1)     3.50(7)      1.40(3)    0.200(4)  setosa
#> 2       4.9(1)     3.00(6)      1.40(3)    0.200(4)  setosa
#> 3      4.70(9)     3.20(6)      1.30(3)    0.200(4)  setosa
#> 4      4.60(9)     3.10(6)      1.50(3)    0.200(4)  setosa
#> 5       5.0(1)     3.60(7)      1.40(3)    0.200(4)  setosa
```

```
plot(iris.e[["Sepal.Length"]], iris.e[["Sepal.Width"]], col=iris.e[["Species"]])
legend(6.2, 4.4, unique(iris.e[["Species"]]), col=1:length(iris.e[["Species"]]), pch=1)

library(ggplot2)

ggplot(iris.e, aes(Sepal.Length, Sepal.Width, color=Species)) +
  geom_point() + theme_bw() + theme(legend.position=c(0.6, 0.8)) +
  geom_errorbar(aes(ymin=errors_min(Sepal.Width), ymax=errors_max(Sepal.Width))) +
  geom_errorbarh(aes(xmin=errors_min(Sepal.Length), xmax=errors_max(Sepal.Length)))
```

In base graphics, the error bars are automatically plotted when an object of class `errors` is passed. Additionally, we provide the convenience functions `errors_min(x)` and `errors_max(x)` for obtaining the boundaries of the interval in **ggplot2** and other plotting packages, instead of writing `x -errors(x)` and `x + errors(x)` respectively.

Example: Simultaneous resistance and reactance measurement

From Annex H.2 of the GUM (BIPM et al., 2008):

> The resistance R and the reactance X of a circuit element are determined by measuring the amplitude V of a sinusoidally-alternating potential difference across its terminals, the amplitude I of the alternating current passing through it, and the phase-shift angle ϕ of the alternating potential difference relative to the alternating current.

The measurands (resistance R, reactance X and impedance Z) are related to the input quantities (V, I, phi) by the Ohm's law:

$$R = \frac{V}{I}\cos\phi; \qquad X = \frac{V}{I}\sin\phi; \qquad Z = \frac{V}{I} \tag{4}$$

Five simultaneous observations for each input variable are provided (Table H.2 of the GUM), which are included in **errors** as dataset `GUM.H.2`. First, we need to obtain the mean input values and set the correlations from the measurements. Then, we compute the measurands and examine the correlations between them. The results agree with those reported in the GUM:

```
V   <- mean(set_errors(GUM.H.2$V))
I   <- mean(set_errors(GUM.H.2$I))
phi <- mean(set_errors(GUM.H.2$phi))

correl(V, I)   <- with(GUM.H.2, cor(V, I))
correl(V, phi) <- with(GUM.H.2, cor(V, phi))
correl(I, phi) <- with(GUM.H.2, cor(I, phi))
```

```
print(R <- (V / I) * cos(phi), digits = 2, notation = "plus-minus")
```

```
#> 127.732 ± 0.071
```

```
print(X <- (V / I) * sin(phi), digits = 3, notation = "plus-minus")
```

```
#> 219.847 ± 0.296
```

```
print(Z <- (V / I), digits = 3, notation = "plus-minus")
```

```
#> 254.260 ± 0.236
```

```
correl(R, X); correl(R, Z); correl(X, Z)
```

```
#> [1] -0.5884298
```

```
#> [1] -0.4852592
```

```
#> [1] 0.9925116
```

Example: Calibration of a thermometer

From Annex H.3 of the GUM (BIPM et al., 2008):

> A thermometer is calibrated by comparing $n = 11$ temperature readings t_k of the thermometer [...] with corresponding known reference temperatures $t_{R,k}$ in the temperature range 21 °C to 27 °C to obtain the corrections $b_k = t_{R,k} - t_k$ to the readings.

Measured temperatures and corrections (Table H.6 of the GUM), which are included in **errors** as dataset GUM.H.3, are related by a linear calibration curve:

$$b(t) = y_1 + y_2(t - t_0) \tag{5}$$

In the following, we first fit the linear model for a reference temperature $t_0 = 20$ °C. Then, we extract the coefficients, and assign the uncertainty and correlation between them. Finally, we compute the predicted correction $b(30)$, which agrees with the value reported in the GUM:

```
fit <- lm(bk ~ I(tk - 20), data = GUM.H.3)

y1 <- set_errors(coef(fit)[1], sqrt(vcov(fit)[1, 1]))
y2 <- set_errors(coef(fit)[2], sqrt(vcov(fit)[2, 2]))
covar(y1, y2) <- vcov(fit)[1, 2]

print(b.30 <- y1 + y2 * set_errors(30 - 20), digits = 2, notation = "plus-minus")
```

```
#> -0.1494 ± 0.0041
```

Discussion

The **errors** package provides the means for defining numeric vectors, matrices and arrays with errors in R, as well as to operate with them in a transparent way. Propagation of uncertainty implements the commonly used first-order TSM formula from Equation (1). This method has been pre-computed and expanded for each operation in the S3 groups Math and Ops, instead of differentiating symbolic expressions on demand or using functions from other packages for this task. The advantages of this approach are twofold. On the one hand, it is faster, as it does not involve simulation nor symbolic computation, and very lightweight in terms of package dependencies.

Another advantage of **errors** is the built-in consistent and formally sound representation of measurements with errors, rounding the uncertainty to one significant digit by default and supporting two widely used notations: *parenthesis* (e.g., $5.00(1)$) and *plus-minus* (e.g., 5.00 ± 0.01). These notations are applied for single numbers and data frames, as well as tbl_df data frames from the **tibble** (Wickham et al., 2017) package.

Full support is provided for both data.frame and tbl_df, as well as matrices and arrays. However, some operations on those data structures may drop uncertainties (i.e., object class and attributes). More specifically, there are six common *data wrangling* operations: row subsetting, row ordering, column

Table 1: Compatibility of **errors** and R base data wrangling functions.

Operation	R base function(s)	Compatibility
Row subsetting	`[`, `[[`, `subset`	Full
Row ordering	`order` + `[`	Full
Column transformation	`transform`, `within`	Full
Row aggregation	`tapply`, `by`, `aggregate`	with `simplify=FALSE`
Column joining	`merge`	Full
(Un)Pivoting	`reshape`	Full

transformation, row aggregation, column joining and (un)pivoting. Table 1 shows the correspondence between these operations and R base functions, as well as the compatibility with **errors**.

Overall, **errors** is fully compatible with data wrangling operations embed in R base, and this is because those functions are mainly based on the subsetting generics, for which **errors** provides the corresponding S3 methods. Nonetheless, special attention must be paid to aggregations, which store partial results in lists that are finally simplified. Such simplification is made with `unlist`, which drops all the input attributes, including custom classes. However, all these aggregation functions provide the argument `simplify` (sometimes `SIMPLIFY`), which if set to `FALSE`, prevents this destructive simplification, and lists are returned. Such lists can be simplified *non-destructively* by calling `do.call(c,...)`.

```
unlist <- function(x) if (is.list(x)) do.call(c, x) else x
iris.e.agg <- aggregate(. ~ Species, data = iris.e, mean, simplify=FALSE)
as.data.frame(lapply(iris.e.agg, unlist), col.names=colnames(iris.e.agg))
```

