The R Journal

Volume 11/2, December 2019

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

Contents

News and Notes

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

Email:
r-journal@R-project.org

R Journal Homepage:
https://journal.r-project.org/

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

Editorial

by Michael J. Kane

On behalf of the editorial board, I am pleased to present Volume 11, Issue 2 of the R Journal and my first issue as the Editor in Chief. This year, both Colin Gillespie and Catherine Healey join the Editorial Board, and Norm Matloff will rotate out. The R Journal continues to see increases in impact and popularity and this year we plan on making advances to better serve the community and streamline the publishing process to meet the increase in submissions we have seen over the last few years.

In this issue

Along with news and notes are provided from the ConectaR 2019 conference, the R Foundation, CRAN, and Bioconductor, this issue features 26 articles. I have categorized them below.

Papers focusing on performance and novel, domain-specific applications:

- "Comparing **namedCapture** with other R packages for regular expressions
- "**cvcrand**: a Package for Covariate-constrained Randomization and the Clustered Permutation Test for Cluster Randomization Trials
- "Indoor Positioning and Fingerprinting: The R package **ipft**

Data preprocessing, imputation, validation, and exploration:

- "**jomo**: a Flexible Package for Two-level Joint Modelling Multiple Imputation"
- "**auditor**: an R Package for Model-Agnostic Visual Validation and Diagnostics"
- "Fitting tails by the empirical residual coefficient of variation: The **ercv** package"
- "The Landscape of R Packages for Automated Exploratory Data Analysis"
- "The R Package **trafo** for Transforming Linear Regression Models"
- "**orthoDr**: semiparametric dimension reduction via orthogonality constrained optimization"

Spatial statistics:

- "**spGARCH**: An R Package for Spatial and Spatiotemporal ARCH models"
- "The **IDSpatialStats** R package: Quantifying spatial dependence of infectious disease spread"
- "Using Web Services to Work with Geodata in R"

Time-series analysis and finance:

- "**lpirfs**: An R-package to estimate impulse response functions by local projections"
- "Time Series Forecasting with KNN in R: the **tsfknn** Package"
- "**rollmatch**: An R Package for Rolling Entry Matching"
- "**BondValuation**: An R Package for Fixed Coupon Bond Analysis"

- "Modeling regimes with extremes: the **bayesdfa** package for identifying and forecasting common trends and anomalies in multivariate time-series data"

Clustering:

- "**roahd** Package: Robust Analysis of High Dimensional Data"
- "**PPCI**: an R Package for Cluster Identification using Projection Pursuit"
- "**ConvergenceClubs**: A Package for Performing the Phillips and Sul's Club Convergence Clustering Procedure"
- "**biclustermd**: An R Package for Biclustering with Missing Values"

And supervised modeling:

- "**dr4pl**: A stable convergence algorithm for the 4 Parameter Logistic model"
- "**coxed**: An R Package for Computing Duration Based Quantities from the Cox Proportional Hazards Model"
- "Analysis of Multivariate Data and Repeated Measures Designs with the R Package **MANOVA.RM**"
- "Associative Classification in R: **arc**, **arulesCBA**, and **rCBA**"
- "**HCmodelSets**: An R Package for Specifying Sets of Well-fitting Models in High Dimensions"

Michael J. Kane
michael.kane@r-project.org

Using Web Services to Work with Geodata in R

by Jan-Philipp Kolb

Abstract Through collaborative mapping, a massive amount of data is accessible. Many individuals contribute information each day. The growing amount of geodata is gathered by volunteers or obtained via crowd-sourcing. One outstanding example of this is the OpenStreetMap (OSM) Project which provides access to big data in geography. Another online mapping service that enables the integration of geodata into the analysis is Google Maps. The expanding content and the availability of geographic information radically changes the perspective on geodata (Chilton 2009). Recently many application programming interfaces (APIs) have been built on OSM and Google Maps. That leads to a point where it is possible to access sections of geographical information without the usage of a complex database solution, especially if one only requires a small data section for a visualization.

First tools for spatial analysis have been included in the R language very early (Bivand and Gebhardt, 2000) and this development will continue to accelerate, underpinning a continual change. Notably, in recent years many tools have been developed to enable the usage of R as a geographic information system (GIS). With a GIS it is possible to process spatial data. QuantumGIS (QGIS) is a free software solution for these tasks, and a user interface is available for this purpose. R is, therefore, an alternative to geographic information systems like QGIS (QGIS Development Team 2009). Besides, add-ins for QGIS and R-packages (**RQGIS**) are available, that enables the combination of R and QGIS (Muenchow and Schratz 2017). It is the target of this article to present some of the most important R-functionalities to download and process geodata from OSM and the Google Maps API. The focus of this paper is on functions that enable the natural usage of these APIs.

Introduction and outline

This paper introduces some interesting web services for downloading, processing and visualizing geodata. The focus especially in the second half of the paper is on OpenStreetMap-data, because it is released under the Open Database License (ODbL) 1.0. That allows multiple uses of the data (Schmidt et al., 2013). The study of Barrington-Leigh and Millard-Ball (2017) shows for example, that the data quality available at OSM is already sufficient in many countries to use it for scientific and analytic purposes. However, Barron et al. (2014) state that the quality of the OSM-data depends on the individual use case. And Grippa et al. (2018) mention that it is essential to consider the variations at regional or national scales. One example of a scientific analysis based on OSM-data is the Simulation of Urban MObility (SUMO) project (Behrisch et al. 2011). Meijer et al. (2018) for example use OSM-data to analyze global patterns of road infrastructure. Gervasoni et al. (2018) use OSM-data to generate urban features that help to estimate population density at a higher resolution. Arsanjani et al. (2015) give an overview of typical and recent examples of studies done with OSM-data. Much more research, carried out in various countries, is listed at OpenStreetMap Wiki (2017e).

The focus is on the most important APIs to download geodata. The significant advantage of using these specific APIs is that we can obtain data free of charge. Short examples are used to describe how the data can be imported into R and processed. Some examples show the easiest and fastest way to get the information needed. In other examples I look a bit further behind the scenes. Static maps can be used as background information for geographic visualization and may be used to highlight positions of so-called points of interest (poi). A prerequisite to visualise these points is the availability of their exact spatial location. With the Overpass API (`http://wiki.openstreetmap.org/wiki/Overpass_API`) for example, we can get the positions for many points of interest. This application programming interface (API) is perfect to download data on very particular topics. For example, if you are looking for special map features.

The used API's are listed in the individual sections below. I discuss an example where I am interested in public transportation in Amsterdam. In the next section (Background Maps - Download via Map Tile Servers), hints on the download of static maps from so-called map tile servers are presented. In the third section (Geocoding with Application Programming Interfaces (APIs) the functionality of APIs like the Google Maps and OSM Nominatim API is used to realize geocoding. It is shown, how the Nominatim API can be used to search OSM-data by name and address (OpenStreetMap Wiki 2018a). In the fourth section (Downloading and Importing OSM-data) I show various possibilities to download more general OSM-data. The usage of the main OSM-API is presented as well as some functions of the **osmdata** package, which also uses the Overpass API, are described in this section. Possibilities to process OSM-data with R are presented in the fifth section (Processing OSM-data). A

summary follows at the end.

Background maps - download via Map Tile Servers

If a background map is needed, it is possible to use a tile server to download them. Map tiles are quadratic bitmap graphics which are arranged in a grid to show a map. The vector tile is a newer format developed recently which is for example used by Mapbox (https://www.mapbox.com/). Vector tiles have a vector representation (OpenStreetMap Wiki 2018d, p. 1). The tiles contain vector data instead of the rendered image and provide readable, descriptive, and extensible content (Li et al. 2018). Vector tiles can be rendered dynamically and allow for an efficient extraction of the relevant data (Gaffuri 2012, p. 94).

So-called map tile servers offer to download static maps of various types. It is, for example, possible to get maps on such diverse issues as biking, public transportation, or land shading.[1] Map Tiles are very suitable for the use as background image. Various R-packages can be used to access map tile servers. One way to get static maps is the package **OpenStreetMap**. It is a package to access high-resolution raster maps using the OSM protocol (Fellows, 2016). A high number of satellite, topographic and road map servers can be accessed directly using the JMapViewer Java component (Stotz, 2018). The used map servers are for example CloudMade, Mapnik, Bing, Stamen, and MapQuest. The function `openmap` can be used to retrieve a map. It is necessary to provide values for the upper left latitude and longitude value as well as for the lower right values. In the example below, this is done for some coordinates in Amsterdam. Also, we have to specify the type of source. That may be the tile server from which to get the map or the uniform resource locator (URL) pattern. However, OSM servers have limited capacity, and heavy use adversely affects the purpose of use. With the package **OpenStreetMap**, it is also possible to access other web services. Bing Maps, the web mapping service provided by Microsoft is one example. In the following code example, the function `openmap` is used to get a map based on latitude and longitude coordinates.