```
#>      Species Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1     setosa       5.0(1)     3.43(7)      1.46(3)     0.25(1)
#> 2 versicolor       5.9(1)     2.77(6)      4.26(9)     1.33(3)
#> 3  virginica       6.6(1)     2.97(6)       5.6(1)     2.03(4)
```

Summary and future work

We have introduced **errors**, a lightweight R package for managing numeric data with associated standard uncertainties. The new class `errors` provides numeric operations with automated propagation of uncertainty through a first-order TSM, and a formally sound representation of measurements with errors. Using this package makes the process of computing indirect measurements easier and less error-prone.

Future work includes importing and exporting data with uncertainties and providing the user with an interface for plugging uncertainty propagation methods from other packages. Finally, **errors** enables ongoing developments for integrating **units** and uncertainty handling into a complete solution for quantity calculus. Having a unified workflow for managing measurements with units and errors would be an interesting addition to the R ecosystem with very few precedents in other programming languages.

Bibliography

K. O. Arras. An introduction to error propagation: Derivation, meaning and examples of cy = fx cx fx'. Technical report, ETH Zurich, 1998. Technical Report Nr. EPFL-ASL-TR-98-01 R3, Autonomous Systems Lab, EPFL, September 1998. [p550]

S. M. Bache and H. Wickham. *Magrittr: A Forward-Pipe Operator for R*, 2014. URL `https://CRAN.R-project.org/package=magrittr`. R package version 1.5. [p553]

BIPM, IEC, IFCC, ILAC, IUPAC, IUPAP, ISO, and OIML. Evaluation of Measurement Data – Guide to the Expression of Uncertainty in Measurement, 1st Edn. JCGM 100:2008. *Joint Committee for Guides in Metrology*, 2008. URL `https://www.bipm.org/en/publications/guides/gum.html`. [p549, 550, 551, 554, 555]

BIPM, IEC, IFCC, ILAC, IUPAC, IUPAP, ISO, and OIML. The International Vocabulary of Metrology – Basic and General Concepts and Associated Terms (VIM), 3rd Edn. JCGM 200:2012. *Joint Committee for Guides in Metrology*, 2012. URL `http://www.bipm.org/vim`. [p549]

S. L. R. Ellison. *metRology: Support for Metrological Applications*, 2017. URL https://CRAN.R-project.org/package=metRology. R package version 0.9-26-2. [p551]

D. Flater. Architecture for Software-Assisted Quantity Calculus. *Computer Standards & Interfaces*, 56: 144–147, 2018. ISSN 0920-5489. URL https://doi.org/10.1016/j.csi.2017.10.002. [p549]

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. SAGE Publications, 2nd edition, 2011. ISBN 9781412975148. [p551]

J. Fox and S. Weisberg. *Car: Companion to Applied Regression*, 2016. URL https://CRAN.R-project.org/package=car. R package version 2.1-4. [p551]

F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel. *Robust Statistics: The Approach Based on Influence Functions*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2011. ISBN 9781118150689. [p553]

C. Jackson. *Msm: Multi-State Markov and Hidden Markov Models in Continuous Time*, 2016. URL https://CRAN.R-project.org/package=msm. R package version 1.6.4. [p551]

C. H. Jackson. Multi-state models for panel data: The msm package for R. *Journal of Statistical Software*, 38(8):1–29, 2011. URL https://doi.org/10.18637/jss.v038.i08. [p551]

M. Kohl. *Distr: Object Oriented Implementation of Distributions*, 2017. URL https://CRAN.R-project.org/package=distr. R package version 2.6.2. [p551]

E. Pebesma and T. Mailund. *Units: Measurement Units for R Vectors*, 2017. URL https://CRAN.R-project.org/package=units. R package version 0.4-4. [p549]

E. Pebesma, T. Mailund, and J. Hiebert. Measurement Units in R. *The R Journal*, 8(2):486–494, 2016. URL https://journal.r-project.org/archive/2016/RJ-2016-061/index.html. [p549]

P. Ruckdeschel, M. Kohl, T. Stabla, and F. Camphausen. S4 classes for distributions. *R News*, 6(2):2–6, 2006. [p551]

K. Sawicka and G. Heuvelink. *Spup: Uncertainty Propagation Analysis*, 2017. URL https://CRAN.R-project.org/package=spup. R package version 0.1-0. [p551]

A.-N. Spiess. *Propagate: Propagation of Uncertainty*, 2014. URL https://CRAN.R-project.org/package=propagate. R package version 1.0-4. [p551]

I. Ucar. *Errors: Uncertainty Propagation for R Vectors*, 2018. URL https://CRAN.R-project.org/package=errors. R package version 0.3.0. [p549]

H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. [p553]

H. Wickham and W. Chang. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2016. URL https://CRAN.R-project.org/package=ggplot2. R package version 2.2.1. [p553]

H. Wickham, R. Francois, and K. Müller. *Tibble: Simple Data Frames*, 2017. URL https://CRAN.R-project.org/package=tibble. R package version 1.3.0. [p555]

Iñaki Ucar
UC3M-Santander Big Data Institute, Universidad Carlos III de Madrid
Calle Madrid 135. 28903 Getafe (Madrid), Spain
inaki.ucar@uc3m.es

Edzer Pebesma
Institute for Geoinformatics
Heisenbergstraße 2. 48149 Münster, Germany
edzer.pebesma@uni-muenster.de

Arturo Azcorra
Universidad Carlos III de Madrid and *IMDEA Networks Institute*
Avda. de la Universidad, 30. 28911 Leganés (Madrid), Spain
azcorra@it.uc3m.es

Navigating the R Package Universe

by Julia Silge, John C. Nash, and Spencer Graves

Abstract Today, the enormous number of contributed packages available to R users outstrips any given user's ability to understand how these packages work, their relative merits, or how they are related to each other. We organized a plenary session at useR!2017 in Brussels for the R community to think through these issues and ways forward. This session considered three key points of discussion. Users can navigate the universe of R packages with (1) capabilities for directly searching for R packages, (2) guidance for which packages to use, e.g., from CRAN Task Views and other sources, and (3) access to common interfaces for alternative approaches to essentially the same problem.

Introduction

As of our writing, there are over 13,000 packages on CRAN. R users must approach this abundance of packages with effective strategies to find what they need and choose which packages to invest time in learning how to use. At useR!2017 in Brussels, we organized a plenary session on this issue, with three themes: **search**, **guidance**, and **unification**. Here, we summarize these important themes, the discussion in our community both at useR!2017 and in the intervening months, and where we can go from here.

Users need options to search R packages, perhaps the content of DESCRIPTION files, documentation files, or other components of R packages. One author (SG) has worked on the issue of searching for R functions from within R itself in the **sos** package (Graves et al., 2017). Other options have been built such as RDocumentation.org (Cornelissen, 2018).

Guidance about what package to use for any given task is available from multiple resources for users. R users can turn to long-established resources like CRAN Task Views (Zeileis, 2005), or newer options under current development such as **packagemetrics** (Firke et al., 2018) or the **CRANsearcher** RStudio addin (Krouse and Calatroni, 2017). One author (JS) organized a survey before useR about how R users learn about R packages that informed our discussion and is summarized here.

By unification, we largely mean meta-packages or wrappers, packages that call other, related packages for a common set of tasks. A unified wrapper package provides a common API through which to access many different implementations for a certain task. One author (JN) has been particularly involved in numerical optimization techniques and presented possibilities there and beyond. More generally, as revealed during breakout discussions at useR!2017 and afterward, there are opportunities to merge either packages or their functionality. Such ideas require human cooperation and some give and take in a realm where egos can take precedence over ease of use.

After our main presentation at useR!2017, we broke out into three smaller sessions focused on these themes. We are encouraged by the engaged attendance and vigorous participation from the community we experienced, and hope to use our community's enthusiasm and ideas to move forward with steps that will improve the value of the R ecosystem.

Search

There are a number of different search capabilities for R, with different characteristics and strengths. The ways to search for help using R have proliferated in much the same way that the number of packages has, with some of the same challenges. It is not clear to users what the best way to search is, and many if not most users are not even aware of the search capabilities that have been built.

The R Project for Statistical Computing's website has had a page focused on "Getting Help with R" (R Core Team, a) from early in R's history. This official overview of various help facilities recognized by the R Core Team includes functions such as `help()`, `demo()`, `apropos()`, `help.search()`, and vignettes. This search and help functionality, used from R itself, accesses only locally installed documentation. This page also currently points to resources such as the CRAN Task Views, FAQ pages, Stack Overflow, and R email lists.

Other search capabilities have been developed by the R community over the years, focused on different types of searching. The R Site Search built by Baron is one of the longest running, and can also be accessed from R itself using the `RSiteSearch()` function in **utils** (R Core Team, 2018) and the **sos** package. The `sos::findFn()` function returns a `data.frame` of class `findFn` summarizing the help pages matching a search term. These objects can be combined by union and intersection, summarized by package, and output to an Excel file with sheets giving one row for each package and for each help

Table 1: Percentage of respondents who chose each answer on survey

How do you currently discover and learn about R packages?	% of respondents
Social media such as blogs, R-bloggers, Twitter, Slack, or GitHub contacts	79.8%
General search websites such as Google and Yahoo	57.0%
Your personal network, such as colleagues and professors	41.6%
Books, textbooks, or journal articles (JSS, JOSS, R-Journal)	31.9%
Conferences, meet-ups, or seminars	24.1%
CRAN Task Views	21.8%
Email lists such as r-help, r-packages, or r-pkg-devel	15.3%
R-specific search websites such as METACRAN or Rdocumentation	11.1%
Other	4.2%
R packages built for search such as the sos package	2.2%

page sorted by package.

The rseek site of Goodman (2007) is another resource that has been available for over a decade, and searches not only documentation but also GitHub issues, social media, and more. The R Package Documentation site of Howson (2018) is a newer option that includes not only a way to search R documentation but also the ability to run R in the browser. Another full-featured and modern website for search is RDocumentation.org, which offers users the ability to contribute examples.

Other options for search capability focus not on documentation or individual functions but on packages and package descriptions. The search capability at METACRAN (Csárdi and Ooms), crantastic (Wickham and Mæland), and the RStudio addin **CRANsearcher** allow users to search the descriptions and titles of packages to find tools that fit their analysis needs.

This discussion of search options for R users is intended to be thorough but not entirely exhaustive, and to demonstrate the variety of resources available. Many search sites are not clear how to use their search capabilities (single words vs. structured queries), and the algorithms used for ranking results give disparate results. An additional point to be considered is to what extent these options for search are open source, and what effect that could have, either positive or negative. For example, the source code for both METACRAN and **CRANsearcher** is entirely open on GitHub while RDocumentation.org is maintained by DataCamp, a privately held company.

The analysis in this section is summarized in a Wikiversity article (Graves, 2017); the article is on Wikiversity so anyone can contribute to it. At useR!2017, after the large contributed session, we broke out into three smaller sessions for discussion and brainstorming. In the breakout session focused on search, some participants commented that user reviews, available on at least crantastic among these existing resources, are useful alongside search. The discussion overall led to the development of a draft proposal for improving the ability of R users to search R packages, which is available on Wikiversity; anyone can contribute to moving it closer to realization (Graves, 2018). Overall, perhaps it is time for the R Project for Statistical Computing's "Search" (R Core Team, c) page to include more updated options.

Guidance

In preparation for this session, one author (JS) ran a brief online survey in the spring of 2017 to ask R users how they currently discover and learn about R packages. The results of this survey are available in an R package **packagesurvey** (Silge, 2018) on GitHub. There were 1039 respondents to this survey, which had a single multiple select question on it, "How do you currently discover and learn about R packages?"

Responses to this survey were fielded from R email help lists, local R meetup groups, social media such as Twitter, and affinity groups such as R-Ladies. Figure 1 shows when users responded to the survey. The respondents to this survey overwhelmingly look to social media including blogs and Twitter to learn about R packages, and also make use of general search sites and their personal network.

There were helpful, insightful answers from people contributing to the "other" option. R users use Stack Overflow to learn about R packages, as well as options like CRANberries (Eddelbuettel) and crantastic, both of which have RSS feeds that users follow. Other users mentioned learning by reading code on GitHub, the R Special Interest Group mailing lists (R Core Team, b) such as `r-sig-mixed-models` and `r-sig-geo`, and other search websites including rpackages.io.

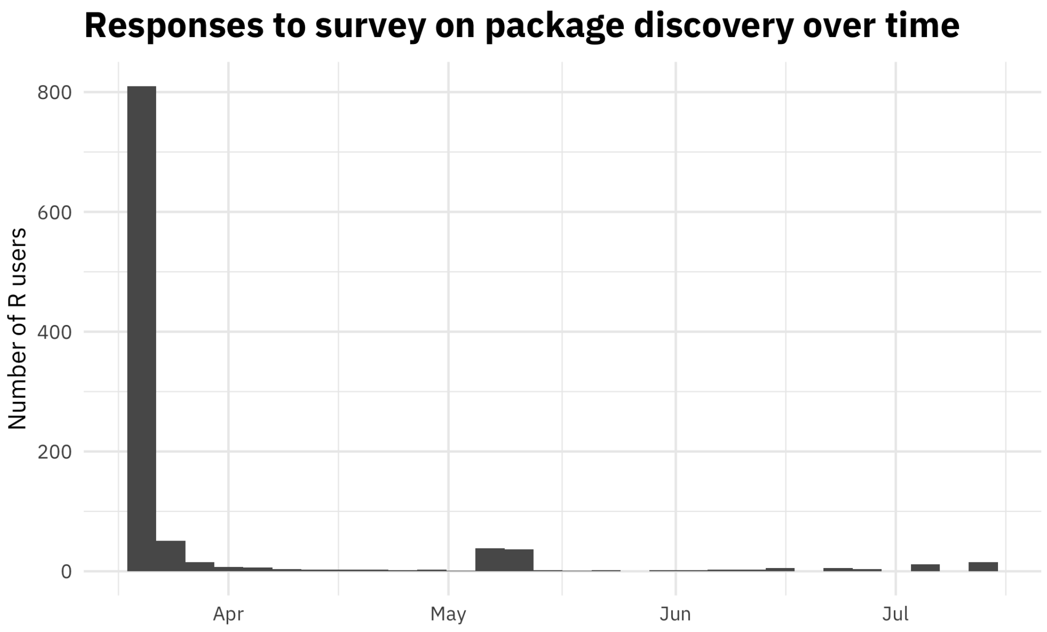


Figure 1: Responses to survey on package discovery during the spring of 2017

In the breakout session at useR!2017 focused on guidance for package choice and package evaluation, we had about 40 participants in our discussion. It was a fruitful discussion and several important themes emerged.

Value of personal impact

Participants in this session emphasized how impactful personal relationships can be in how packages are shared and evaluated. Some participants discussed how building local networks of R users may be more important in this effort than top-down, technological solutions. Our survey does show that personal recommendations have been important for many individuals in evaluating R packages. This is yet another area where local user groups can continue to have important impact. Some ways to share this experience more broadly would be online video series or live data analysis, such as those by Sean Taylor (Taylor, 2017) and Roger Peng (Peng, 2017). Learning through personal networks does not invalidate the importance of other tools like search, especially when it comes to more specialized tasks.

CRAN Task Views

Some participants wondered whether the idea of a CRAN Task View (Zeileis, 2005) is outdated in the current climate with so many packages, and whether it is even possible for one person to maintain one effectively. In fact, several participants voiced frustration with CTVs that have focused on older packages alone. Others responded that CTVs are focused on curation, which *is* still important, perhaps even more so now in a world with over 13,000 packages. We had at least one CTV maintainer present in our breakout session, and several things were presented as important in order for CTV maintainers to do their jobs:

- Package maintainers should update their `NEWS` files.
- Package maintainers need to write good documentation.

These are helpful for *all* R users, of course, but also for maintainers of CRAN Task Views. The **pkgdown** (Wickham and Hesselberth, 2018) package was mentioned as an effective option for making documentation more visible.

CRAN and *you*

Participants had several ideas about how things are done on CRAN now and adjustments that might be made in the interest of discovering and evaluating packages. One idea that came up several times was the possibility of keywords or tagging for packages. Since useR!2017, the authors have learned that there is support for some tagging architecture for packages on CRAN in the DESCRIPTION file using ACM, JEL, or MSC classifications. For an example of this in action, check out the **lfe** (Gaure, 2018) package. These are fairly unwieldy lists currently and something like an RStudio addin could be used to navigate them, if they were widely used.

Another desire participants voiced was for more information directly on CRAN, such as the number of downloads for packages. Participants also suggested that vignettes for context-specific tasks like the Bioconductor Workflows (Bioconductor) would be helpful for package discovery and evaluation, either associated with CRAN or perhaps the *R Journal*. Finally, there was some discussion about whether gate-keeping on CRAN, perceived by most to be reasonable and minimal currently, is good or bad for the community. The conclusion in our discussion was that increased editorial efforts to keep packages off CRAN would not be positive for our community.

More data, more problems

Some of the package developers at the session wondered why, when R is a data-centric language, developers have such primitive analytics about their users. Issues of user privacy are central here, but there might be opt-in options that could help both package developers and users make better decisions. The idea of a recommender system for R packages was brought up multiple times, perhaps a Tinder for R packages like papr, the Tinder for academic preprints (McGowan et al., 2017). Both the users and developers present thought that data on package use (instead of package downloads alone) would be helpful in evaluating how important or helpful R packages are. Participants also discussed the possibility of a linter for analysis scripts, similar in concept to linters for code (such as Hester (2017)) that automatically analyze code for potential problems and errors, that would suggest packages and good practice. Such a linter would necessarily be opinionated, but almost all efforts to suggest and evaluate R packages are, by definition.

Unification

Unification, as we describe it here, attempts to reduce the package count and the span of knowledge required of users. When there are many ways to carry out the same calculation, there are inevitable differences of approach. However, in many respects it is the **similarity** of approaches that causes most confusion. Very similar calling sequences, unless they are entirely compatible, lead to confusing experiences for users, and threaten the validity of results.

From the experience of one author (JN), the most satisfactory form of unification from the user perspective is the use of wrapper functions that consolidate a number of similar tools into a single calling sequence. This has been the goal of the package **optimx**, which in its 2018 incarnation consolidates several R-internal and package-based function minimization tools. Moreover, the present version subsumes a number of other packages, thereby offering a reduction in the effective package count.

A real downside of unification is the burden of work for the developers involved, including dealing with complex dependencies *and* reverse dependencies. At the time of writing, the new **optimx** fails reverse dependency checks because of new, stricter checks for CRAN policies. Embracing unification as an approach to deal with package proliferation means that such issues are a part of life and must be resolved.

While a wrapper such as **optimx** can, with effort, be created, merging two existing but different packages that provide similar capability with very different user interfaces requires human cooperation. At this time, and despite the open and collaborative R community, the level of effort to do such work is daunting. Moreover, there is a general lack of financial or other reward for such efforts.

During discussions at useR!2017, it was clear that R users are quite interested in unification of packages. Younger participants expressed the opinion that there are egos and interest groups standing in the way of some such unifications, and the status of some of the players impeded the discussion of such possibilities.

Conclusion

Our exploration of these topics leads us to call for increased respect and value for the work done by local meetup group organizers and individuals who contribute to spreading R knowledge, both online and in their communities. Our survey and discussions show how impactful these community networks are; investing in community building is not something we need do only because of idealism, but because it is effective.

We also identify the importance of growing the skills of package developers across the R ecosystem, in dependency management, documentation, and beyond. Wider adoption of best practices makes the

R package universe easier for everyone to navigate, from the developer with downstream dependencies to the CRAN Task View maintainer to the new R user.

Bibliography

J. Baron. Jonathan Baron's R Help Page. Online; accessed 18-November-18. URL http://search.r-project.org/. [p558]

Bioconductor. Bioconductor workflow packages. Online; accessed 18-November-18. URL https://www.bioconductor.org/packages/release/workflows/. [p561]

J. Cornelissen. *RDocumentation: Integrate R with 'RDocumentation'*, 2018. URL https://CRAN.R-project.org/package=RDocumentation. R package version 0.8.2. [p558]

G. Csárdi and J. Ooms. METACRAN. Online; accessed 18-November-18. URL https://www.r-pkg.org/. [p559]

D. Eddelbuettel. CRANberries. Online; accessed 18-November-18. URL http://dirk.eddelbuettel.com/cranberries/. [p559]

S. Firke, B. Krouse, E. Grand, L. Shepherd, W. Ampeh, and H. Frick. *Packagemetrics: Collect Metrics on Packages from CRAN, GitHub, and StackOverflow*, 2018. URL https://github.com/ropenscilabs/packagemetrics. R package version 0.0.1.9001. [p558]

S. Gaure. *lfe: Linear Group Fixed Effects*, 2018. URL https://CRAN.R-project.org/package=lfe. [p561]

S. Goodman. Rseek search engine. Online; accessed 18-November-18, 2007. URL https://rseek.org/. [p559]

S. Graves. Searching r packages, 2017. URL https://en.wikiversity.org/wiki/Searching_R_Packages. [p559]

S. Graves. Draft proposal for improving the ability of r users to search r packages, 2018. URL https://en.wikiversity.org/wiki/Draft_Proposal_for_improving_the_ability_of_R_users_to_search_R_packages. [p559]

S. Graves, S. Dorai-Raj, and R. Francois. *Sos: Search Contributed R Packages, Sort by Package*, 2017. URL https://CRAN.R-project.org/package=sos. R package version 2.0-0. [p558]

J. Hester. *Lintr: A 'Linter' for R Code*, 2017. URL https://CRAN.R-project.org/package=lintr. R package version 1.0.2. [p561]

I. Howson. R package documentation, 2018. URL https://rdrr.io/. [p559]

B. Krouse and A. Calatroni. *CRANsearcher: RStudio Addin for Searching Packages in CRAN Database Based on Keywords*, 2017. URL https://CRAN.R-project.org/package=CRANsearcher. R package version 1.0.0. [p558]

L. D. McGowan, N. Strayer, and J. Leek. Papr: Tinder for pre-prints, a shiny application for collecting gut-reactions to pre-prints from the scientific community. In *The R User Conference, useR! 2017 July 4-7 2017 Brussels, Belgium*, page 97, 2017. [p561]

R. Peng. Analyst mindset - episode 2. Online; accessed 18-November-18, 2017. URL https://youtu.be/jWePleDwmQo. [p560]

R Core Team. Getting Help with R. Online; accessed 18-November-18, a. URL https://www.r-project.org/help.html. [p558]

R Core Team. Mailing Lists. Online; accessed 18-November-18, b. URL https://www.r-project.org/mail.html. [p559]

R Core Team. Search. Online; accessed 18-November-18, c. URL https://www.r-project.org/search.html. [p559]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL https://www.R-project.org/. [p558]

J. Silge. *Packagesurvey: Survey of R Users on Package Discovery*, 2018. URL https://github.com/juliasilge/packagesurvey. R package version 0.1.0. [p559]

S. Taylor. Live estimating nfl statistics. Online; accessed 18-November-18, 2017. URL https://www.facebook.com/seanjtaylor/videos/10103088186201897/. [p560]

H. Wickham and J. Hesselberth. *Pkgdown: Make Static HTML Documentation for a Package*, 2018. URL https://CRAN.R-project.org/package=pkgdown. R package version 1.1.0. [p560]

H. Wickham and B. Mæland. crantastic! Online; accessed 18-November-18. URL https://www.crantastic.org/. [p559]

A. Zeileis. CRAN task views. *R News*, 5(1):39–40, 2005. URL https://CRAN.R-project.org/doc/Rnews/. [p558, 560]

Julia Silge
Stack Overflow
110 William Street, Floor 28
New York City, NY 10038
julia.silge@gmail.com

John C. Nash
University of Ottawa
Telfer School of Management
Ottawa, ON K1N 6N5, Canada
nashjc@uottawa.ca

Spencer Graves
EffectiveDefense.org
4550 Warwick Blvd. Apt. 508
Kansas City, MO 64111
spencer.graves@effectivedefense.org

Conference Report: LatinR 2018

by Laura Acion, Natalia da Silva, and Riva Quiroga

Conference summary

`LatinR <- Latin American Conference about the Use of R in Research + Development` (LatinR) was an international conference whose goal was bringing together the Latin American R community. The inagural LatinR took place at the Universidad de Palermo in Buenos Aires, Argentina, on September 3 to 5, 2018. About 100 participants from more than 10 different countries (e.g., Argentina, Uruguay, Chile, Peru, Ecuador, Brazil, Costa Rica, Venezuela, Spain, United States, Canada) attended LatinR.

LatinR will be an annual meeting that will rotate among different countries in Latin America. LatinR 2019 will be hosted by the Universidad Católica de Chile in Santiago de Chile on September 25 to 27.

Getting started

Up to now, Latin America never hosted a useR! conference. Until 2017, only Brazil, the biggest Latin American country, had some events gathering the local R community (i.e., R Day - Encontro Nacional de Usuários do R and SER - International Seminar on Statistics with R).

On October 25, 2017, an announcement was made by Heather Turner on the R User Group (RUG) Organizers Slack: "the R Foundation Conference Committee would like to see academic-focused R events in regions not currently covered by useR!" In less than a week, a group of academic Latin American R-Ladies organized their first conference call to start thinking about how this challenge could be achieved. This fast response was not the result of mere chance, but the consequence of a year in which the R community grew stronger in South America. By mid-November, everything was set up: a name, a place, a date, and a motivated, international, organizing committee.