We need a geocode to get a map of Amsterdam. In the next section, geocodes will be explained in more detail. We specify the tile server with the argument `type`. In this example, OSM is chosen, but a Bing map would also be possible. The result of this call is visible in Figure 1.

```
library("OpenStreetMap")
map <- openmap(c(52.278174, 4.729242),
               c(52.431064, 5.079162),
               type = "osm")
plot(map)
```

Stamen is an alternative source. Stamen Design publishes maps under a Creative Commons license CC BY-3.0 (Attribution). The maps are based on OSM-data (Lamigueiro 2014, p. 95). We get a Stamen map when we add further arguments to the `openmap` call. In the following the source is stamen. The type was specified as toner and watercolor. The resulting Stamen maps are depicted in Figure 2. The downloaded maps are very suitable as background for info graphics. It is possible to add further layers using, for example, the **ggplot2** framework (Wickham 2009). That will be shown later.

```
map_stt <- OpenStreetMap::openmap(c(52.385914, 4.874383), c(52.35514, 4.92054),
              type = "stamen-toner")
map_st <- OpenStreetMap::openmap(c(52.278174, 4.729242), c(52.431064, 5.079162),
              type = "stamen-watercolor")
plot(map_st)
plot(map_stt)
```

Another package to get static maps is **ggmap** (Kahle and Wickham 2013). This package provides a collection of functions to visualize spatial data and models on top of static maps from various online sources, like Google Maps, OSM, Stamen Maps and CloudMade Map (Kahle and Wickham, 2013). Only a few lines of code are necessary to get a map for a freely selectable location. The default source of **ggmap** is the Google Static Maps API, and with the download of these images, you agree to the terms of usage (https://developers.google.com/maps/terms - Dorman 2014). Recently, the Google Maps API terms of use have changed. Now you need an account to use the API for downloading a static map. The development version on Github (https://github.com/dkahle/ggmap) already has the function `register_google` where you can define your key. Previously you have to register your project at https://cloud.google.com/maps-platform/. The function `qmap` is a wrapper for `ggmap` and `get_map`. It is necessary to specify the place for which the map should be downloaded and a zoom factor, whereas the zoom parameter takes values between three and 21. A whole continent is visible

[1] An overview on the map tile servers is available at http://wiki.openstreetmap.org/wiki/Tileserver

Figure 1: Map of Amsterdam - data from OSM

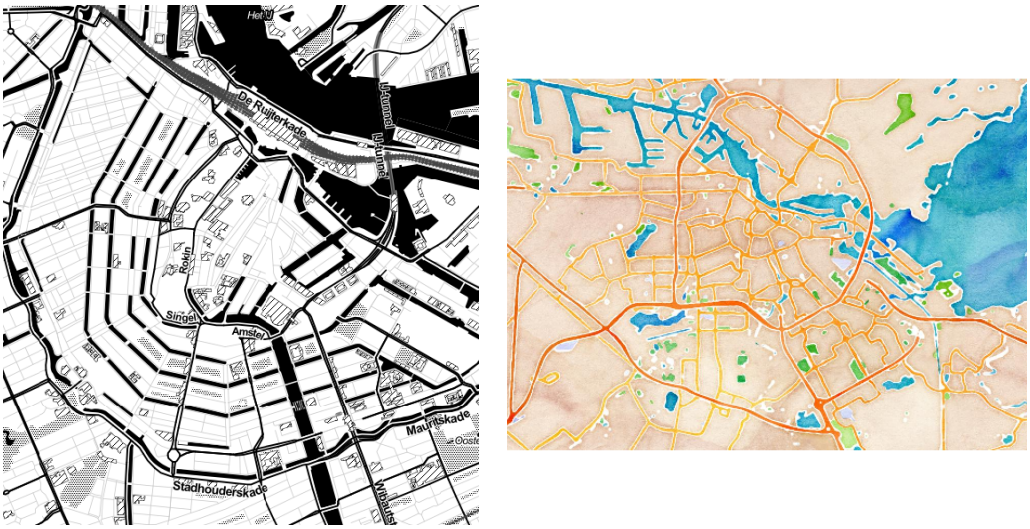


Figure 2: Static stamen maps of Amsterdam

on the map for a zoom factor of three whereas only one building is on the map for a zoom factor of 21. The default value is ten. In this case, a city is visible on the map.

The **RgoogleMaps** package can be used for querying OSM servers for static maps in the form of portable network graphics (PNGs) (Loecher and Ropkins, 2015). Map tiles can be downloaded using, for example, the command `GetMap.bbox`. In this case, the center and a zoom level have to be provided. The center is determined using so-called geocodes. Geocodes are used to specify a precise location on the map. More information about these codes will be given in the next section.

Also, it is possible to create interactive maps with online mapping services. This can be done for example with the R-package **leaflet** created by Cheng and Xie (2016). The package can be used to create interactive maps for websites. These kinds of maps are also known as slippy maps where it is possible to zoom and pan (OpenStreetMap Wiki 2016b). That means that the map slips around when you drag the mouse. Slippy maps in OSM are based on the AJAX library OpenLayers which is written in JavaScript. Here, the default OSM tiles can be added to the interactive map visualization. It is also possible to use Stamen maps or CartoDB as background for slippy maps (Abernathy 2016, p. 311). A good starting point for the work with this package is `https://rstudio.github.io/leaflet/`. In the following example, the pipe-forward operator of package **magrittr** is used which enables chain operations. The first operation in this chain is the creation of a leaflet map widget (function `leaflet`). The second operation is to add a layer (function `addTiles`), and in the last operation a marker is added (function `addMarkers`). Here we have to specify the position of the marker and the text that pops up.

```
library(leaflet)
leaflet() %>%
  addTiles() %>%
  addMarkers(lng=4.891013, lat=52.38054, popup = "Amsterdam")
```

Map Tiles are a good possibility for geographic data visualization. They can be valuable to get a first impression, but these visualizations may also occlude relevant geodata. Therefore it is useful to know, how to add further information to the map. In the next section, it is shown how to append more information to either the static map or the interactive map. Interactive maps can for example be produced very easily with **tmap** and **mapview** (Appelhans et al. 2018) and many other packages. The functionalities for interactive maps in both mentioned packages are built on top of the R-package **leaflet**. The interface to Javascript allows a very vital exchange. The R-package **mapdeck** is for example a very suitable tool, for a browser based visualization of geodata. Like most of the interactive graphics in R this package is also based on a Javascript library. In this case Mapbox GL JS is used (Cooley 2018a). Like for package **deckard** (see Hansel 2018), the package also provides access to the Deck.gl framework of Uber (Lovelace et al. 2018). A registration is necessary to use the framework. The package **lawn** provides a client for the geospatial analysis with the javascript library `Turfjs` (Chamberlain and Hollister 2017). That allows us to use for example Javascript libraries like `geojson-random` and `geojsonhint`, which can be used to randomly create GeoJSON objects and to color them. For example the `gr_polygon` function can be used to create an example object. Then you can plot the object with the generic function `view`.

We have seen in the examples above, that we need a set of coordinates to locate a point of interest or pop-up. For a study on the transport system, it is for example good to know which public transport stops (train or bus stops, etc.) are located in the surrounding of the area under research. The geocoding for an address list of these stops can be done with the Nominatim API of the OSM project. In the next section it is shown how this geocoding process is done in R.

Geocoding with Aplication Programming Interfaces (APIs)

Geocoding is the derivation of a structured spatial representation from textual information like postal codes (Aitchison, 2009, p. 157). Many possibilities are available to realize geocoding with R. The most popular is perhaps the usage of interfaces like the Google Maps API. The API is described in Svennerberg (2010). It can be accessed directly from R using the R-package **ggmap** (Kahle and Wickham, 2013, p. 156). The **ggmap**-package was one of the first R packages to provide an interface for data exchange between R and the Google Maps API. The process can be implemented using the `geocode` function. Then we just need an adress to get the corresponding coordinates. For the example "Waterlooplein 1, Amsterdam, Zentrum" we get the latitude and longitude coordinates visible in Table 1 as a result of this call.

The Mercator-projection is used in Google Maps (Moore and Drecki 2008, p. 206). The European Petroleum Survey Group Geodesy (EPSG) published a system of globally unique key numbers of geodetic data records (EPSG codes). The used coordinate reference system and the projection are determined using this EPSG codes (`http://epsg.org/`). The EPSG code used in this example is 3857 (Harris 2016). One of the issues when using the Google Maps API is that only 2,500 requests per day

	lat	lon
1	4.901323	52.36896

Table 1: Latitude and longitude coordinates of the address "'Amsterdam, Waterlooplein 1"'

can be performed free of charge, which might cause problems with large data sets. The terms of usage of the API can be seen at https://developers.google.com/maps/terms. It is clear that nobody should use such an online service to geocode privacy sensitive data. The **googleway** package also connects to Google (Cooley 2018b). To use the package, the registration and an API key is necessary to use most of the functionalities. When we have done this, it is for example possible to query the distance, the elevation or the timezone. Additional information on the visinity of a point is accessible using for example the packages **geonames** (https://github.com/ropensci/geonames) or **RDSTK**.

A less known alternative described here in more detail is the usage of the Nominatim API of the OSM project (Warden, 2011, p. 25). This tool is an open source system designed on top of OSM-data to search by name and address (Clemens, 2015). Nominatim (OpenStreetMap Wiki 2018a) is the main geocoder maintained by OSM (Abernathy, 2016). Detailed information on this geocoder is available at http://wiki.openstreetmap.org/wiki/Nominatim. Similar to the geocoding with the Google Maps API, the service Nominatim can be used to query the name of the reference object to obtain corresponding GPS coordinates. When using the Nominatim API, it is possible to choose between different output formats. We can choose between HTML, XML, JSON and JSONV2. The **RJSONIO** and **jsonlite** packages can be used to import JSON files to R (Lang 2014 and Ooms 2014). The core from the following example is the command `fromJSON` of package **RJSONIO**. It converts JSON content to R objects. The code chunk is designated to get the corresponding coordinates for the address "'Rozengracht 1"' in Amsterdam. With the function `url` we specify a path to be opened. In this case it is the adress of the Nominatim API http://nominatim.openstreetmap.org/ plus some additional information (format and adress details).

```
library("RJSONIO")
con <- url("http://nominatim.openstreetmap.org/search?format=json&
addressdetails=1&extratags=1&q=Amsterdam+Niederlande+Rozengracht+1")
geoc <- fromJSON(paste0(readLines(con, warn=F)))
close(con)
```

The result is an object that contains a lot of information. We can get an overview when we query the names of the first object in geoc.

```
names(geoc[[1]])
```

```
 [1] "place_id"     "licence"      "osm_type"     "osm_id"       "boundingbox"
 [6] "lat"          "lon"          "display_name" "class"        "type"
[11] "importance"   "address"      "extratags"
```

This object contains for example information on the license which is ODbL 1.0 (http://www.openstreetmap.org/copyright). We get the latitude and longitude coordinates:

```
geoloc <- c(geoc[[1]][which(names(geoc[[1]]) == "lat")],
            geoc[[1]][which(names(geoc[[1]]) == "lon")])
```

	lat	lon
1	52.3737223	4.8826404

We also get the combination of OSM id and the OSM type, which can be useful information, when downloading and processing specific OSM-data. And then we have some key-value pairs, which will be explained below. The package **jsonlite** can also be used for importing json data:

```
link <- url("http://nominatim.openstreetmap.org/search?format=json&
          addressdetails=1&extratags=1&q=Amsterdam+Niederlande+Rozengracht+1")
geoc2 <- jsonlite::fromJSON(link)
geoc2df <- with(geoc2, data.frame(osm_id, lat,lon))
geoc2df$house_number <-  geoc2$address$house_number
```

	osm_id	lat	lon	house_number
1	2721815875	52.3737223	4.8826404	1
2	2743624072	52.3719482	4.8755534	237-1
3	2721830930	52.3736673	4.8823914	7-1
4	2721827922	52.3734021	4.8813371	53-1
5	2721824637	52.372232	4.8767542	231-1
6	2721823434	52.3724786	4.8776618	187-1
7	2721820122	52.3727335	4.8786657	137-1
8	2721816644	52.3729874	4.8797588	105E-1
9	2720971311	52.3727658	4.8775263	194-1
10	2720971056	52.3728019	4.8775994	184-1

Table 2: Adresses for house number with one - "Rozengracht"

The dataframe `geoc2df` contains several addresses in the "Rozengracht" street that all start with a one. Parts of the adresses are visible in the following table.