To ease the organizational load of its first edition, LatinR 2018 was hosted within the 47th Argentinean Meetings of Informatics and Operational Research (JAIIO). JAIIO usually includes about 13 simultaneous meetings over five days and is organized by the Argentinean Informatics Society (SADIO). The 47th JAIIO was not only the organizational umbrella under which LatinR 2018 took place but also the first time JAIIO had a Code of Conduct (CoC). The CoC was requested by the R Foundation, one of LatinR endorsers, and was written by LatinR organizers for all meetings within JAIIO.

Program

LatinR had three official languages: Spanish, Portuguese, and English. It received submissions in the three languages and had presentations also in all three languages. September 3rd was dedicated to three, half-day, hands-on tutorials:

- Natalia da Silva: Static and interactive visualization with **ggplot2** and **plotly**
- Andrés Farall: Introduction to deep learning with R
- Jenny Bryan: How to repeat yourself with **purrr**

LatinR 2018 also had two outstanding plenary talks: "The Zen and the Art of Workflow Maintenance" by Jenny Bryan and "Aprender a Computar vs. Computar para Aprender" by Walter Sosa Escudero. Bryan presented the interaction between Statistics and Data Science and several practical tips to learn, improve, and maintain good data workflows with R. On the other hand, Sosa Escudero showed some examples about how teaching statistics can benefit by incorporating R programming in theoretical Statistics courses. Both plenary talks complemented each other and left a clear message about the need to embrace changes both when teaching and practicing Statistics and Data Science.

LatinR received 93 abstracts from several different countries. Only 62 abstracts were accepted, 32 of them were 15-minute oral presentations and the rest of the accepted abstracts were presented in a poster session. Figure 1 shows a bar plot with the abstract submission country distribution based on first author origin. Most of the submissions were from South America (Argentina, Uruguay, Peru, and Brazil). Nine out of the 32 oral presentations (28%) were conducted by women or other under-represented minorities (URM) in the R community. Seventeen out of the 30 posters (56%) had women or URM as first authors and out of the 8 invited talks, 6 were presented by women (75%). That is, out of all 40 oral presentations, 38% were presented by women or URM.

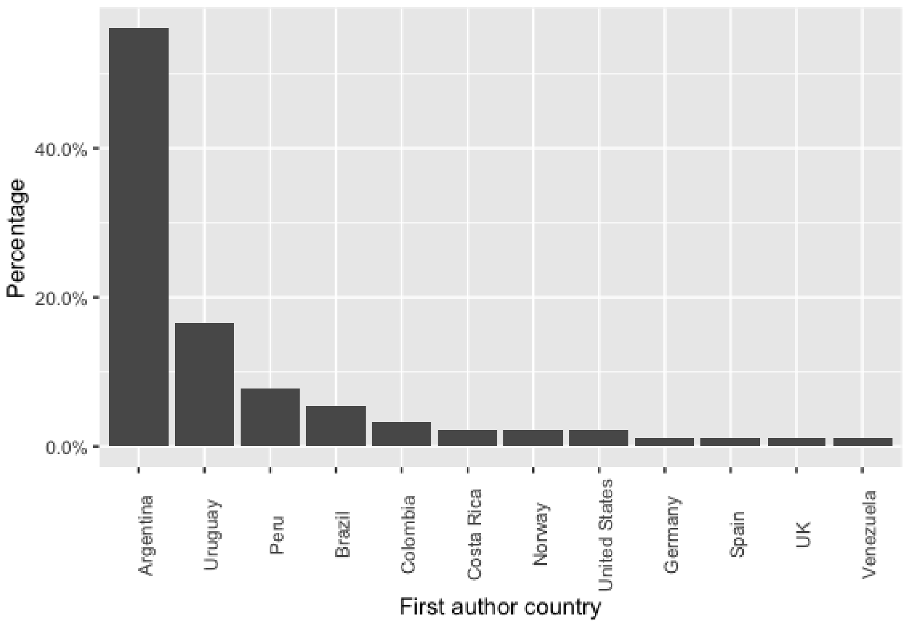


Figure 1: LatinR 2018 abstract submission distribution by first author country

Topics presented included applications of R in academic and industry settings throughout a wide number of fields such as Data Science, Statistics, Informatics, Biological and Health Sciences, Atmospheric Sciences, Social Sciences, Humanities, Economics, and Creativity. Talks and posters encompassed, among others, new R packages, innovative uses of R in Education, R visualization tools, and open data analysis with R.

The program also included presentations by members of vibrant communities such as Global and Latin American R-Ladies and The Carpentries. One of LatinR tracks during Wednesday morning was a summary tutorial about how to become a Carpentries instructor attracting 28 attendees. Wednesday was the moment of rOpenSci represented by Maëlle Salmon via teleconference and a live Q&A session led by Jenny Bryan. R user groups were also represented on Wednesday by Joshua Kunst from the Santiago de Chile RUG. The Spanish-speaking community efforts for translating R for Data Science (R4DS) (Wickham and Grolemund, 2017) was also presented by Riva Quiroga, one of the project leaders.

LatinR had a very important networking role. For instance, its three chairs, almost all its organizers, volunteers, and many community members (such as those working on the translation of R4DS) who were used to working together virtually, got to meet in person for the first time during the conference. Additionally, LatinR impulsed the community in the region in the form of at least three new RUGs that just launched in Rosario, Montevideo, and Buenos Aires.

LatinR also collaborated in a JAIIO-wide activity that debated during two hours the participation of women in sciences, technology, engineering, and math. Six women from varied backgrounds presented their views about this topic and discussed them with the attendees.

Scientific and organizing committees

LatinR was possible thanks to the effort of a highly motivated and compromised team of about 80 people including: 3 chairs (100% women from 3 Latin American countries), a 36-member scientific committee (44% women or URM; representing 8 countries around the globe), the SADIO organizing team (88% women), and a 16-member LatinR organizing committee (88% women; representing 6 Latin American countries). In addition, a highly motivated group of 15 volunteers (66% women) helped the conference run smoothly. Among all these people, Yanina Bellini Saibene, Elio Campitelli, Paola Corrales, and Florencia D'Andrea received the Chairs' Recognition Award for their outstanding and continuous contribution since the very beginning of LatinR.

Sponsors

LatinR was also possible to the following sponsors: RStudio, Fundación Sadosky, Escuelas Argentinas de Nuevas Tecnologías, R Consortium, DataCamp, and IBM Argentina.

Further information

- Video of Jenny Bryan plenary talk: https://bit.ly/2EIRaEs
- Video of Walter Sosa Escudero plenary talk: https://bit.ly/2RfwW6F
- LatinR presentations: https://github.com/LatinR/presentaciones-LatinR2018
- Twitter account: @LatinR_Conf #LatinR2018
- Webpage: http://latin-r.com

Bibliography

H. Wickham and G. Grolemund. *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2017. URL https://r4ds.had.co.nz/. ISBN-13 978-1491910399. [p565]

Laura Acion
Chair of LatinR2018
Programa de Ciencia de Datos, Fundación Sadosky
Argentina
lacion@fundacionsadosky.org.ar

Natalia da Silva
Chair of LatinR2018
Instituto de Estadística, Universidad de la República
Uruguay
natalia@iesta.edu.uy

Riva Quiroga
Chair of LatinR2018
Facultad de Letras, Pontificia Universidad Católica de Chile
Chile
riva.quiroga@uc.cl

Conference Report: SER III

by Ariel Levy, Luciane F. Alcoforado, and Orlando Celso Longo

Introduction

SER is a multidisciplinary event, which integrates professionals, students, and practioners from most diversified knowledge areas who make use of data analysis. The first edition took place on May 2016 as the initiative of a group of professors from the Fluminense Federal University, partners of other Institutions, and was supported by CAPES (Coordination for higher Education Staff Development). SER event was recognized by the R foundation (2018)[1] for its pioneering in Latin America in bringing together an expressive number of R users.

From its very first beginning, the motivation was to bring together those who wish to learn, R users, and to spread R language knowledge through the Federal Fluminense University, UFF. Organizers came up with the first event title: SER, Seminários de Estatística com R, notice that *ser* means to be in Portuguese. Soon, they discovered the borders of their ambition were too shy, as some other institutions, even from abroad, joined the project. The event name had changed but the acronym was kept.

After the resounding success of the two prior editions the third International Seminars of Statistics with R, III SER, took place, may, 22^{nd} - 24^{th}, 2018 at Federal Fluminense University - Niterói - Rio de Janeiro. In this issue the event call was: R for Science Integration Challenge. A high level event program was presented, diversified in themes, and with a remarkable feminine touch, represented in talks by Julia Silge, Jesse Maegan, Gabriela de Queiroz, Luciane Alcoforado, Becky Pattinson, Nicole Barros, Cristiane Ramos, and Karla Esquerre.

The program had 517 attendees, 357 freebies for beginners in the R basic mini-course, 27 speakers from different regions and countries, 7 authorities present among coordinators, pro-rectors, and unit directors. Fifty papers were selected by the scientific committee and presented during the oral contributor and poster sessions, in the morning of the third day, involving about 144 authors from 31 institutions.

Pre-conference Tutorials

The first day of the event occurred at the Administration Faculty in Valonguinho Campus, consisted of nine, sold out, R tutorials in diversified themes:

- Jurimetry - Julio Trecenti, ABJ;
- Importing and Wrangling Financial Data in R - Wilson Freitas, B3;
- Interactive 2D and 3D Graphics in R using OpenGL Tools - Alex Laier Bordingon, UFF;
- Dynamic Reports - Cassio Freitas, ENCE;
- My first R package - Steven Dutt Ross, UNIRIO;
- Bayesian Inference - Alexandre Silva, UNIRIO;
- Regression Models for Politomous Data - José Rodrigo de Morais, UFF;
- Web Scrapping Using **rvest** Package - Karla Esquerre & Adelmo Filho, UFBA;
- Mining and Modeling Text Using Tidy Data Principles - Julia Silge, Stack Overflow.

[1]https://rforwards-auto.github.io/blog/2018/02/05/r-in-latin-america/

The Conference

The opening day, at the Praia Vermelha Campus, began with Prof. Levy and guests in a relaxed warm-up where they discussed how they learn and teach R. After showing and commenting on several books, articles, courses, and posts on twitter and slack, the message left was: "In R we are all apprentices in different stages."

The event opening was attended by officials from UFF and ENCE. Prof. Vitor Francisco Ferreira, currently Research Pro-Rector and representative of UFF Rector, as well as authorities, congratulated Luciane Alcoforado and the other organizers for their excellent work at event organization. Prof. Fábio Barboza Passos, current director of the Engineering Faculty, reinforced the importance for a market culture change over free software usage, which is to be achieved by its dissemination over the academic community. He also pointed out that data wrangling and reports reaches all areas of knowledge in which R language stands out. He concludes that this event is to be shown as a positive result on how a relationship between at least 3 faculties within UFF and several other Universities and Institutions demonstrates that collaboration works.

The keynote opening talk was given by prof. Luciane Alcoforado, SER General Coordinator, exposing how the event organizer team has been working on the challenge of integrating the community around the dissemination of the R language. The main target audience for this project are beginners in the R language, most of them undergraduate students for whom the project mission is to embrace, and for which several courses and activities all developed in Portuguese. One of these activities, within this event issue, was a free online mini course with 400 attendees. This concluded by inviting the public to visit the event site where several articles and code bases that have been produced since the first event issue is deposited.

The following talks dealt with varied topics with strong interaction between the speakers and the audience.

Jesse Mostipak, Teaching Trust, brought up one of the biggest shortcomings for both teachers and learners of data science in R: the often unspoken prerequisite skills and content knowledge necessary to successfully apply R to data science problems. She taught us strategies to more effectively bring learners up to speed, while, for learners, how to develop strategies to identify and address their own knowledge gaps. She shared strategies from a data-driven culture that can be immediately implemented with groups of any size in order to more quickly develop data science skills in R.

R-Ladies project was the subject for Gabriela de Queiroz talk. She was so emphatic in presenting the spread throughout the world, seeking to defend and encouraging the increase of women's participation in the field of data science with the use of the R-language, that succeeded in motivating Noelle Camello in establishing the new Niterói chapter[2].

Becky Pattinson, Lancaster University and UFF, approached the ageing population problem which is faced by many countries in the world, including Brazil. Using a nationally representative sample from the 2008 PNAD, she demonstrated how they developed measurement models for the health and economic well-being of older people (aged 60+ years old). Clustering older individuals by sector, multilevel structural equation modeling provided greater understanding of the challenges for the older people of Brazil. This understanding allows for the development of policies that are efficient in the application of resources in the care of older people. Analyses were conducted using the **MplusAutomation** package in R for data formatting and estimation of the multilevel SEM in Mplus (Version 6); a commercial SEM program. In clustering individuals by sector, the model had a different structure of latent variables at the sector level. Strong associations existed between health and economic well-being and demographic variables of both individuals and sectors had significant effect on health and economic well-being.

Daniel Takata, researcher from ENCE/IBGE, discuss how to handle distributions that present heavy tail, without assumptions on the presence of moments, by introducing proce-

[2]A week after SER, we wish her success.

dures for applications in R and the **stable** package.

Gareth McCray, Keele University, followed in video conference explaining the rationale behind and the utility of Item Response Theory (IRT), a specific family of psychometric models for creating continuous score variables from binary or ordinal responses to sets of test items or questionnaire questions. IRT models have distinct advantages for score creation over other scale creation methods, e.g., Classical Test Theory (CTT). Different data structures call for different types of IRT model and the presenter briefly sketched out the landscape of this family of models. Following, he discussed how to apply the various models using various R packages.

In sequence, the most antecipated lecture of the day, Julia Silge, data scientist at Stack Overflow, with her brilliance, taught us how to manipulate, summarize, and visualize text characteristics using the methods and R packages from the tidy tool ecosystem. These tools are highly effective for many analytical questions that allow analysts to integrate natural language processing into effective workflows already in wide use. We explored how to implement approaches such as sentiment analysis of texts, measuring tf-idf, and finding word vectors.

Surprising the audience and closing the second day talks, Rodrigo Hartmann, a practitioner data scientist at Casa & Video, showed us the evolution in the use of the Shiny package with several set up details for installations and some business applications.

The activities ended with the evening of autographs, Julia Silge signing "Text Mining with R", Pedro Ferreira autographing "Analysis of Time Series in R", and Luciane Alcoforado autographing "Introduction to R using Basic Statistics."

The second day morning was reserved for the poster and the oral contributors sections.

The afternoon started with Prof. Maysa Magalhães (ENCE) as the spokesperson for the honor to the illustrious Prof. Djalma Pessoa, one of the precursors of the use of the R language in Brazil. Currently retired, Prof. Djalma Pessoa says that R is the only computational tool that he knows how to use and that has always resisted the use of other statistical programs, attracting other people to R. After his retirement he thought he would do nothing else, but today he works even harder due to the help he provides to R users.

In sequence, the contributors awards section took place, almost a tradition in our events.

The third day presentations begun with Leonardo and Jonatha, UFF Statistical students, who reported on their experience in the academic world by encouraging colleagues to study R every day, search for new packages, look at available documentation, participate in projects and events such as SER. Followed by Prof. Marcelo Perlin (UFRGS), he introduced his **GetHFData** package, which downloads and aggregates high frequency trading data for Brazilian instruments directly from B3, the Brazilian stock exchange.

Closing the III SER lectures, Prof. Karla Esquerre quickly introduced the GAMMA Group, an extension project for data analysis in UFBA. Tassio Barreto showed how to communicate with R codes through the **tidyverse** package using his passion, cinema. His presentation was permeated by movie quotes.

The event closure was made official by Prof. Orlando Longo current coordinator of the Post-Graduation Program in Civil Engineering at UFF, who highlighted the importance of new partnerships for the continuation of the next SER.

The awards

Poster Section

1. Hugo Henrique Oliveira, Adriane Caroline Portela, Denise Nunes Viola - Use of Software R as a Tool for Teaching and Learning of Combinatory Analysis.

2. Luiz Fernando Guilhem Nassif Maia, Alinne de Carvalho Veiga, Renata Souza Bueno - Brazilian Musical Genres Similarity Analysis Using Web Scraping and Text Mining with R.

3. Lucas José Gonçalves Freitas, Marcelo dos Santos Ventura - Cryptocurrency and an Application for Linear Hyperbolic Models.

Oral Contributors Section

1. Andrea Ugolini and Juan Carlos Reboredo - Multivariate Conditional Quantile Dependence Between Energy Prices and Clean Energy Stock Returns.
2. Silvio Augusto Jr. and Vinícius Basseto Félix - Brazilian Spotify Rankings Survival Analysis: Differences Between Domestic and Foreign Artists.
3. Arthur Rios de Azevedo, Anderson Ara, Mariana Yukari Noguti and Angela Ernestina Cardoso de Brito - Shiny Application: Intersection Between Gender, Class and Race in 2016 ENEM.

Testimonials

"Beyond programming and statistics, SER was a bright moment to meet and be inspired by people who make our praxis so challenging and pleasurable. This kind of connection is what makes difference for the new ones and builds a strong foundation for data science culture in Brazil" Adelmo FIlho.

"I was so happy to be introduced to the vibrant R community in Brazil via SER. It was a pleasure to meet professors, industry professionals, and students using R in their daily work and context" Julia Silge.

"Participating in SER was an absolutely joyful experience. The event was well-organized, the presentations covered a wide variety of topics, and the networking opportunities were a wonderful time to share a passion for R amongst peers" Jesse Mostipak.

"I had difficulty in the beginning, I had never installed any program in my life, I needed help but I was winning the challenge gradually, being encouraged by my daughter. I confess that I enjoyed the joke and for me it was a possibility to keep the mind active and break the routine. I am retired, I am over 70 years old, and I like challenges. The usefulness of the course for me is to show that it does not matter the age but the desire to search for the new one always" (An anonymous student of the basic course).

Organizing Committee

Orlando Celso Longo - PPGEC/UFF, Luciane Ferreira Alcoforado - PPGEC/UFF, Ariel Levy - PPGAD/UFF, José Rodrigo de Moraes - IME/UFF, Alex Laier Bordingon - IME/UFF, Manuel Febrero Bande - Un. de Santiago de Compostela/ Spain, Steven Dutt Ross - UNIRIO

Scientific Committee

Wenceslao Gonzalez Manteiga - Un. Santiago de Compostela - SP, Manuel Febrero Bande - Un. Santiago de Compostela - SP, Luís Torgo - Un. do Porto - PT, Jorge Passamani Zubelli - IMPA - BR, Orlando Celso Longo - UFF - BR, Luciane Ferreira Alcoforado - UFF - BR, Ariel Levy - UFF - BR, Steven Dutt Ross - UNIRIO - BR, Pedro Costa Ferreira - FGV/IBRE - BR, Maysa Sacramento de Magalhães - ENCE/IBGE - BR, Djalma Galvão Carneiro Pessoa - ENCE/IBGE - BR, José Rodrigo de Moraes - UFF - BR, Ludmilla da Silva Viana Jacobson - UFF - BR, Carlos Alberto Pereira Soares - UFF - BR, Assed Naked Haddad - UFRJ - BR

Further Information

Several pictures and more can be seen in our home page or in facebook.

- Conference homepage: http://www.ser.uff.br
- Facebook page: https://www.facebook.com/eventoser.uff/
- e-mail: ser.uff.br@gmail.com

Next SER

Next SER will take place in May 21-23rd 2019, we require(You).

Figure 1: SER logo.

Ariel Levy
Department of Business Administration
Federal Fluminense University
Brazil
ORCiD: 0000-0003-3557-1201
alevy@id.uff.br

Luciane F. Alcoforado
Department of Statistics
Federal Fluminense University
Brazil
ORCiD: 0000-0002-9504-8087
lucianea@id.uff.br

Orlando Celso Longo
Department of Construction Engineering
Federal Fluminense University
Brazil
ORCiD: 0000-0002-0323-473X
orlandolongo@gmail.com

Conference Report: *Why R?* 2018

by Michał Burdukiewicz, Marta Karas, Leon Eyrich Jessen, Marcin Kosiński, Bernd Bischl, and Stefan Rödiger

Figure 1: *Why R?* 2018 conference banner used for social media promotion. The background displays banks of the Odra river in Wroclaw – the city of Poland where the conference was held.

Why R? 2018 conference

The primary purpose of the *Why R?* 2018 conference was to provide R programming language enthusiasts with an opportunity to meet and discuss experiences in R software development and analysis applications, for both academia and industry professionals. The event was held 2-5 August, 2018 in a city of Wroclaw, a strong academic and business center of Poland. The total of approximately 250 people from 6 countries attended the main conference event. Additionally, approximately 540 R users attended the pre-meetings in eleven cities across Europe (Figure 2).

Why R? 2018 conference is the continuation of the *Why R?*'s first edition that took place Sep 27-29, 2017 at the Warsaw University of Technology in Warsaw (Poland). Given the success of the first event, this year's conference extended its program concept and scope; importantly, *Why R?* 2018 conference was held as international.

Conference program

The format of the conference was aimed at exposing participants to recent developments in the R language, as well as a wide range of application examples. It consisted of workshops, invited talks, field-specific series of talks, lighting-talks, special interest groups, and a full-day programming hackathon.

The conference program had a strong focus on machine learning techniques and applications, with **mlr** (Bischl et al., 2016) R package – an interface to a large number of classification and regression methods – being emphasized in a number of presentations, as well as employed during workshops and the hackathon provided by the **mlr** team. The scope of conference program included statistical methodology, data visualization, R code performance, building products based on data analyses, and R's role in academia / industry.

The event offered extensive networking opportunities. The cocktail party was held at the conference venue on the 2nd conference day. In addition, convenient location in the close proximity of the old town market square facilitated many informal gatherings that were happening each conference day.

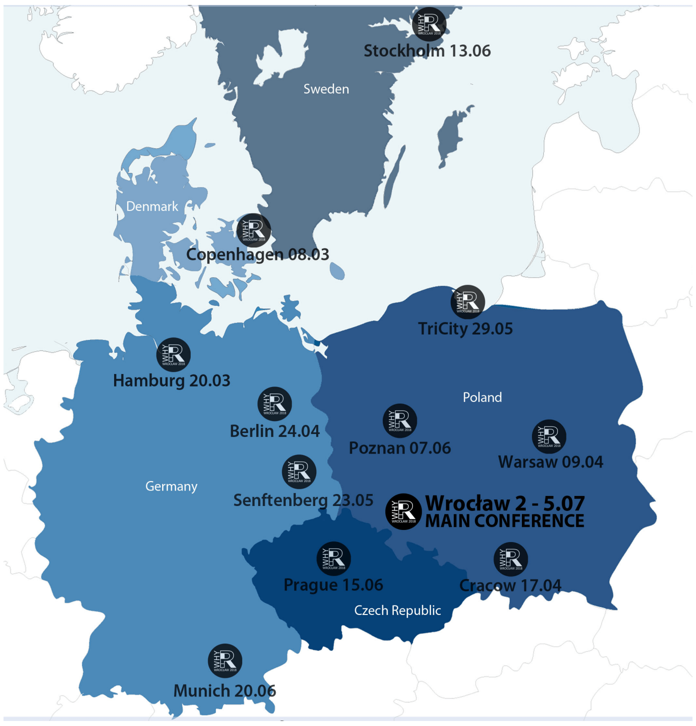


Figure 2: Locations and dates of the *Why R?* 2018 main conference event and 11 *Why R?*-branded pre-meetings.

Why R? Pre-meetings

The novel idea of pre-meetings has proved to be successful in popularizing *Why R?* conference in the international community of R users. Eleven pre-meetings took place in Czech Republic, Denmark, Germany, Poland, and Sweden in the run-up to the *Why R?* main event. The pre-meetings either constituted a part of another conference, one day-long workshop and discussion event, or a meeting of a local R user group.

As R provides a versatile framework for reproducible research in different scientific domains (Gentleman and Temple Lang, 2007; Gandrud, 2013; Leeper, 2014; Liu and Pounds, 2014; Rödiger et al., 2015), we considered the *Why R?* pre-meetings as a great opportunity to convey and popularize R as an analytics tool in groups of professionals from different fields. The pre-meeting held at International Biotechnology Innovation Days (*IBID*), an open-access conference held 23-25 May, 2018 at the Brandenburg University of Technology Cottbus - Senftenberg (Senftenberg, Germany)[1] is an example where the R came in close contact with scientist from other domains. *IBID* brought together specialists and experts in the fields of bioanalytics, biomedical and translational research, autoimmune diagnostics, digitalization, and engineering; hence it posed an excellent platform to promote R and the *Why R?* 2018 conference.

Workshops

Why R? 2018 conference had a wide portfolio of workshops:

- **Maps in R** by Piotr Sobczyk (OLX Group). Piotr showed how to create spatial data visualization efficiently in the R. He gave a plenty of tips to follow, pitfalls to avoid and a number of useful hacks. Starting from a basic plot function, he covered the usage of **ggplot2** as well as R packages that use interactive javascript libraries to prepare data reports.

[1]http://web.archive.org/web/20180701084524/https://ibid-2018.b2match.io/

- **iDash - Make your R slides awesome with xaringan** by Mikołaj Olszewski (iDash) and Mikołaj Bogucki (iDash). The workshop introduced the **xaringan** (Xie et al., 2018) package – an alternative approach to preparing a slide deck. The **xaringan** package allows customizing each slide entirely and previewing slides dynamically in RStudio; moreover, the export of the slide deck (natively in HTML) to a pixel-perfect PDF is fairly easy. As **xaringan** also uses RMarkdown, it allows for reproducible results.

- **Jumping Rivers - Shiny Basics** and **Advanced Shiny** by Roman Popat (Jumping Rivers). The instructor Roman Popat from Jumping Rivers conducted two workshops. In the first (Shiny Basics), he gave an introduction to creating interactive visualizations of data using Shiny. Here, participants learned how to use **rmarkdown** and **htmlwidgets**; input and output bindings to interact with R data structures; and input widgets and render functions to create complete page layouts using shiny and shiny dashboard. The advanced Shiny workshop explored how to add functionality to shiny apps using javascript packages and code. In particular, it was showed how one might deal with routines in a Shiny application that take a long time to run and how to provide a good experience for simultaneous users of an app. Finally, the instructor showed how to create a standalone web server API to the R code and how to integrate the use of it into a Shiny application using the **plumber** (Technology et al., 2018) package.

- **DALEX - Descriptive mAchine Learning EXplanations** by Mateusz Staniak(Uniwersytet Wrocławski). THe workshop covered tools for exploration, validation, and explanation of complex machine learning models. The packages explored in this workshop include **mlr** (Bischl et al., 2016), **DALEX** (Biecek, 2018), **live** (Staniak and Biecek, 2018), **FactorMerger** (Sitko and Biecek, 2017), **archivist** (Biecek and Kosinski, 2017), **pdp** (Greenwell, 2017) and **ALEPlot** (Apley, 2018).