The package **tmaptools** offers tool functions to supply the workflow to create thematic maps. It provides the function `geocode_OSM` which is a wrapper for geocoding using the OSM Nominatim API (Tennekes 2018). A bounding box can be created with the `tmap::bb` command. The functions `tmaptools::bb_poly`, and `osmdata::getbb` are also worth mentioning here, in particular with regard to extracting bounding polygons rather than mere boxes. In the following example for function `geocode_OSM` of package **tmaptools** we get only the coordinates as output because the default value of the argument `details` is `FALSE`.

```
library("tmaptools")
gc_tma <- geocode_OSM("Amsterdam, Buiten Brouwersstraat")
```

Info	Nominatim	Google Maps
CRS	EPSG:4326	EPSG:3857
Projection	Mercator	Mercator
Longitude	4.891013	4.900478
Latitude	52.380541	52.36859

Table 3: Result of the geocoding request for a poi in Amsterdam

The result of the request for a postal address in Amsterdam is visible in Table 3 in the second column. A big difference becomes apparent when we compare the result for Nominatim and the result for Google Maps API (third column). The projection and the coordinate reference system (CRS) may be of great importance (Brown, 2016). The projection is used to display the three-dimensional earth. Typically, this is a projection onto a two-dimensional map display. Google Maps uses, for example, the Mercator projection, which is good for zoomed-in viewing, but causes distortions when zoomed out (Turner 2006). For the Nominatim-query we get EPSG:4326 (Maier 2014).

The projection of the data is often necessary for the work with geodata from different sources, and this is true if we want to combine the gained information with other geodata, for example, static maps or satellite pictures. In the following example a transformation is shown. In a first step we create a `data.frame` which is called `poi`. In a second step we use the function `coordinates` to set spatial coordinates and to create a spatial object. Then we set the projection attributes with the command `proj4string`. In this case we use the epsg projection 4326. Afterwards it is possible to transform the spatial points using the function `spTransform`.

```
library(sp)
poi <- data.frame(lat = gc_tma$coords["x"],
                  lon = gc_tma$coords["y"])
sp::coordinates(poi) <- c("lat", "lon")
sp::proj4string(poi) <- sp::CRS("+init=epsg:4326")
res <- spTransform(poi, CRS("+init=epsg:3035"))
```

We get the following numbers for the coordinates:

```
res@coords
      lat     lon
  3973434 3264547
```

A clearer alternative to realize the coordinate reference system transformation is included in package **sf**:

```
library(sf)
poi2 <- st_sfc (st_point(gc_tma$coords), crs = 4326)
res2 <- st_transform (poi2, crs = 3035)
```

Further functions are available to switch between reference systems. The **OpenStreeetMap** package has the `openproj` function to translate from Mercator to another coordinate reference system and can be used to create **ggplot2** and base graphics. Lovelace et al. (2017) present a possibility to transform the reference system. A whole section in the book of Pebesma and Bivand (2019) is dedicated to this topic. Brown (2016) also explains how to work with map projections and coordinate reference systems in R. More information is available in Plant et al. (2012). Further, the **mapmisc** package provides projection capabilities and utilities for producing maps (Brown 2016). The function `projection` supplies information on the coordinate reference system.

The accessed geocodes might be combined with a static map in the next step. As already shown, the `openmap` function from package **OpenStreetMap** can be utilized to download a background map. We have to re-project the original OSM map, for example with the function `openproj` from package **OpenStreetMap**. The static map can then be combined with the extracted geocode. The result is visible in Figure 3.

```
poi <- data.frame(lon = gc_tma$coords["x"],
                  lat = gc_tma$coords["y"])
adm_map <- openproj(map_stt,
                    projection = "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
library(ggmap)
autoplot(adm_map) + geom_point(aes(x = lon, y = lat), data = poi,size=5, col="red")
```

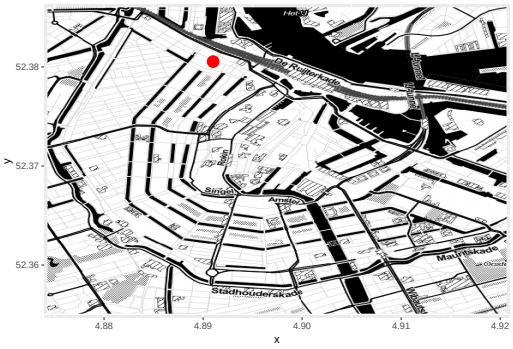


Figure 3: Amsterdam, Buiten Brouwersstraat

Only the coordinates are available for the download in the Waterlooplein example. In the following example, the package **opencage** is used for geocoding. The Opencage service (`https://opencagedata.com/`) combines the quality of multiple geocoders in one API. An access token is necessary to use this geocoder. With the free token, up to 2,500 calls per day are possible. The command `opencage_forward` can be used for geocoding. A small part of the information in the resulting object (for argument `placename = "Amsterdam,Van Woustraat"`), the address information, is visible in Table 4. We also get the suburb, the city district, the city, the state district, the state and the postcode area in which the object is located.

Reverse geocoding is the counterpart to geocoding. It is the target to retain additional information from the geocodes. The procedure described above is reversed. Thus, the starting point

	Value
road	Van Woustraat
neighbourhood	de Pijp
suburb	Zuid
city	Amsterdam
state	Noord-Holland
postcode	1017
country	Nederland
country_code	nl

Table 4: Address information for an object in Amsterdam, Van Woustraat.

is the geocode and the task is to extract textual information, such as an address or a name, from these geographic coordinates (Kounadi et al. 2013). The Nominatim API can be used for this. The function `rev_geocode_OSM` of package **tmaptools** can be used to realize this. The Nominatim service runs on donated servers. It is necessary to be careful with the usage. No heavy use is accepted. The absolute maximum is one request per second. For details on the usage see https://operations.osmfoundation.org/policies/nominatim/. If bulk geocoding of larger amounts of OSM-data is intended, it is necessary to look for alternatives. One possibility are third-party providers like MapQuest (https://developer.mapquest.com/documentation/open/nominatim-search/). Another possibility is the offline geocoding. The source code of the Nominatim API is available in the OSM svn repository. The readme is very detailed and can be used as a step by step guide to set up an offline geocoder.

As seen above, it is possible to geocode a list of addresses. E.g. with the Nominatim API. If we want to know in general which objects are present in a certain map section, we can use the main OSM Api. This is introduced in the following section.

Downloading general OSM-data

The 'tagging' scheme is crucial for the work with OSM-data. Its basic data structures (nodes, ways, and relations) are tagged with a key value pair (Ramm et al., 2011). Examples of this scheme are given in Table 4. A topic, category, or type of feature is named in the key. The value is used to describe the particular form. For example, there are numerous OSM objects with key=highway. It can be a footpath (value=pathway) or a highway (value=motorway). This tagging scheme is a core part of the OSM project (Haklay and Weber, 2008). A list of the available OSM map features is available at http://wiki.openstreetmap.org/wiki/Map_Features.

Three different object types exist in OSM. There are simple nodes or points. This can be, for example, a public transport stop (`key=highway` and `value=bus_stop`) or a station where you get potable water for consumption (key=amenity and `value=drinking_water`). The second object type is ways. This is a sequence of points that describe, for example, the course of roads or rivers. The third object type is relations, a grouping of objects that are logically related (Pruvost and Mooney 2017).

There are several ways to get the OSM data. The Humanitarian Open Streetmap Team (https://export.hotosm.org) offers customized extracts of up-to-date OSM data. Basically the raw data are offered in protocolbuffer binary format (PBF) or in extensible markup language (XML) format. An alternative to download large OSM sections is the Geofabrik page (http://download.geofabrik.de/). Here you can download current excerpts as well as shapefiles. The shapefile format is a popular format of spatial vector data for geographic information systems (GIS). This file type is a geodata format originally developed for ESRI's ArcView software.

A special feature of the OSM project is that you can not only get geolocations, but also download individual objects or collections of objects from OSM. The OSM API version 0.6 (http://api.openstreetmap.org/api/0.6/) is optimized for editing. That means, it is used for fetching and saving raw geodata. It is a REST (representational state transfer) API. Thus, it is possible to make HTTP GET calls to the OSM API (Mooney and Corcoran, 2011). For more information on REST APIs and RESTful web service interfaces see for example Masse (2011). For this paper, it is important that an extract for a pre-defined area can be retrieved from the OSM database using REST.

It is instructive to see how to parse the information to R. In the following use case, information for Amsterdam is downloaded to R. We can get the OSM id using the search function at https://www.openstreetmap.org/. This id is essential for the work with OSM-data. With the object id it is possible to uniquely identify an object. In the case below we download information for the relation

Amsterdam, which has the OSM id 47811. The first step is to specify the correct url. This url is a combination of a fixed part, the address of the API (`https://www.openstreetmap.org/api/0.6/`) and a variable part based on the object to be downloaded (`relation/4290854847`). Thus, we need to know if the object is a node, a way, or a relation. In addtion we need the object id. In a second part, the file is downloaded from the Internet using the `download.file` command and saved.

```
url2 <-"https://www.openstreetmap.org/api/0.6/node/4290854847"
download.file(url2, "4290854847.xml")
```

It is also possible to save the object as a `.osm` file, a format that uses the data tree structure of XML (OpenStreetMap Wiki 2017c). This format is human readable due to a clear structure, and it is machine independent because of exact definitions. But a `.osm` file might be very huge, when it is decompressed. In the following case information for a node is downloaded.

```
ghURL2<-"https://www.openstreetmap.org/api/0.6/relation/47811"
download.file(ghURL2, "amsterdam.osm")
```

If the interest is in information in the vicinity of a point, we can download information for a small section. The following example shows the download for a section around the main train station in Amsterdam. I used the export function of `https://www.openstreetmap.org/` to download the section visible in Figure 4 as `.osm`. The same section can be downloaded using the command `curl_download` from package **curl** (Ooms 2018). The download of these small map sections are subject to the OSM API usage policy (OpenStreetMap Wiki 2018b and OpenStreetMap Wiki 2017a).

```
library(curl)
uac<-"https://api.openstreetmap.org/api/0.6/map?bbox=4.89359,52.37640,4.90589,52.38172"
curl_download(uac, "amsterdamcentraal.osm")
```

Figure 4: Export functionality `https://www.openstreetmap.org/`

Thus, it is possible to use the OSM API to download all objects in a map section. The Overpass API can be used to download all elements of a specific type. The Overpass API written by Roland Olbricht allows developers to download small extractions of user-generated content from OSM according to given criteria (OpenStreetMap Wiki 2016a). Overpass is a read-only API that provides custom selected parts of the OSM-data. It can be understood as a database over the web, it uses the fact that OSM is enriched with additional information ranging from city names to e.g. locations of street lamps or energy generators (Schmidt et al., 2013). If it is the target, to get all bus stops in Amsterdam, then it is possible to download the information from Overpass Turbo (`https://overpass-turbo.eu/`), using the key highway and the value bus_stop. An example is depicted in Figure 5. Overpass turbo is a web-based data collection tool for OSM. It runs with any overpass API query and displays the results on an interactive map (OpenStreetMap Wiki 2017d).

With a query by the client to the API you get the corresponding data. The API is streamlined for data consumers that need a few elements within a short time, selected by search criteria like for example type of objects, location, proximity, tag properties, or combinations of them (Mongiello et al., 2015). The usage of the Overpass API is especially advisable if only points of interest for a particular topic and a bigger section are relevant. It is possible to download the data in GeoJSON, GPX and KML format. The Keyhole Markup Language (KML) is a language used to describe geodata. It was used in Google Earth. KML also follows the XML-syntax. In addition also the raw data can be downloaded form Overpass Turbo. The package **XML** can be used to access the API directly from R (Lang and the CRAN Team 2016).

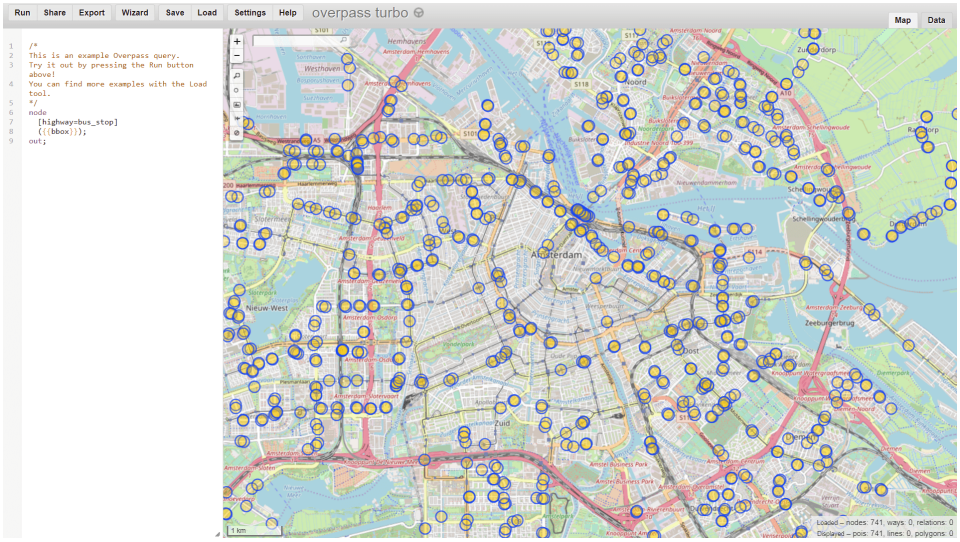


Figure 5: Bus stops in Amsterdam

Processing OSM-data

In this section the processing of the downloaded OSM-data is presented. The online book of Lovelace et al. (2018) gives a very good overview on tasks like this. In the previous section we have seen how to download data from the main OSM API. Now we parse the string containing XML content and generate an R-structure using the function `xmlParse` from package **XML**.

```
library(XML)
AM <- xmlParse("amsterdam.osm")
```

This XML-file contains information about nodes, ways, and relations. In a next step, the information has to be extracted. For some OSM objects there is a lot of tagged information available. But some key-value pairs are very seldom, whereas the geocode is available for every object. With the XML Path syntax, it is possible to find XML nodes that match a particular criterion. You can get the right search criterion by looking closer at the downloaded XML file. In the following example the typical OSM key-value pair structure is used to get information about the downloaded node.

```
xpathApply(AM,"//tag[@k = 'population']")
```

In the result we get the population of Amsterdam:

```
[[1]]
<tag k="population" v="844952"/>

attr(,"class")
[1] "XMLNodeSet"
```

The `.osm` files can also be imported using the package **sf**. With the function `st_layers` we can check which layers are available.

```
library(sf)
st_layers("amsterdam.osm")

Driver: OSM
Available layers:
        layer_name       geometry_type features fields
1           points               Point       NA     10
2            lines         Line String       NA      9
3 multilinestrings   Multi Line String       NA      4
4    multipolygons       Multi Polygon       NA     25
5  other_relations Geometry Collection       NA      4
```

In a seond step, the `multipolygons` layer can be imported with the function `st_read`.

```
points_am <- st_read("amsterdam.osm", "points")
```

Sf is the abbreviation for simple features. With the integration of simple features in R, the international standard ISO 19125-1:2004 was implemented. This standard describes how geographical information is handled (Pebesma et al. 2018). A feature might be a tree or a building. A feature may consist of several other features. The same as abouth also works for an `.xml` file. The outcome is the Table 5 with the layer names, the geometry type and the number of fields.

```
library("sf")
st_layers("4290854847.xml")
```

	name	geomtype	fields
1	points	Point	10
2	lines	Line String	9
3	multilinestrings	Multi Line String	4
4	multipolygons	Multi Polygon	25
5	other_relations	Geometry Collection	4

Table 5: Available layers in object 4290854847.xml (Driver: osm)

We can also import `.xml` files with the command`st_read`.

```
centraal <- st_read("4290854847.xml", "points")
```

The result is information of the the node or point. The function `st_read` can also be used to import for example the GeoJSON format. The information in the object `centraal` can be accessed relatively conveniently with the dollar sign. For example, the coordinates result from the following command:

```
centraal$geometry
```

For a query on the first level of the object `centraal` the first lines the geometry type, the dimension, the bounding box and the epsg code results. If we work with simple features we can use the function `st_transform` from package **sf**.

```
Geometry set for 1 feature
geometry type:  POINT
dimension:      XY
bbox:           xmin: 4.90058 ymin: 52.3789 xmax: 4.90058 ymax: 52.3789
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
POINT (4.900581 52.3789)
```

If very large excerpts or the entire planet file are downloaded from OSM, the XML files can become very large. Here the protocolbuffer binary format (PBF) offers an efficient alternative, which will spread further in the future (OpenStreetMap Wiki 2018c). It is for example planned to implement this file format in the **osmdata** package. The XML structure of OSM objects is unfortunately not very intuitive at first glance. Despite these disadvantages, the XML format still has its justification, since the use of XML formats is widespread (see for example Nolan and Lang 2014). In the following I use the command `st_layers` from **sf** to see which layers are available in the downloaded data.

The **osmdata** package provides functionality to download OSM-data using the Overpass API. The data can be imported as simple features and spatial objects (Padgham et al. 2017). With this package, it is possible to download information for one object (node, way or relation) or a collection of objects.

```
library(osmdata)
dat_rw <- opq(bbox = 'Amsterdam') %>%
    add_osm_feature(key = 'railway',
                    value = 'tram_stop') %>%
osmdata_sf()
```

It is obvious and simple to combine the information obtained with **osmdata** with further information to a map. We can e.g. also extract information for other types of highways. In the following that is done with the package **osmplotr**. In a first step a bounding box is generated with the command `getbb` from **osmdata**. The functions `add_osm_feature` and `osmdata_sf` can be used to get the raw XML data from Overpass API.

```
library(osmplotr)
bbox <- getbb("Amsterdam")
dat_pa <- extract_osm_objects(key = 'highway',
                              value = "primary",
                              bbox = bbox)
dat_sa <- extract_osm_objects(key = 'highway',
                              value = "secondary",
                              bbox = bbox)
```

The function `osm_basemap` from package **osmplotr** creates a base OSM plot (Padgham 2017).

```
map <- osm_basemap(bbox = bbox, bg = c("#F5F5DC"))
map <- add_osm_objects(map, dat_pa, col = c("#00008B"))
map <- add_osm_objects(map, dat_sa, col = "green")
# further objects can be added
print_osm_map(map)
```

In the following figure 6, the highways with value cycleway, footway, pedestrian, primary, secondary, tertiary and residential are used to depict the colored lines.

Figure 6: Map of Amsterdam with roads

```
library(tmap)
qtm(dat_pa$geometry, fill = "#8B1A1A")
```

In the following code chunk I use this object to create an interactive map with the aforementioned R-package **mapview**.

```
library(mapview)
mapview(centraal)
```

In the previous section I downloaded an object and named it `amsterdamcentraal.osm`. The `.osm`-file (e.g. for the section visible in Figure 4) can parsed to R with the function `st_read` from package **sf**. With the **sf** package it is possible to import all available layers of the `.osm` file. We can check which layers are available with the function `st_layers`. In this case we also get information on ways and relations:

```
library(sf)
st_layers("amsterdamcentraal.osm")
am_cen <- st_read("amsterdamcentraal.osm", "multipolygons")
```

An outcome is an simple feature collection. We can access the information relatively conveniently with the dollar sign. The first part of the object contains general information about the geometry type, bounding box, and EPSG code:

```
Simple feature collection with 53 features and 25 fields
geometry type:  MULTIPOLYGON
dimension:      XY
bbox:           xmin: 11.48829 ymin: 48.13825 xmax: 11.5604 ymax: 48.14688
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

The second part then lists specific information about the individual features. For example the OSM id and in the last column the geometry we need for visualization. In Table 6 we can see the first rows and some selected columns of the feature table.

osm_id	type	building	geometry
56955	multipolygon	yes	MULTIPOLYGON (((11.54238 48...
<NA>	<NA>	yes	MULTIPOLYGON (((11.54157 48...
<NA>	<NA>	yes	MULTIPOLYGON (((11.54245 48...
<NA>	<NA>	yes	MULTIPOLYGON (((11.54201 48...
<NA>	<NA>	yes	MULTIPOLYGON (((11.54041 48...
<NA>	<NA>	yes	MULTIPOLYGON (((11.54134 48...
<NA>	<NA>	yes	MULTIPOLYGON (((11.54058 48...
<NA>	<NA>	yes	MULTIPOLYGON (((11.54254 48...
<NA>	<NA>	<NA>	MULTIPOLYGON (((11.54276 48...
<NA>	<NA>	yes	MULTIPOLYGON (((11.54038 48...

Table 6: Table of tags for the imported osm section

A comparison between Table 2 and 6 shows that the level of information varies. **sf** simply calls GDAL, which only imports the osm_id value of the containing object but discards values of all members of a MULTIPOLYGON for example. The different degree of information can therefore be traced back to the definition of simple features standard. The **osmdata** package will provide an equivalent version of Table 6 in which all osm_id values will have been retained.

The number of nodes is bounded at 50,000 as well as the size of the area (Roick et al., 2011, p. 4). Subsequently, it is possible to work with the data in the standard R-manner. The data can be converted into available infrastructure provided by existing R-packages, e.g. **sp** and **igraph** objects (Bivand et al. 2013 and Csardi and Nepusz 2006).