- **Constructing scales from survey questions** by Tomasz Żółtak (Educational Research Institute in Warsaw, Poland). Tomasz showed how to create scales based on sets of categorical variables using Categorical Exploratory/Confirmatory Factor Analysis (CEFA / CCFA) and IRT models. He used models with bi-factor rotation to deal with different forms of asking questions and corrected for differences in a style of answering questions asked using a Likert scale. In addition, it was showed how to correct self-assessment knowledge/skill indicators using fake items.

- **From RS data to knowledge – Remote Sensing in R** by Bartłomiej Kraszewski (Forest Research Institute, Poland). Remote sensing data from different sensors is a rich source of information for studying the natural environment, natural phenomena and monitoring some extreme phenomena, such as floods. Bartłomiej presented R language packages that can be used to work with remote sensing data. These included (a) for geographic information system analysis: **rgdal** (Bivand et al., 2018a), **rgeos** (Bivand et al., 2018b) and **sf** (Pebesma et al., 2018); (b) for raster data processing: **raster** (Hijmans et al., 2017); (c) for Airborne LaserScanning data processing: the **lidR** (Roussel et al., 2018) package.

- **Introduction to Deep Learning with Keras in R** by Michał Maj (Appsilon Data Science). The workshop covered many important aspects of Deep Learning with the Keras in R, including sequential model building, performing data ingestion and using pre-trained models and performing fine-tuning. The **keras** (Allaire et al., 2018) R package was explored.

Invited talks

The invited talks topics included domain knowledge from statistics, computer science, natural sciences, and economics. The speakers list presents as follows:

- Tomasz Niedzielski (University of Wroclaw): *Forecasting streamflow using the HydroProg system developed in R*,

- Daria Szmurło (McKinsey & Company): *The age of automation – What does it mean for data scientists?*,
- Agnieszka Suchwałko (Wroclaw University of Technology): *Project evolution – from university to commerce*,
- Bernd Bischl (Ludwig-Maximilians-University of Munich): *Machine learning in R*,
- Artur Suchwałko (QuantUp): *A business view on predictive modeling: goals, assumptions, implementation*,
- Maciej Eder (Institute of Polish Language): *New advances in text mining: exploring word embeddings*,
- Thomas Petzoldt (Dresden University of Technology): *Simulation of dynamic models in R*,
- Leon Eyrich Jessen (Technical University of Denmark): *Deep Learning with R using TensorFlow*.

Special Interest Groups

Three Special Interest Groups were organized to facilitate topic-specific discussion between conference participants.

- **Diversity in Data Science**, moderated by R-Ladies Warsaw, aimed to discuss boosting the diversity of R community and inspire members of affinity groups to pursue careers in data science.
- **The Career planning in data science**, moderated by Artur Suchwałko (QuantUp) and Marcin Kosiński (Why R? Foundation), gave participants a chance to learn from experienced R enthusiasts about their career paths.
- **Teaching of data science**, moderated by Leon Eyrich Jessen (Technical University of Denmark) and Stefan Rödiger (Brandenburg Technical University Cottbus-Senftenberg), gathered data science experts from academia an industry to share their experiences and discuss challenges and solutions in teaching different concepts of data science.

Conference organizers

The quality of the scientific program of the conference was the achievement of Marcin Kosiński, Alicja Gosiewska, Aleksandra Grudziąż, Malte Grosser, Andrej-Nikolai Spiess, Przemysław Gagat, Joanna Szyda, Paweł Mackiewicz, Bartosz Sękiewicz, Przemysław Biecek, Piotr Sobczyk, Marta Karaś, Marcin Krzystanek, Marcin Łukaszewicz, Agnieszka Borsuk - De Moor, Jarosław Chilimoniuk, Michał Maj, and Michał Kurtys. The organization was in the hands of Michał Burdukiewicz (chair).

The organizers want to acknowledge R user groups from Berlin, Copenhagen, Cracow, Hamburg, Munich, Poznan, Prague, Stockholm, TriCity, Wroclaw, and Warsaw.

Acknowledgements

We would like to say thank you to all the sponsors, the University of Wrocław, Wrocław Center of Biotechnology Consortium, the local organizers of the pre-meetings, the **mlr** team, and student helpers.

Additional information

***Why R?* 2018 website** http://whyr.pl/2018
Corporate sponsors: McKinsey & Company, Wrocław Center for Biotechnology, KRUK S.A., iDash s.c., R Consortium, WLOG Solutions, Jumping Rivers Ltd., RStudio, Inc., Analyx®GmbH, and Pearson IOKI.

Bibliography

J. J. Allaire, F. Chollet, RStudio, Google, Y. Tang, D. Falbel, W. V. D. Bijl, and M. Studer. keras: R Interface to 'Keras', Apr. 2018. URL https://CRAN.R-project.org/package=keras. [p574]

D. Apley. ALEPlot: Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots, May 2018. URL https://CRAN.R-project.org/package=ALEPlot. [p574]

P. Biecek. DALEX: Descriptive mAchine Learning EXplanations, June 2018. URL https://CRAN.R-project.org/package=DALEX. [p574]

P. Biecek and M. Kosinski. archivist: An R package for managing, recording and restoring data analysis results. *Journal of Statistical Software*, 82(11):1–28, 2017. doi: 10.18637/jss.v082.i11. [p574]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL http://jmlr.org/papers/v17/15-066.html. [p572]

R. Bivand, T. Keitt, B. Rowlingson, E. Pebesma, M. Sumner, R. Hijmans, E. Rouault, F. Warmerdam, J. Ooms, and C. Rundel. rgdal: Bindings for the 'Geospatial' Data Abstraction Library, June 2018a. URL https://CRAN.R-project.org/package=rgdal. [p574]

R. Bivand, C. Rundel, E. Pebesma, R. Stuetz, K. O. Hufthammer, P. Giraudoux, M. Davis, and S. Santilli. rgeos: Interface to Geometry Engine - Open Source ('GEOS'), June 2018b. URL https://CRAN.R-project.org/package=rgeos. [p574]

C. Gandrud. *Reproducible Research with R and RStudio*. Chapman and Hall/CRC, July 2013. ISBN 978-1-4665-7284-3. [p573]

R. Gentleman and D. Temple Lang. Statistical Analyses and Reproducible Research. *Journal of Computational and Graphical Statistics*, 16(1):1–23, Mar. 2007. ISSN 1061-8600, 1537-2715. doi: 10.1198/106186007X178663. URL http://www.tandfonline.com/doi/abs/10.1198/106186007X178663. [p573]

B. M. Greenwell. pdp: An r package for constructing partial dependence plots. *The R Journal*, 9(1):421–436, 2017. URL https://journal.r-project.org/archive/2017/RJ-2017-016/index.html. [p574]

R. J. Hijmans, J. van Etten, J. Cheng, M. Mattiuzzi, M. Sumner, J. A. Greenberg, O. P. Lamigueiro, A. Bevan, E. B. Racine, A. Shortridge, and A. Ghosh. raster: Geographic Data Analysis and Modeling, Nov. 2017. URL https://CRAN.R-project.org/package=raster. [p574]

T. J. Leeper. Archiving Reproducible Research with R and Dataverse. *The R Journal*, 6(1):151–158, June 2014. URL http://journal.r-project.org/archive/2014-1/leeper.pdf. [p573]

Z. Liu and S. Pounds. An R package that automatically collects and archives details for reproducible computing. *BMC Bioinformatics*, 15(1):138, May 2014. ISSN 1471-2105. doi: 10.1186/1471-2105-15-138. URL http://www.biomedcentral.com/1471-2105/15/138/abstract. [p573]

E. Pebesma, R. Bivand, E. Racine, M. Sumner, I. Cook, T. Keitt, R. Lovelace, H. Wickham, J. Ooms, and K. Müller. sf: Simple Features for R, May 2018. URL https://CRAN.R-project.org/package=sf. [p574]

S. Rödiger, M. Burdukiewicz, K. A. Blagodatskikh, and P. Schierack. R as an Environment for the Reproducible Analysis of DNA Amplification Experiments. *The R Journal*, 7(2): 127–150, 2015. URL http://journal.r-project.org/archive/2015-1/RJ-2015-1.pdf. [p573]

J.-R. Roussel, D. A. R. the documentation), F. D. B. F. a. bugs improved catalog features), and A. S. M. I. lassnags). lidR: Airborne LiDAR Data Manipulation and Visualization for Forestry Applications, June 2018. URL https://CRAN.R-project.org/package=lidR. [p574]

A. Sitko and P. Biecek. *The Merging Path Plot: adaptive fusing of k-groups with likelihood-based model selection*, 2017. URL https://arxiv.org/abs/1709.04412. [p574]

M. Staniak and P. Biecek. Explanations of model predictions with live and breakDown packages. *ArXiv e-prints*, Apr 2018. URL https://arxiv.org/abs/1804.01955. [p574]

T. Technology, LLC, J. Allen, F. van Dunné, S. Vandewoude, and S. Software (swagger-ui). plumber: An API Generator for R, June 2018. URL https://CRAN.R-project.org/package=plumber. [p574]

Y. Xie, C. T. Ekstrøm, D. Lang, G. Aden-Buie, O. P. B. C. in rmarkdown/templates/xaringan/resources/default.css), P. Schratz, and S. Lopp. xaringan: Presentation Ninja, Feb. 2018. URL https://CRAN.R-project.org/package=xaringan. [p574]

Michał Burdukiewicz
Warsaw University of Technology, Why R? Foundation
Pl. Politechniki 1, 00-661 Warsaw
Poland
michalburdukiewicz@gmail.com

Marta Karas
Johns Hopkins Bloomberg School of Public Health
615 N Wolfe St Rm E3527
Baltimore, MD
USA
marta.karass@gmail.com

Leon Eyrich Jessen
Technical University of Denmark
Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark
Denmark
author1@work

Marcin Kosiński
Gradient Metrics, Why R? Foundation
Warsaw
Poland
m.p.kosinski@gmail.com

Bernd Bischl
Ludwig Maximilian University of Munich
Ludwigstraße 33, München

Germany
bernd_bischl@gmx.net

Stefan Rödiger
Brandenburg University of Technology Cottbus–Senftenberg
Universitätsplatz 1, Senftenberg
Germany
stefan.roediger@b-tu.de

Conference Report: R / Pharma 2018

by Joseph Rickert

The R / Pharma conference began as grass-roots initiative led by data scientists working in the pharmaceutical industry to promote the use of R in Pharma, and to establish and share best practices. The founding members organized the project as an R Consortium working group, and undertook the ambitious task of launching an annual conference envisioned as a relatively small, collegial, industry-oriented event with a strong scientific program.

Inaugural Conference

The inaugural conference, R / Pharma 2018 (`https://bit.ly/2RVKKI8`) held on August 16th and 17th at Harvard University attracted a representatives from academia, government, and industry.

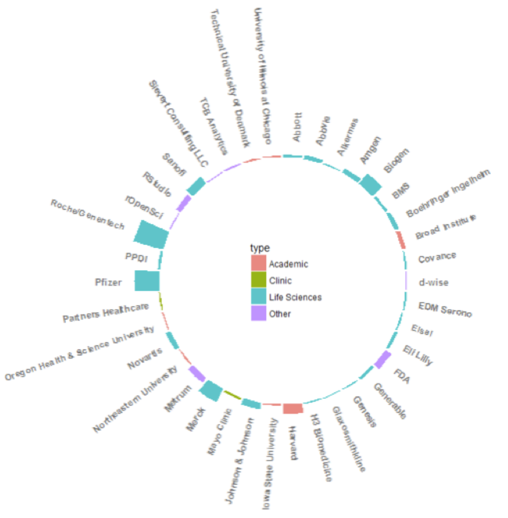


Figure 1: R / Pharma 2018 participants

Keynotes (`https://bit.ly/2SdvMNd`) were presented by Lilliam Rosario of the FDA's Center for Drug Evaluation and Research, Michael Lawrence Genentech/Roche and the R Core Group, and Max Kuhn and Joe Cheng from RStudio.

Over forty-five talks and workshops were delivered with topics ranging from reproducible research, regulatory constraints and considerations, and package administration to scaling R for production. A considerable number of talks emphasized the development of production-grade Shiny applications. Although considerably larger than most of the Shiny applications developed, Roche's 500,000 line application, which was presented at the conference and subsequently written up in a post (`https://bit.ly/2NeXKFJ`), reflects a common use case. Abstracts for the talks are available (`https://bit.ly/2CUTEeG`).

Organizing Committee

Melvin Munsaka - AbbVie; Bella Fang and Min Lee - Amgen; Eric Nantz - Eli Lilly; Elena Rantou and Paul Schuette - FDA; Elizabeth Hess - Harvard; James Black, Reinhold Koch, and Michael Lawrence - Genentech / Roche; Edward Louzier - Merck; Michael Blanks - PPD; Phil Bowsher - RStudio; and Harvey Lieberman - Sanofi.

Program Committee

Co-chairs for the program committee were Bella Feng - Amgen, John Sims - Pfizer, and Ryan Benz - SocalBioinformatics.

Program committee leads included: Melvin Munsaka - AbbVie; Robert Engle - Biogen; Elena Rantou and Paul Schuette - FDA; Elizabeth Hess - Harvard University; James Black and Reinhold Koch - Genentech/Roche; Paulo Bargo - Johnson & Johnson; Eric Nantz - Eli Lilly; Edward Lauzier - Merck; Xiao Ni- Novartis; Thomas Tensfeldt - Pfizer; and Harvey Lieberman - Sanofi.

R / Pharma 2019

R / Pharma 2019 will be held on August 21, 22, and 23, 2019 at Harvard University. In a response to attendee feedback, a third day is being planned for workshops. Please watch the conference website for more information as it becomes available.

R / Pharma 2019 conference website: http://rinpharma.com/

Twitter: #rinpharma

Joseph Rickert
RStudio
E-mail: joseph.rickert@gmail.com

Conference Report: R / Medicine Report

by Joseph Rickert, Naras Balasubramanian, and Michael Kane

R has found widespread use and is flourishing in bioinformatics, the pharmaceutical industry, clinical trials, and basic science labs. While R is being adopted in clinical informatics, its potential has not yet been realized. We believe R will fundamentally transform the space because of its strengths in making new methods available, the availability of tools for reproducible research, its interfaces to other languages, and it's ability to disseminate new approaches through web interfaces and packaging for rapid prototyping of ideas and implementations.

Moreover, while large number of clinicians, scientists, and statisticians contribute to data driven medical science, communication between individuals working in similar areas has been limited. The goals of the R Medicine working group, and the annual R / Medicine conference, are: (1) to form an international community that can help to facilitate collaboration and awareness of developments in the field; and (2) to do a better job of integrating data scientists into medical research to form more meaningful collaborations with clinicians.