```
houses <- am_cen[!is.na(am_cen$building),]
other_tags <- am_cen[!is.na(am_cen$other_tags),]
```

Based on this information maps can be plotted using either base graphics, or the plot functions in package **sp**. The package **tmap** by Tennekes (2018) also offers an intuitive and fast way to visualize this information (see Kolb 2016 for a description). With this package it is possible to create thematic maps with flexibility. In the following, we use the `tm_shape` command from the **tmap**-package. So far we extracted mainly nodes and lines within the specified bounding box. It is also possible to obtain polygones. In this case, we can use for instance building as key. Than we subselect the related objects and we plot them. The result is visible on the right-hand side of figure 7.

```
library(tmap)
tm_shape(houses) + tm_fill("royalblue") + tm_shape(other_tags) + tm_borders("gray")
```

Summary

The OSM project offers a great variety of possibilities to access interesting geo-information. We can combine this data with static maps (e.g. satellite maps) from the Google Maps API. Thus, it is for example possible to create valuable visualizations quite fast with the APIs. And they offer much more possibilities. Very early, there were packages available in R that could be used to process and display geographic information. Especially lately, the number of these packages is increasing.

The map tile servers of the OSM project provide possibilities to work with static maps in R. They also provide the necessary information to create interactive maps using the package **leaflet**. Lately

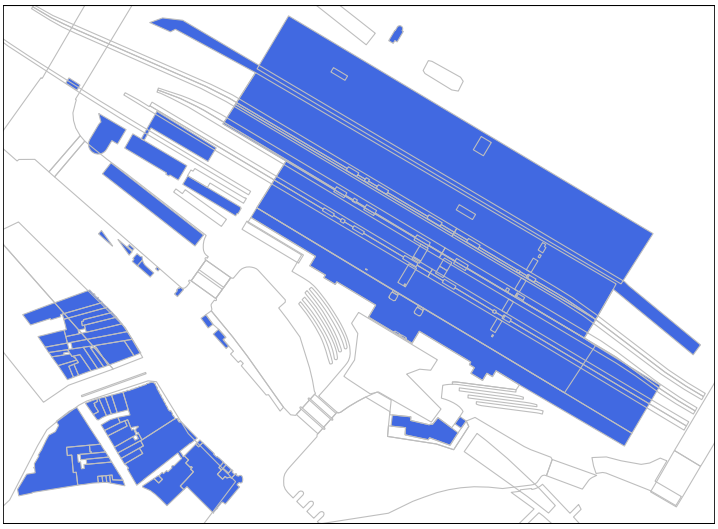


Figure 7: Amsterdam Centraal - two levels of information

many packages have been developped based on Javascript libraries like e.g. **mapdeck** and **deckard**, are a good example of how quickly interactive maps evolve. With these packages it is possible to create many different interactive graphics. These kind of visualizations are developing very fast. Already now there are various fascinating possibilities available, and it is to be expected that soon many more possibilities will be added.

There are packages available in R to realize geocoding using the Nominatim API or the Google Maps API. With the Overpass API, it is possible to obtain useful information for particular points of interest like restaurants, fuel stations, bakeries and so on. The **ggmap** package can be used to combine and visualize this information. The combination of the OSM API's with packages like **osmdata** allows downloading collections of objects. The Overpass API is about downloading information on a specified key-value combination. In contrast, with the main OSM API you can get all the information available for a section.

There are already massive amounts of data available free of charge. And thanks to numerous volunteers, the database is getting better and more comprehensive. So it is very crucial to handle this treasure of information with care. If it is the target to work with big data from OSM, it is highly recommended to install OSM locally and therefore, to download the whole planet file, or an excerpt. Geofabrik (`http://www.geofabrik.de/`) may be used for this (OpenStreetMap Wiki 2017b). It has to be mentioned that the data quality of the information may be heterogeneous and that the completeness may vary, depending on the region (see Kounadi 2009 and Goodchild and Li 2012).

Bibliography

D. Abernathy. *Using Geodata and Geolocation in the Social Sciences: Mapping Our Connected World*. SAGE, 2016. doi: 10.4135/9781473983267. [p9, 10]

A. Aitchison. *Beginning Spatial with SQL Server 2008*. Apress, 2009. [p9]

T. Appelhans, F. Detsch, C. Reudenbach, and S. Woellauer. *mapview: Interactive Viewing of Spatial Data in R*, 2018. URL `https://CRAN.R-project.org/package=mapview`. R package version 2.3.0. [p9]

J. J. Arsanjani, A. Zipf, P. Mooney, and M. Helbich. An introduction to OpenStreetMap in geographic information science: Experiences, research, and applications. In J. J. Arsanjani, A. Zipf, P. Mooney, and M. Helbich, editors, *OpenStreetMap in GIScience*, pages 1–15. Springer, 2015. doi: 10.1007/978-3-319-14280-7_1. [p6]

C. Barrington-Leigh and A. Millard-Ball. The world's user-generated road map is more than 80% complete. *PloS one*, 12(8):e0180698, 2017. doi: 10.1371/journal.pone.0180698. [p6]

C. Barron, P. Neis, and A. Zipf. A comprehensive framework for intrinsic openstreetmap quality analysis. *Transactions in GIS*, 18(6):877–895, 2014. doi: 10.1111/tgis.12073. [p6]

M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. SUMO–simulation of urban mobility: An overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011. [p6]

R. Bivand and A. Gebhardt. Implementing functions for spatial statistical analysis using the R language. *Journal of Geographical Systems*, 2(3):307–317, 2000. doi: 10.1007/pl00011460. [p6]

R. S. Bivand, E. Pebesma, and V. Gomez-Rubio. *Applied spatial data analysis with R*. Springer, NY., second edition. edition, 2013. doi: 10.1007/978-1-4614-7618-4. URL http://www.asdar-book.org/. [p18]

P. Brown. Maps, coordinate reference systems and visualising geographic data with mapmisc. *R-Journal*, 2016. [p11, 12]

S. Chamberlain and J. Hollister. *lawn: Client for 'Turfjs' for 'Geospatial' Analysis*, 2017. URL https://CRAN.R-project.org/package=lawn. R package version 0.4.2. [p9]

J. Cheng and Y. Xie. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2016. URL https://CRAN.R-project.org/package=leaflet. R package version 1.0.1. [p9]

S. Chilton. Crowdsourcing is radically changing the geodata landscape: case study of OpenStreetMap. In *Proceedings of the UK 24th International Cartography Conference*, 2009. [p6]

K. Clemens. Qualitative comparison of geocoding systems using OpenStreetMap data. *International Journal on Advances in Software*, 8(3 & 4), 2015. [p10]

D. Cooley. *mapdeck: Interactive Maps Using 'Mapbox GL JS' and 'Deck.gl'*, 2018a. URL https://CRAN.R-project.org/package=mapdeck. R package version 0.1.0. [p9]

D. Cooley. *googleway: Accesses Google Maps APIs to Retrieve Data and Plot Maps*, 2018b. URL https://CRAN.R-project.org/package=googleway. R package version 2.7.1. [p10]

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.org. [p18]

M. Dorman. *Learning R for geospatial analysis*. Packt Publishing Ltd, 2014. [p7]

I. Fellows. *OpenStreetMap: Access to OpenStreetMap Raster Images*, 2016. URL https://CRAN.R-project.org/package=OpenStreetMap. R package version 0.3.3. [p7]

J. Gaffuri. Toward web mapping with vector data. In *International Conference on Geographic Information Science*, pages 87–101. Springer, 2012. [p7]

L. Gervasoni, S. Fenet, R. Perrier, and P. Sturm. Convolutional neural networks for disaggregated population mapping using open data. In *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2018. [p6]

M. F. Goodchild and L. Li. Assuring the quality of volunteered geographic information. *Spatial statistics*, 1:110–120, 2012. doi: 10.1016/j.spasta.2012.03.002. [p19]

T. Grippa, S. Georganos, S. Zarougui, P. Bognounou, E. Diboulo, Y. Forget, M. Lennert, S. Vanhuysse, N. Mboga, and E. Wolff. Mapping urban land use at street block level using OpenStreetMap, remote sensing data, and spatial metrics. *ISPRS International Journal of Geo-Information*, 7(7):246, 2018. doi: 10.3390/ijgi7070246. [p6]

M. Haklay and P. Weber. OpenStreetMap: User-generated street maps. *IEEE Pervasive Computing*, 7(4): 12–18, 2008. doi: 10.1109/mprv.2008.80. [p13]

I. Hansel. *deckard: Visualise Data with deck.gl*, 2018. R package version 0.0.0.9000. [p9]

R. Harris. *Quantitative Geography: The Basics*. Sage, 2016. doi: 10.4135/9781473920446. [p9]

D. Kahle and H. Wickham. ggmap: Spatial visualization with ggplot2. *The R Journal*, 5(1):144–161, 2013. URL http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf. [p7, 9]

J.-P. Kolb. Visualizing geodata with R. *Austrian Journal of Statistics*, 45(1):45–54, 2016. doi: 10.17713/ajs.v45i1.88. [p18]

O. Kounadi. Assessing the quality of OpenStreetMap data. *Msc geographical information science, University College of London Department of Civil, Environmental And Geomatic Engineering*, 2009. [p19]

O. Kounadi, T. J. Lampoltshammer, M. Leitner, and T. Heistracher. Accuracy and privacy aspects in free online reverse geocoding services. *Cartography and Geographic Information Science*, 40(2):140–153, 2013. doi: 10.1080/15230406.2013.777138. [p13]

O. P. Lamigueiro. *Displaying time series, spatial, and space-time data with R*. CRC Press, 2014. doi: 10.1201/b16713. [p7]

D. T. Lang. *RJSONIO: Serialize R objects to JSON, JavaScript Object Notation*, 2014. URL https://CRAN.R-project.org/package=RJSONIO. R package version 1.3-0. [p10]

D. T. Lang and the CRAN Team. *XML: Tools for Parsing and Generating XML Within R and S-Plus*, 2016. URL https://CRAN.R-project.org/package=XML. R package version 3.98-1.5. [p14]

C. Li, H. Lu, Y. Xiang, Z. Liu, W. Yang, and R. Liu. Bringing geospatial data closer to mobile users: A caching approach based on vector tiles for wireless multihop scenarios. *Mobile Information Systems*, 2018, 2018. doi: 10.1155/2018/5186495. [p7]

M. Loecher and K. Ropkins. RgoogleMaps and loa: Unleashing R graphics power on map tiles. *Journal of Statistical Software*, 63(4):1–18, 2015. doi: 10.18637/jss.v063.i04. URL http://www.jstatsoft.org/v63/i04/. [p9]

R. Lovelace, J. Cheshire, R. Oldroyd, et al. Introduction to visualising spatial data in R, 2017. URL https://cran.r-project.org/doc/contrib/intro-spatial-rl.pdf. [p12]

R. Lovelace, J. Nowosad, and J. Muenchow. *Geocomputation with R*, 2018. URL https://geocompr.robinlovelace.net/. [p9, 15]

G. Maier. OpenStreetMap, the wikipedia map. *Region*, 1(1):3–10, 2014. doi: 10.18335/region.v1i1.70. [p11]

M. Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2011. ISBN 1449319904. [p13]

J. Meijer, M. A. Huijbregts, K. Schotten, and A. Schipper. Global patterns of current and future road infrastructure. *Environmental Research Letters*, 13(6), 2018. doi: 10.1088/1748-9326/aabd42. [p6]

M. Mongiello, T. Di Noia, and E. Di Sciascio. A temporal logic-based approach to query OpenStreetMap. In *23nd Italian Symposium on Advanced Database Systems. SEBD 2015*, 2015. [p14]

P. Mooney and P. Corcoran. Accessing the history of objects in OpenStreetMap. In *Proceedings AGILE*, page 155, 2011. [p13]

A. Moore and I. Drecki. *Geospatial Vision: New Dimensions in Cartography*. Springer Science & Business Media, 2008. ISBN 978-3-540-70970-1. doi: 10.1007/978-3-540-70970-1. [p9]

J. Muenchow and P. Schratz. *RQGIS: Integrating R with QGIS*, 2017. URL https://CRAN.R-project.org/package=RQGIS. R package version 0.2.0. [p6]

D. Nolan and D. T. Lang. *XML and web technologies for data sciences with R*. Springer, 1 edition, 2014. ISBN 9781461478997. doi: 10.1007/978-1-4614-7900-0. [p16]

J. Ooms. The jsonlite package: A practical and consistent mapping between JSON data and R objects. *arXiv:1403.2805 [stat.CO]*, 2014. URL https://arxiv.org/abs/1403.2805. [p10]

J. Ooms. *curl: A Modern and Flexible Web Client for R*, 2018. URL https://CRAN.R-project.org/package=curl. R package version 3.2. [p14]

OpenStreetMap Wiki. Overpass API — OpenStreetMap wiki, 2016a. URL http://wiki.openstreetmap.org/w/index.php?title=Overpass_API&oldid=1405270. [Online; accessed 7-Februar-2017]. [p14]

OpenStreetMap Wiki. Slippy map — OpenStreetMap wiki, 2016b. URL http://wiki.openstreetmap.org/w/index.php?title=Slippy_Map&oldid=1412781. [Online; accessed 2-Februar-2017]. [p9]

OpenStreetMap Wiki. Api usage policy — OpenStreetMap wiki, 2017a. URL http://wiki.openstreetmap.org/w/index.php?title=API_usage_policy&oldid=1538003. [Online; accessed 26-November-2018]. [p14]

OpenStreetMap Wiki. Geofabrik — OpenStreetMap wiki, 2017b. URL http://wiki.openstreetmap.org/w/index.php?title=Geofabrik&oldid=1501676. [Online; accessed 4-March-2018]. [p19]

OpenStreetMap Wiki. OSM XML — OpenStreetMap wiki, 2017c. URL http://wiki.openstreetmap.org/w/index.php?title=OSM_XML&oldid=1419416. [Online; accessed 26-November-2018]. [p14]

OpenStreetMap Wiki. De:overpass turbo — OpenStreetMap wiki, 2017d. URL http://wiki.openstreetmap.org/w/index.php?title=DE:Overpass_turbo&oldid=1485193. [Online; accessed 28-Juni-2018]. [p14]

OpenStreetMap Wiki. Research — OpenStreetMap wiki, 2017e. URL http://wiki.openstreetmap.org/w/index.php?title=Research&oldid=1422855. [Online; accessed 2-Februar-2017]. [p6]

OpenStreetMap Wiki. Nominatim — OpenStreetMap wiki, 2018a. URL http://wiki.openstreetmap.org/w/index.php?title=Nominatim&oldid=1667946. [Online; accessed 27-November-2018]. [p6, 10]

OpenStreetMap Wiki. Downloading data — OpenStreetMap wiki, 2018b. URL http://wiki.openstreetmap.org/w/index.php?title=Downloading_data&oldid=1631037. [Online; accessed 26-November-2018]. [p14]

OpenStreetMap Wiki. Pbf format — OpenStreetMap Wiki, 2018c. URL http://wiki.openstreetmap.org/w/index.php?title=PBF_Format&oldid=1580310. [Online; accessed 12-September-2018]. [p16]

OpenStreetMap Wiki. Vector tiles — OpenStreetMap wiki, 2018d. URL http://wiki.openstreetmap.org/w/index.php?title=Vector_tiles&oldid=1551297. [Online; accessed 27-Oktober-2018]. [p7]

M. Padgham. *osmplotr: Bespoke Images of 'OpenStreetMap' Data*, 2017. URL https://CRAN.R-project.org/package=osmplotr. R package version 0.3.0. [p17]

M. Padgham, B. Rudis, R. Lovelace, and M. Salmon. osmdata. *The Journal of Open Source Software*, 2 (14), jun 2017. doi: https://doi.org/10.21105/joss.00305. [p16]

E. Pebesma and R. Bivand. *Spatial Data Science*, 2019. URL https://www.r-spatial.org/book. [p12]

E. Pebesma, R. Bivand, E. Racine, M. Sumner, I. Cook, T. Keitt, R. Lovelace, H. Wickham, J. Ooms, and K. Müller. *Simple Features for R*, 2018. URL https://cran.r-project.org/web/packages/sf/vignettes/sf1.html. R package version 0.6-3. [p16]

R. E. Plant et al. *Spatial data analysis in ecology and agriculture using R*. cRc Press Boca Raton, Florida, 2012. doi: 10.1201/b11769. [p12]

H. Pruvost and P. Mooney. Exploring data model relations in OpenStreetMap. *Future Internet*, 9(4):70, 2017. [p13]

QGIS Development Team. *QGIS Geographic Information System*. Open Source Geospatial Foundation, 2009. URL http://qgis.osgeo.org. [p6]

F. Ramm, J. Topf, and S. Chilton. *OpenStreetMap: using and enhancing the free map of the world*. UIT Cambridge Cambridge, 2011. [p13]

O. Roick, P. Neis, and A. Zipf. Volunteered Geographic Information – Datenqualität und Nutzungspotentiale am Beispiel von OpenStreetMap. In *Kommission Angewandte Kartographie- Geovisualisierung der Deutschen Gesellschaft für Kartographie (DGfK)- Symposium Königslutter*, 2011. [p18]

S. Schmidt, S. Manschitz, C. Rensing, and R. Steinmetz. Extraction of address data from unstructured text using free knowledge resources. In *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*, page 7. ACM, 2013. doi: 10.1145/2494188.2494193. [p6, 14]

J. P. Stotz. JMapViewer, 2018. URL https://wiki.openstreetmap.org/wiki/JMapViewer. [p7]

G. Svennerberg. *Beginning Google Maps API 3*. Apress, 2010. [p9]

M. Tennekes. tmap: Thematic maps in R. *Journal of Statistical Software*, 84(6):1–39, 2018. doi: https://doi.org/10.18637/jss.v084.i06. [p11, 18]

A. Turner. *Introduction to Neogeography*. O'Reilly Media, 2006. ISBN 9780596529956. URL https://books.google.de/books?id=oHgDv4feV-8C. [p11]

P. Warden. *Data Source Handbook*. O'Reilly Media, Inc., 2011. ISBN 1449303145. [p10]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN 978-0-387-98140-6. doi: 10.1007/978-0-387-98141-3. URL http://ggplot2.org. [p7]

Jan-Philipp Kolb
GESIS Leibniz Institute for the Social Sciences
B2,1 Mannheim
Germany
Jan-Philipp.Kolb@gesis.org

orthoDr: Semiparametric Dimension Reduction via Orthogonality Constrained Optimization

by Ruoqing Zhu, Jiyang Zhang, Ruilin Zhao, Peng Xu, Wenzhuo Zhou and Xin Zhang

Abstract orthoDr is a package in R that solves dimension reduction problems using orthogonality constrained optimization approach. The package serves as a unified framework for many regression and survival analysis dimension reduction models that utilize semiparametric estimating equations. The main computational machinery of **orthoDr** is a first-order algorithm developed by Wen and Yin (2012) for optimization within the Stiefel manifold. We implement the algorithm through Rcpp and OpenMP for fast computation. In addition, we developed a general-purpose solver for such constrained problems with user-specified objective functions, which works as a drop-in version of optim(). The package also serves as a platform for future methodology developments along this line of work.

Introduction

Dimension reduction is a long-standing problem in statistics and data science. While the traditional principal component analysis (Jolliffe, 1986) and related works provide a way of reducing the dimension of the covariates, the term "sufficient dimension reduction" is more commonly referring to a series of regression works originated from the seminal paper on sliced inverse regression (Li, 1991). In such problems, we observe an outcome $Y \in \mathbb{R}$, along with a set of covariates $X = (X_1, \ldots, X_p)^{\mathrm{T}} \in \mathbb{R}^p$. Dimension reduction models are interested in modeling the conditional distribution of Y given X, while their relationship satisfies, for some $p \times d$ matrix $\mathbf{B} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_p)$,

$$Y = h(X, \epsilon) = h(\mathbf{B}^{\mathrm{T}} X, \epsilon) = h(\boldsymbol{\beta}_1^{\mathrm{T}} X, \ldots, \boldsymbol{\beta}_d^{\mathrm{T}} X, \epsilon), \tag{1}$$

where ϵ represents any error terms and h, with a slight abuse of notation, represents the link function using X or $\mathbf{B}^{\mathrm{T}} X$. One can easily notice that when d, the number of columns in $\mathbf{B}$, is less than p, a dimension reduction is achieved, in the sense that only a d dimensional covariate information is necessary for fully describing the relationship (Cook, 2009). Alternatively, this relationship can be represented as (Zeng and Zhu, 2010)

$$Y \perp X \mid \mathbf{B}^{\mathrm{T}} X, \tag{2}$$

which again describes the sufficiency of $\mathbf{B}^{\mathrm{T}} X$. Following the work of Li (1991), a variety of methods have been proposed. An incomplete list of literature includes Cook and Weisberg (1991); Cook and Lee (1999); Yin and Cook (2002); Chiaromonte et al. (2002); Zhu et al. (2006); Li and Wang (2007); Zhu et al. (2010b,a); Cook et al. (2010); Lee et al. (2013); Cook and Zhang (2014); Li and Zhang (2017). For a more comprehensive review of the literature, we refer the readers to Ma and Zhu (2013b). One advantage of many early developments in dimension reduction models is that only a singular value decomposition is required to obtain the reduced space parameters $\mathbf{B}$ through inverse sliced averaging. However, this comes at a price of assuming the linearity assumption (Li, 1991), which is almost the same as assuming that the covariates follow an elliptical distribution (Li and Dong, 2009; Dong and Li, 2010). Moreover, some methods require more restrictive assumptions on the covariance structure (Cook and Weisberg, 1991). Many methods attempt to avoid these assumptions by resorting to nonparametric estimations. The most successful ones include Xia et al. (2002) and Xia (2007). However, recently a new line of work started by Ma and Zhu (2012b,a, 2013a) shows that by formulating the problem into semiparametric estimating equations, not only we can avoid many distributional assumptions on the covariates, the obtained estimator of $\mathbf{B}$ also enjoys efficiency. Extending this idea, Sun et al. (2017) developed a framework for dimension reduction in survival analysis using a counting process based estimating equations. The method performs significantly better than existing dimension reduction methods for censored data such as Li et al. (1999); Xia et al. (2010) and Lu and Li (2011). Another recent development that also utilizes this semiparametric formulation is Zhao et al. (2017), in which an efficient estimator is derived.

Although there are celebrated theoretical and methodological advances, estimating $\mathbf{B}$ through the semiparametric estimating equations is still not a trivial task. Two challenges remain: first, by a careful look at the model definition 1, we quickly noticed that the parameters are not identifiable unless certain constraints are placed. In fact, if we let $\mathbf{A}$ be any $d \times d$ full rank matrix, then $(\mathbf{BA})^{\mathrm{T}} X$ preserves the same

column space information of $\mathbf{B}^{\mathrm{T}}X$, hence, we can define $h^*((\mathbf{BA})^{\mathrm{T}}X, \epsilon)$ accordingly to retain exactly the same model as (1). While traditional methods can utilize singular value decompositions (SVD) of the estimation matrix to identify the column space of $\mathbf{B}$ instead of recovering each parameter (Cook and Lee, 1999), it appears to be a difficult task in the semiparametric estimating equation framework. One challenge is that if we let $\mathbf{B}$ change freely, the rank of the $\mathbf{B}$ matrix cannot be guaranteed, which makes the formulation meaningless. Hence, for both computational and theoretical concerns, Ma and Zhu (2012b) resorts to an approach that fixes the upper $d \times d$ block of $\mathbf{B}$ as an identity matrix, i.e., $\mathbf{B} = (\mathbf{I}_{d\times d}, \mathbf{B}^{*\mathrm{T}})^{\mathrm{T}}$, where $\mathbf{B}^*$ is a $(p-d)\times d$ matrix that sits in the lower block of $\mathbf{B}$. Hence, in this formulation, only $\mathbf{B}^*$ needs to be solved. While the solution is guaranteed to be rank d in this formulation, as pointed out by Sun et al. (2017), this approach still requires correctly identifying and reordering of the covariate vector $\mathbf{x}$ such that the first d entries are indeed important, which creates another daunting task. Another challenge is that solving semiparametric estimating equations requires the estimation of nonparametric components. These components need to be computed through kernel estimations, usually the Nadaraya-Watson type, which significantly increases the computational intensity of the method considering that these components need to be recalculated at each iteration of the optimization. Up to date, these drawbacks remain as the strongest criticism of the semiparametric approaches. Hence, although enjoying superior statistical asymptotic properties, are not as attractive as a traditional sliced inverse type of approaches such as Li (1991) and Cook and Weisberg (1991).

The goal of our **orthoDr** package is to develop a computationally efficient optimization platform for solving the semiparametric estimating equation approaches proposed in Ma and Zhu (2013a), Sun et al. (2017) and possibly any future work along this line. Revisiting the rank preserving problem of $\mathbf{B}$ mentioned above, we can essentially set a constraint that

$$\mathbf{B}^{\mathrm{T}}\mathbf{B} = \mathbf{I}, \tag{3}$$

where $\mathbf{I}$ is a $d \times d$ identity matrix. A solution of the estimating equations that satisfies the constraint will correctly identify the dimensionality-reduced subspace. This is known as optimizing on the Stiefel manifold, which is a class of well-studied problems (Edelman et al., 1998). A recent R development (Martin et al., 2016) utilizes quasi-Newton methods such as the well known BFGS method on the Riemannian manifold (Huang et al., 2018). However, second order optimization methods always require forming and storing large hessian matrices. In addition, they may not be easily adapted to penalized optimization problems, which often appear in high dimensional statistical problems Zhu et al. (2006); Li and Yin (2008). On the other hand, first-order optimization methods are faster in each iteration, and may also incorporate penalization in a more convenient way Wen et al. (2010). By utilizing the techniques developed by Wen and Yin (2012), we can effectively search for the solution in the Stiefel manifold, and this becomes the main machinery of our package. Further incorporating the popular **Rcpp** (Eddelbuettel and François, 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson, 2014) toolboxes and the OpenMP parallel commuting, the computational time for our package is comparable to state-of-the-art existing implementations (such as **ManifoldOpthm**), making the semiparametric dimension reduction models more accessible in practice.

The purpose of this article is to provide a general overview of the **orthoDr** package (version 0.6.2) and provide some concrete examples to demonstrate its advantages. **orthoDr** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=orthoDr` and GitHub at `https://github.com/teazrq/orthoDr`. We begin by explaining the underlying formulation of the estimating equation problem and the parameter updating scheme that preserves orthogonality. Next, the software is introduced in detail using simulated data and real data as examples. We further demonstrate an example that utilizes the package as a general purpose solver. We also investigate the computational time of the package compared with existing solvers. Future plans for extending the package to other dimension reduction problems are also discussed.

Model description

Counting process based dimension reduction

To give a concrete example of the estimating equations, we use the semiparametric inverse regression approach defined in Sun et al. (2017) to demonstrate the calculation. Following the common notations in the survival analysis literature, let X_i be the observed p dimensional covariate values of subject i, $Y_i = \min(T_i, C_i)$ is the observed survival time, with failure time T_i and censoring time C_i, and $\delta_i = I(T_i \leq C_i)$ is the censoring indicator. A set of i.i.d. observations $\{X_i, Y_i, \delta_i\}_{i=1}^n$ is observed. We are interested in a situation that the conditional distribution of failure time $T_i|X_i$ depends only on the

reduced space $\mathbf{B}^{\mathrm{T}}X_i$. Hence, to estimate $\mathbf{B}$, the estimating equation is given by

$$\widehat{\psi}_n(\mathbf{B}) = \mathrm{vec}\left[\frac{1}{n}\sum_{i=1}^{n}\sum_{\substack{j=1\\ \delta_j=1}}^{n}\left\{X_i - \widehat{E}\big(X|Y \geq Y_j, \mathbf{B}^{\mathrm{T}}X_i\big)\right\}\widehat{\varphi}^{\mathrm{T}}(Y_j)\left\{\delta_i I(j=i) - \widehat{\lambda}\big(Y_j|\mathbf{B}^{\mathrm{T}}X_i\big)\right\}\right], \tag{4}$$

where the operator $\mathrm{vec}(\cdot)$ is the vectorization of matrix. Several components are estimated nonparasitically: the function $\widehat{\varphi}(u)$ is estimated by sliced averaging,

$$\widehat{\varphi}(u) = \frac{\sum_{i=1}^{n} X_i I\big(u \leq Y_i < u + \triangle u, \delta_i = 1\big)}{\sum_{i=1}^{n} I\big(u \leq Y_i < u + \triangle u, \delta_i = 1\big)} - \frac{\sum_{i=1}^{n} X_i I\big(Y_i \geq u\big)}{\sum_{i=1}^{n} I\big(Y_i \geq u\big)}, \tag{5}$$

where $\triangle u$ is chosen such that there are hn number of observations lie between u and $u + \triangle u$. The conditional mean function $\widehat{E}\big(X|Y \geq u, \mathbf{B}^{\mathrm{T}}X = z\big)$ is estimated through the Nadaraya-Watson kernel estimator

$$\widehat{E}\big(X|Y \geq u, \mathbf{B}^{\mathrm{T}}X = z\big) = \frac{\sum_{i=1}^{n} X_i K_h(\mathbf{B}^{\mathrm{T}}X_i - z) I(Y_i \geq u)}{\sum_{i=1}^{n} K_h(\mathbf{B}^{\mathrm{T}}X_i - z) I(Y_i \geq u)}. \tag{6}$$

In addition, the the conditional hazard function at any time point u can be estimated by

$$\widehat{\lambda}(u|\mathbf{B}^{\mathrm{T}}X = z) = \frac{\sum_{i=1}^{n} K_b(Y_i - u)\delta_i K_h\big(\mathbf{B}^{\mathrm{T}}X_i - z\big)}{\sum_{j=1}^{n} I\big(Y_j \geq u\big) K_h\big(\mathbf{B}^{\mathrm{T}}X_j - z\big)}. \tag{7}$$

However, this substantially increase the computational burden since the double kernel estimator requires $\mathcal{O}(n^2)$ flops to calculate the hazard at any given u and z. Instead, an alternative version using Dabrowska (1989) can greatly reduce the computational cost without compromising the performance. Hence, we estimate the conditional hazard function by

$$\widehat{\lambda}(u|\mathbf{B}^{\mathrm{T}}X = z) = \frac{\sum_{i=1}^{n} I\big(Y_i = u\big) I\big(\delta_i = 1\big) K_h\big(\mathbf{B}^{\mathrm{T}}X_i - z\big)}{\sum_{i=1}^{n} I\big(Y_i \geq u\big) K_h\big(\mathbf{B}^{\mathrm{T}}X_i - z\big)}, \tag{8}$$

which requires only $\mathcal{O}(n)$ flops. In the above equations (5), (6) and (8), h is a pre-specified kernel bandwidth and $K_h(\cdot) = K(\cdot/h)/h$, where $K(\cdot)$ is the Gaussian kernel function. By utilizing the method of moments estimators (Hansen, 1982) and noticing our constraint for identifying the column space of $\mathbf{B}$, solving for the solution of the estimating equations (4) is equivalent to

$$\text{minimize} \quad f(\mathbf{B}) = \widehat{\psi}_n(\mathbf{B})^{\mathrm{T}}\widehat{\psi}_n(\mathbf{B}) \tag{9}$$

$$\text{subject to} \quad \mathbf{B}^{\mathrm{T}}\mathbf{B} = \mathbf{I}. \tag{10}$$

Essentially all other semiparametric dimension reduction models described in Ma and Zhu (2013a), and more recently Ma and Zhang (2015) Xu et al. (2016), Sun et al. (2017), Huang and Chiang (2017) and many others can be estimated in the samimilar fashion as the above optimization problem. However, due to the difficult in the constrains and the purpose of identifiability, all of these methods resort to either fixing the upper block of the $\mathbf{B}$ matrix as an identity matrix or adding a penalty of $\|\mathbf{B}^{\mathrm{T}}\mathbf{B} - \mathbf{I}\|_F$ to preserve the orthogonality constraint. There appears to be no existing method that solves (9) directly. Here, we utilize Wen and Yin (2012)'s approach which can effectively tackle this problem.

Orthogonality preserving updating scheme

The algorithm works in the same fashion as a regular gradient decent, except that we need to preserve the orthogonality at each iteration of the update. As described in Wen and Yin (2012), given any feasible point $\mathbf{B}_0$, i.e., $\mathbf{B}_0{}^{\mathrm{T}}\mathbf{B}_0 = \mathbf{I}$, which can always be generated randomly, we update $\mathbf{B}_0$ as follows. Let the $p \times d$ gradient matrix be

$$\mathbf{G} = \left(\frac{\partial f(\mathbf{B}_0)}{\partial \mathbf{B}_0(i,j)}\right)_{\{i,j\}}. \tag{11}$$

Then, utilizing the Cayley transformation, we have

$$\mathbf{B}_{\mathrm{new}} = \Big(\mathbf{I} + \frac{\tau}{2}\mathbf{A}\Big)^{-1}\Big(\mathbf{I} - \frac{\tau}{2}\mathbf{A}\Big)\mathbf{B}_0, \tag{12}$$

with the orthogonality preserving property $\mathbf{B}_{\mathrm{new}}^{\mathrm{T}}\mathbf{B}_{\mathrm{new}} = \mathbf{I}$. Here, $\mathbf{A} = \mathbf{G}\mathbf{B}_0{}^{\mathrm{T}} - \mathbf{B}_0\mathbf{G}^{\mathrm{T}}$ is a skew-symmetric matrix. It can be shown that $\{\mathbf{B}_{\mathrm{new}}(\tau)\}_{\tau \geq 0}$ is a descent path. Similar to line search

algorithms, we can then find a proper step size τ through a curvilinear search. Recursively updating the current value of $\mathbf{B}$, the algorithm stops when the tolerance level is reached. An initial value is also important for the performance of nonconvex optimization problems. A convenient initial value for our framework is the computational efficient approach developed in Sun et al. (2017), which only requires a SVD of the estimation matrix.

The R package orthoDr

There are several main functions in the **orthoDr** package: `orthoDr_surv`, `ortho_reg` and `ortho_optim`. They are corresponding to the survival model described perviously (Sun et al., 2017), the regression model in Ma and Zhu (2012b), and a general constrained optimization function, respectively. In this section, we demonstrate the details of using these main functions, illustrate them with examples.

Semiparametric dimension reduction models for survival data

The `orthoDr_surv` function implements the optimization problem defined in Equation (9), where the kernel estimations and various quantities are implemented and calculated within C++. Note that in addition, the method defined previously, some simplified versions are also implemented such as the counting process inverse regression models and the forward regression models, which are all described in Sun et al. (2017). These specifications can be made using the `method` parameter. A routine call of the function `orthoDr_surv` proceed as