R / Medicine 2018

The inaugural conference, R / Medicine 2018 (https://bit.ly/2Hzpm76), was held on September 7th and 8th at a venue just off the Yale University campus. The conference was produced by the Yale School of Public Health and the Department of Biostatistics. Sponsors included the R Consortium, Medidata Inc., The New England Statistical Society, and RStudio.

Keynotes and Talks

Four keynote addresses were delivered: Robert Tibshirani of Stanford University spoke about predicting platelet demand at the Stanford Blood Center (https://bit.ly/2B3xWFi). Victoria Stodden of the University of Illinois at Urbana-Champaign presented *Computational Reproducibility in Medical Research: Toward Open Code and Data* (https://stanford.io/2UlAqWF). Michael Lawrence of Genentech and the R Core Group presented *Scientific software in-the-large* (https://bit.ly/2RU7rg4). And. Dr. Harlan Krumholz of Yale University and the Yale-New Haven Hospital presented *Dream Crazy: Imagine the Possibilities of Data Science in Medicine*.

Additional talks ranged from tutorials on Shiny and Stan to presentations on reproducible research and tidy models, analyzing genomic data and more. For a complete list of talks refer to the conference website. A high point of the conference was the closing roundtable discussion with the theme *Bridging the Two Cultures*. This considered how the professional representing the statistical and clinical points of view may conceptualize and approach medical challenges in very different ways. For a fuller account of this discussion and the conference as a whole see https://bit.ly/2Myn5rm.

Organizing and Program Committees

The conference organizing committee consisted of: Beth Atkinson, The Mayo Clinic; Denise Esserman, Yale University; Michael Kane, Yale University (Conference Chair); Balasubramanian Narasimhan (Naras), Stanford University; Joseph Rickert, RStudio; and Hongyu Zhao, Yale University.

The program committee was composed of Denise Esserman (Program Chair), Beth Atkinson, Michael Kane, Balasubramanian Narasimhan, and Hongyu Zhao

R / Medicine 2019

R / Medicine 2019 will be held at Yale University from September 19th through September 21st. The conference goal is to grow the community and nurture the conversation about developing useful clinical research and operational solutions. The conference organizers invite clinicians, statistical researchers and others who have an interest in promoting R in clinical practice and medical research to participate however they can. We also invite interested organizations to consider sponsoring R / Medicine 2019. Please watch the website below for announcements regarding abstract submissions, sponsorship opportunities and important dates.

R / Medicine 2019 conference website: `https://r-medicine.com/`

Twitter: `#rmedicine`

For information: `r-medicine@protonmail.com`

Joseph Rickert
RStudio
E-mail: `joseph.rickert@gmail.com`

Naras Balasubramanian
Department of Statistics
Stanford University
Palo Alto, CA
URL: `naras.web.stanford.edu`

Michael Kane
Yale Center for Analytical Sciences
Yale University
New Haven, CT
E-mail: `michael.kane@yale.edu`

R Foundation News

by Torsten Hothorn

Donations and members

Membership fees and donations received between 2018-08-13 and 2019-01-07.

Donations

Pedro Albuquerque (France), Robert Baskin (United States), Shalese Fitzgerald (United States), Michael Forster (Austria), Brian Gough (United Kingdom), Gen KOBAYASHI (Japan), Ravinderpal Vaid (United States), and Novartis Pharma AG, (Basel, Switzerland)

Supporting benefactors

Jay Raol (United States) and b-data GmbH, (Zürich, Switzerland)

Supporting institutions

General Counsel Metrics, LLC, PRINCETON (United States), Marine Scotland, (Aberdeen City Aberdeen, United Kingdom), and vizGet, (Autrans, France)

Supporting members

Tim Appelhans (Germany), Joaquín Baquer-Miravete (Spain), Marcel Baumgartner (Switzerland), Daniel Booth (Australia), Petr Bouchal (Czech Republic), Florian Brezina (Germany), Robert Daly (Australia), Steph de Silva (Australia), Ajit de Silva (United States), Jasja Dekker (Netherlands), Shaban Demirel (United States), Lukman Edwindra (Indonesia), Arturo Erdely (Mexico), Michael Feyder (United States), Naohiro Furutani (Japan), Jan Galkowski (United States), Krushi Gurudu (United States), Karl Habermeier (United States), Joe Harwood (United Kingdom), Bela Hausmann (Austria), Joshua Hruzik (Germany), Abner Huertas (Guatemala), Ken Ikeda (Japan), Larry Jamner (United States), Knut Helge Jensen (Norway), Woojune Jung (Korea, Republic of), June Kee Kim (Korea, Republic of), Miha Kosmac (United Kingdom), Jan Herman Kuiper (United Kingdom), Yannick Lelu (France), Chin Soon Lim (Singapore), Michael Mahoney (United States), Daniel McNichol (United States), Guido Möser (Germany), Yoshinobu Nakahashi (Japan), Vialaneix Nathalie (France), Michael Neale (United States), Mark Niemann-Ross (United States), Berk Orbay (Turkey), Dan Orsholits (Switzerland), Mucio Osorio (Mexico), Floris Padt (Netherlands), Peter Perez (United States), Elgin Perry (United States), Que Binh Phung (France), Emma Rand (United Kingdom), Ma Reader (United Kingdom), Kun Ren (China), Marty Rose (United States), Ingo Ruczinski (United States), Antonio J. Saez-Castillo (Spain), Sarah Shakil (Canada), Jagat Sheth (United States), Kevin Shook (Canada), Rachel Smith-Hunter (United States), Tobias Strapatsas (Germany), Robert Szabo (Sweden), Koray Tascilar (Germany), Uku Vainik (Estonia), Jaime Vera (Colombia), Marcus Vollmer (Germany), Jaap Walhout (Netherlands), Sandra Ware (Australia), Nan Xiao (United States), Metin Yazici (Turkey), and Victor Zurkowski (Canada)

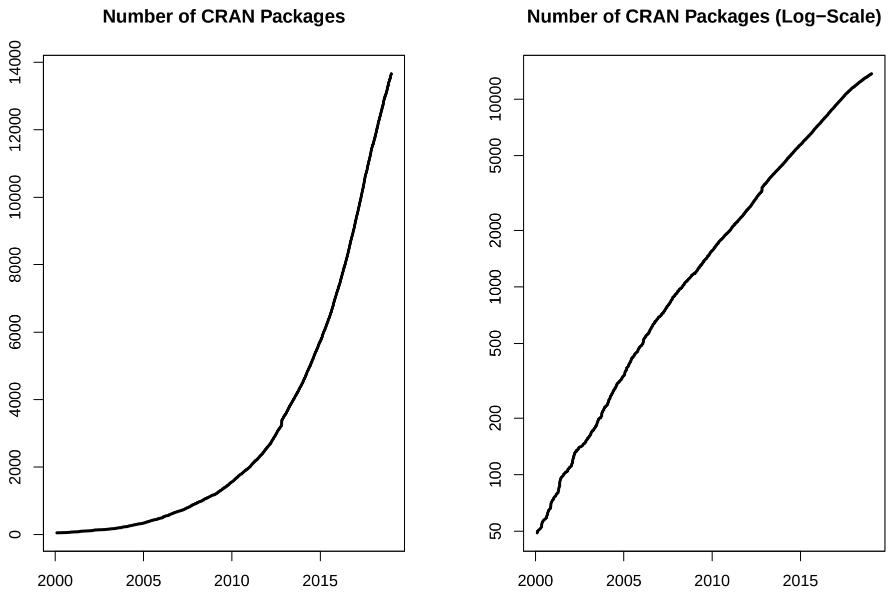
Torsten Hothorn
Universität Zürich, Switzerland
Torsten.Hothorn@R-project.org

Changes on CRAN

2018-07-01 to 2018-12-31

by Kurt Hornik, Uwe Ligges, and Achim Zeileis

In the past 6 months, 1029 new packages were added to the CRAN package repository. 68 packages were unarchived, 122 archived, and one removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2018-12-31, the number of active packages was around 13592.

Changes in the CRAN Repository Policy

The Policy now says the following:

- Packages which use Internet resources should fail gracefully with an informative message if the resource is not available (and not give a check warning nor error).

CRAN package submissions

During the last 6 months (July to December 2018), CRAN received 10259 package submissions. For these, 16830 actions took place of which 11618 (69%) were auto processed actions and 5212 (31%) manual actions.

Minus some special cases, a summary of the auto-processed and manually triggered actions follows:

action	archive	inspect	pending	pretest	publish	recheck
auto	2731	4341	460	0	3148	938
manual	1902	70	286	145	2262	547

These include the final decisions for the submissions which were

action	archive	publish
auto	2562 (26.1%)	2779 (28.3%)
manual	1861 (18.9%)	2620 (26.7%)

where we only count those as *auto* processed whose publication happened automatically in all steps.

CRAN mirror security

Currently, there are 100 official CRAN mirrors, 68 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).

New CRAN task views

Databases Topic: Databases with R. Maintainer: Yuan Tang. Packages: **DBI***, **DBItest**, **MonetDBLite**, **R4CouchDB**, **RCassandra**, **RGreenplum**, **RH2**, **RJDBC**, **RMariaDB**, **RMySQL**, **ROracle**, **RPostgreSQL**, **RPostgres**, **RPresto**, **RSQLite**, **RcppRedis**, **TScompare**, **bigrquery**, **dbfaker**, **dbplyr**, **dplyr**, **dplyr.teradata**, **elastic**, **filehashSQLite**, **implyr**, **influxdbr**, **liteq**, **mongolite**, **odbc***, **pivot**, **pointblank**, **pool**, **redux**, **rpostgis**, **sqldf**, **tidyr**, **uptasticsearch**.

MissingData Topic: Missing Data. Maintainer: Julie Josse, Nicholas Tierney and Nathalie Vialaneix (r-miss-tastic team). Packages: **Amelia***, **BaBooN**, **BaylorEdPsych**, **CALIBERrfimpute**, **CMF**, **CRTgeeDR**, **CVThresh**, **CoImp**, **DMwR**, **DTWBI**, **DTWUMI**, **DescTools**, **DiffusionRimp**, **DrImpute**, **FHDI**, **FamEvent**, **FastImputation**, **ForImp**, **GSE**, **GenForImp**, **Haplin**, **HardyWeinberg**, **Hmisc**, **HotDeckImputation**, **JointAI**, **MissMech**, **MixedDataImpute**, **NNLM**, **NPBayesImputeCat**, **OpenMx**, **PSIMEX**, **PSM**, **PST**, **QTLRel**, **Qtools**, **RNAseqNet**, **ROptSpace**, **Rmagic**, **Rphylopars**, **SNPassoc**, **StAMPP**, **StatMatch**, **StratifiedRF**, **TAM**, **TAR**, **TVsMiss**, **TestDataImputation**, **TippingPoint**, **TreePar**, **TreeSim**, **VIM***, **VIMGUI**, **VarSelLCM**, **WaverR**, **accelmissing**, **ade4**, **alleHap**, **brlrmr**, **cat**, **cdparcoord**, **cobalt**, **cutoffR**, **dejaVu**, **denoiseR**, **dils**, **dlookr**, **eigenmodel**, **experiment**, **extracat**, **fastLink**, **filling**, **forecast**, **gapfill**, **gsynth**, **hmi**, **hot.deck***, **icdGLM**, **icenReg**, **idealstan**, **idem**, **imputePSF**, **imputeTS***, **imputeTestbench**, **ipw**, **jomo***, **lavaan**, **ltm**, **mdmb**, **memisc**, **mi**, **mice***, **miceFast**, **miceMNAR**, **miceadds**, **micemd**, **mirt**, **missForest**, **missMDA***, **mitml**, **mitools**, **mix**, **naniar***, **nipals**, **norm**, **padr**, **pan**, **phylin**, **plsRglm**, **powerlmm**, **prefmod**, **prophet**, **pseval**, **randomForest**, **reddPrec**, **robCompositions**, **robustrao**, **rsem**, **rtop**, **samon**, **sbart**, **sbgcop**, **scorecardModelUtils**, **simputation**, **sjlabelled**, **sjmisc**, **smcfcs**, **softImpute***, **spacetime**, **sptemExp**, **stlplus**, **swgee**, **tabplot**, **tidyimpute**, **timeSeries**, **tsibble**, **wNNSel**, **wrangle**, **xts**, **yaImpute***, **zCompositions**, **zoo**.

(* = core package)

New packages in CRAN task views

ChemPhys **ATmet**, **MALDIrppa**, **NISTunits**, **constants**, **errors**, **fingerprint**, **measurements**, **metRology**, **spectralAnalysis**, **units**.

Cluster **MixAll**.

DifferentialEquations **diffeqr**, **sundialr**.

Distributions **Wrapped**, **bivariate**, **pgdraw**.

Econometrics **bife**, **nse**.

Finance **PeerPerformance**, **crseEventStudy**, **riskParityPortfolio**.

HighPerformanceComputing **bigstatsr**, **qsub**.

MetaAnalysis **MetaStan**, **MetaSubtract**, **dfmeta**, **metamedian**, **ofGEM**, **puniform**.

NumericalMathematics **chebpol**, **freegroup**.

OfficialStatistics **BayesSAE**, **RRreg**, **SmallCountRounding**, **census**, **censusGeography**, **emdi**, **idbr**, **ipumsr**, **noncensus**, **sae**, **tidycensus**.

Optimization **caRamel**, **lbfgsb3c**, **nilde**, **osqp**.

Phylogenetics **apex**, **aphid**, **brms**, **brranching**, **ecospat**, **entropart**, **enveomics.R**, **hisse**, **metacoder**, **nodiv**, **picante**, **warbleR**.

Psychometrics **NetworkToolbox**, **TreeBUGS**, **cNORM**, **equateMultiple**, **ifaTools**, **mudfold**.

Spatial **RPostgreSQL**, **cartogram**, **geogrid**, **geometa**, **geonapi**, **geosapi**, **inlmisc**, **landscapemetrics**, **leaflet**, **lwgeom**, **ows4R**, **rnaturalearth**, **stars**.

TimeSeries **ARCensReg**, **FKF**, **GNAR**, **NTS**, **RTransferEntropy**, **TSEntropies**, **changepoint.mv**, **dsa**, **ggTimeSeries**, **gmvarkit**, **graphicalVAR**, **imputePSF**, **imputeTestbench**, **mgm**, **multDM**, **rollRegres**, **sugrrants**, **sym.arma**, **tbrf**.

WebTechnologies **HIBPwned**, **LendingClub**, **OpenML**, **RSauceLabs**, **RSocrata**, **Rcrawler**, **WikidataQueryServiceR**, **ZillowR**, **ajv**, **analogsea**, **aws.polly**, **aws.s3**, **aws.sns**, **banR**, **birdnik**, **brandwatchR**, **colourlovers**, **crminer**, **crplyr**, **crul***, **crunch**, **crunchy**, **docuSignr**, **duckduckr**, **facebook.S4**, **fauxpas**, **feedeR**, **fulltext**, **ganalytics**, **gdns**, **geonapi**, **geosapi**, **gh**, **giphyr**, **googleComputeEngineR**, **googleLanguageR**, **htm2txt**, **htmltidy**, **htmltools**, **httpcode**, **httptest**, **internetarchive**, **iptools**, **js**, **jstor**, **languagelayeR**, **mapsapi**, **mathpix**, **ndjson**, **notifyme**, **owmr**, **ows4R**, **pivotaltrackR**, **plotly**, **postlightmercury**, **radiant**, **rapiclient**, **rcoreoa**, **rcrossref**, **rdpla**, **rdrop2**, **refimpact**, **reqres**, **rerddap**, **restfulr**, **rhub**, **rjsonapi**, **roadoi**, **robotstxt**, **routr**, **rpinterest**, **rtweet**, **rwars**, **securitytxt**, **seleniumPipes**, **sparkbq**, **spiderbar**, **swagger**, **tidyRSS**, **trelloR**, **tuber**, **tubern**, **ubeR**, **udapi**, **validatejsonr**, **vcr***, **vkR**, **webmockr***, **xslt**.

(* = core package)

Kurt Hornik
WU Wirtschaftsuniversität Wien, Austria
Kurt.Hornik@R-project.org

Uwe Ligges
TU Dortmund, Germany
Uwe.Ligges@R-project.org

Achim Zeileis
Universität Innsbruck, Austria
Achim.Zeileis@R-project.org

News from the Bioconductor Project

by Bioconductor Core Team

The *Bioconductor* project provides tools for the analysis and comprehension of high-throughput genomic data. *Bioconductor* 3.8 was released on 31 October, 2018. It is compatible with R 3.5.2 and consists of 1649 software packages, 360 experiment data packages, and 941 up-to-date annotation packages. The release announcement includes descriptions of 95 new software packages and updated NEWS files for many additional packages.

Start using *Bioconductor* by installing the most recent version of R and evaluating the commands