```
orthoDr_surv(x, y, censor, method, ndr, B.initial, bw, keep.data,
             control, maxitr, verbose, ncore)
```

- `x`: A matrix or data.frame for features (numerical only).
- `y`: A vector of observed survival times.
- `censor`: A vector of censoring indicators.
- `method`: The estimating equation method used.
  - `"dm"` (default): semiparametric inverse regression given in (4).
  - `"dn"`: counting process inverse regression.
  - `"forward"`: forward regression model with one structural dimensional.
- `ndr`: The number of structural dimensional. For `method = "dn"` or `"dm"`, the default is 2. For `method = "forward"` only one structural dimension is allowed, hence the parameter is suppressed.
- `B.initial`: Initial `B` values. Unless specifically interested, this should be left as default, which uses the computational efficient approach (with the `CPSIR()` function) in Sun et al. (2017) as the initial. If specified, must be a matrix with `ncol(x)` rows and `ndr` columns. The matrix will be processed by Gram-Schmidt if it does not satisfy the orthogonality constrain.
- `bw`: A kernel bandwidth, assuming each variables have unit variance. By default we use the Silverman rule-of-thumb formula Silverman (1986) to determine the bandwidth

$$\texttt{bw} = 1.06 \times \left(\frac{4}{d+2}\right)^{\frac{1}{d+4}} n^{-\frac{1}{d+4}}.$$

  This bandwidth can be computed using the `silverman(n,d)` function in our package.
- `keep.data`: Should the original data be kept for prediction? Default is `FALSE`.
- `control`: A list of tuning variables for optimization, including the convergence criteria. In particular, `epsilon` is the size for numerically approximating the gradient, `ftol`, `gtol`, and `btol` are tolerance levels for the objective function, gradients, and the parameter estimations, respectively, for judging the convergence. The default values are selected based on Wen and Yin (2012) .
- `maxitr`: Maximum number of iterations. Default is 500.
- `verbose`: Should information be displayed? Default is `FALSE`.
- `ncore`: Number of cores for parallel computing when approximating the gradients numerically. The default is the maximum number of threads.

We demonstrate the usage of `orthoDr_surv` function by solving a problem with generated survival data.