```
if (!requireNamespace("BiocManager"))
    install.packages("BiocManager")
BiocManager::install()
```

Install additional packages and dependencies, e.g., **SingleCellExperiment**, with

```
BiocManager::install("SingleCellExperiment")
```

Additional installation instructions are available. Docker and Amazon images provide an effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments. Key resources include:

- The bioconductor.org web site to install, learn, use, and develop *Bioconductor* packages.
- A listing of available software, linking to pages describing each package.
- A question-and-answer style user support site and developer-oriented mailing list.
- The F1000Research *Bioconductor* channel for peer-reviewed *Bioconductor* work flows.
- Our package submission repository for open technical review of new packages.
- The *Bioconductor* community slack for in-depth conversation about *Bioconductor* software use and development.

Key training resources include common workflows and last year's conference workshop booklet. Our annual conference will be on June 24 through 27, 2019 in New York City, with a line-up of morning scientific talks and afternoon user-oriente hands-on workshops, as well as a developer day, starting to take shape.

Bioconductor *Core Team*
Biostatistics and Bioinformatics
Roswell Park Comprehensive Cancer Center, Buffalo, NY
USA
maintainer@bioconductor.org

Changes in R

From version 3.5.0 patched to version 3.5.2

by R Core Team

CHANGES IN R 3.5.2

PACKAGE INSTALLATION

- New macro 'CXX_VISIBILITY' analogous to 'C_VISIBILITY' (which several packages have been misusing for C++ code) for the default C++ compiler (but not necessarily one used for non-default C++ dialects like C++14).

TESTING

- The random number generator tests in 'tests/p-r-random-tests.R' no longer fail occasionally as they now randomly sample from "certified" random seeds.

BUG FIXES

- The "glm" method of drop1() miscalculated the score test (test="Rao") when the model contained an offset.
- Linear multiple empty models such as lm(y ~ 0) now have a correctly dimensioned empty coefficient matrix; reported by Brett Presnell.
- vcov(<empty mlm>) and hence confint() now work (via a consistency change in summary.lm()).
- confint(<multiple lm()>) now works correctly; reported on R-devel by Steven Pav.
- quade.test() now also works correctly when its arguments are not yet sorted along groups, fixing PR#15842.
- Installation on a Unix-alike tries harder to link to the 'pthread' library where required (rather than relying on OpenMP to provide it: configuring with '--disable-openmp' was failing on some Linux systems).
- The data.frame method for print(x) is fast now also for large data frames x and got an optional argument max, thanks to suggestions by Juan Telleria.
- hist() no longer integer overflows in very rare cases, fixing PR#17450.
- untar() ignored a character compressed argument: however many external tar programs ignore the flags which should have been set and automagically choose the compression type, and if appropriate gzip or bzip2 compression would have been chosen from the magic header of the tarball.
- zapsmall(x) now works for more "number-like" objects.
- The tools-internal function called from R CMD INSTALL now gets a warnOption = 1 argument and only sets options(warn = warnOption) when that increases the warning level (PR#17453).
- Analogously, the tools-internal function called from R CMD check gets a warnOption = 1 argument and uses the larger of that and getOption("warn"), also allowing to be run with increased warning level.
- Parse data now have deterministic parent nodes (PR#16041).

- Calling `match()` with length one `x` and POSIXlt `table` gave a segfault (PR#17459).
- Fork clusters could hang due to a race condition in cluster initialization (`makeCluster()`).
- `nextn(n)` now also works for larger `n` and no longer loops infinitely for e.g, `n <-214e7`.
- `cooks.distance()` and `rstandard()` now work correctly for multiple linear models (`"mlm"`).
- `polym()` and corresponding `lm()` prediction now also work for a boundary "vector" case fixing PR#17474, reported by Alexandre Courtiol.
- With a very large number of variables `terms()` could segfault (PR#17480).
- `cut(rep(0,7))` now works, thanks to Joey Reid and Benjamin Tyner (PR#16802).
- `download.file(*,method = "curl",cacheOK = FALSE)` should work now on Windows, thanks to Kevin Ushey's patch in PR#17323.
- `duplicated(<dataframe with 'f'>)` now works, too, thanks to Andreas Kersting's PR#17485; ditto for `anyDuplicated()`.
- `legend(*,cex = 1:2)` now works less badly.
- The `print()` method for `POSIXct` and `POSIXlt` now correctly obeys `getOption("max.print")`, fixing a long-standing typo, and it also gets a corresponding optional `max` argument.
- Unserialization of raw vectors serialized in ASCII representation now works correctly.
- `<data frame>[TRUE,<new>] <-list(c1,c2)` now works correctly, thanks to Suharto Anggono's PR#15362 and Emil Bode's patch in PR#17504.
- `seq.int(*,by=by,length=n)` no longer wrongly "drops fractional parts" when `by` is integer, thanks to Suharto Anggono's report PR#17506.
- Buffering is disabled for `file()` connections to non-regular files (like sockets), as well as `fifo()` and `pipe()` connections. Fixes PR#17470, reported by Chris Culnane.

CHANGES IN R 3.5.1

BUG FIXES

- `file("stdin")` is no longer considered seekable.
- `dput()` and `dump()` are no longer truncating when `options(deparse.max.lines = *)` is set.
- Calls with an S3 class are no longer evaluated when printed, fixing part of PR#17398, thanks to a patch from Lionel Henry.
- Allow `file` argument of `Rscript` to include space even when it is first on the command line.
- `callNextMethod()` uses the generic from the environment of the calling method. Reported by Hervé Pagès with well documented examples.
- Compressed file connections are marked as blocking.
- `optim(*,lower = c(-Inf,-Inf))` no longer warns (and switches the method), thanks to a suggestion by John Nash.

- predict(fm,newdata) is now correct also for models where the formula has terms such as splines::ns(..) or stats::poly(..), fixing PR#17414, based on a patch from Duncan Murdoch.
- simulate.lm(glm(*,gaussian(link = <non-default>))) has been corrected, fixing PR#17415 thanks to Alex Courtiol.
- unlist(x) no longer fails in some cases of nested empty lists. Reported by Steven Nydick.
- qr.coef(qr(<all 0,w/ colnames>)) now works. Reported by Kun Ren.
- The radix sort is robust to vectors with >1 billion elements (but long vectors are still unsupported). Thanks to Matt Dowle for the fix.
- Terminal connections (e.g., stdin) are no longer buffered. Fixes PR#17432.
- deparse(x), dput(x) and dump() now respect c()'s argument names recursive and use.names, e.g., for x <-setNames(0,"recursive"), thanks to Suharto Anggono's PR#17427.
- Unbuffered connections now work with encoding conversion. Reported by Stephen Berman.
- '.Renviron' on Windows with Rgui is again by default searched for in user documents directory when invoked *via* the launcher icon. Reported by Jeroen Ooms.
- printCoefmat() now also works with explicit right=TRUE.
- print.noquote() now also works with explicit quote=FALSE.
- The default method for pairs(..,horInd=*,verInd=*) now gets the correct order, thanks to reports by Chris Andrews and Gerrit Eichner. Additionally, when horInd or verInd contain only a subset of variables, all the axes are labeled correctly now.
- agrep("..|..",..,fixed=FALSE) now matches when it should, thanks to a reminder by Andreas Kolter.
- str(ch) now works for more invalid multibyte strings.