```
# generate some survival data with two structural dimensions
R> set.seed(1)
R> N = 350; P = 6; dataX = matrix(rnorm(N*P), N, P)
R> failEDR = as.matrix(cbind(c(1, 1, 0, 0, 0, 0, rep(0, P-6)),
 +  c(0, 0, 1, -1, 0, 0, rep(0, P-6))))
R> censorEDR = as.matrix(c(0, 1, 0, 1, 1, 1, rep(0, P-6)))
R> T = exp(-2.5 + dataX %*% failEDR[,1] + 0.5*(dataX %*%
 +  failEDR[,1])*(dataX %*% failEDR[,2]) + 0.25*log(-log(1-runif(N))))
R> C = exp( -0.5 + dataX %*% censorEDR  + log(-log(1-runif(N))))
R> Y = pmin(T, C)
R> Censor = (T < C)

# fit the model
R> orthoDr.fit = orthoDr_surv(dataX, Y, Censor, ndr = 2)
R> orthoDr.fit

             [,1]        [,2]
[1,] -0.689222616  0.20206497
[2,] -0.670750726  0.19909057
[3,] -0.191817963 -0.66623300
[4,]  0.192766630  0.68605407
[5,]  0.005897188  0.02021414
[6,]  0.032829356  0.06773089
```

To evaluate the accuracy of this estimation, a distance function `distance()` can be used. This function calculates the distance between the column spaces generated by the true $\mathbf{B}$ and the estimated version $\widehat{\mathbf{B}}$. Note that the sine angle distance between the two column spaces is closely related to the canonical correlation between the two matrices $\mathbf{B}$ and $\widehat{\mathbf{B}}$.

```
distance(s1, s2, method, x)
```

- `s1`: A matrix for the first column space (e.g., $\mathbf{B}$).
- `s2`: A matrix for the second column space (e.g., $\widehat{\mathbf{B}}$).
- `method`:
  - `"dist"`: the Frobenius norm distance between the projection matrices of the two given matrices, where for any given matrix $\mathbf{B}$, the projection matrix $\mathbf{P} = \mathbf{B}(\mathbf{B}^{\mathrm{T}}\mathbf{B})^{-1}\mathbf{B}^{\mathrm{T}}$.
  - `"trace"`: the trace correlation between two projection matrices $\mathrm{tr}(\mathbf{P}\widehat{\mathbf{P}})/d$, where d is the number of columns of the given matrix.
  - `"canonical"`: the canonical correlation between $\mathbf{B}^{\mathrm{T}}X$ and $\widehat{\mathbf{B}}^{\mathrm{T}}X$.
  - `"sine"`: the sine angle distance $\|\sin\Theta\|_F$ obtained from $\mathbf{P}_1(\mathbf{I}-\mathbf{P}_2) = \mathbf{U}\sin\Theta\mathbf{V}^{\mathrm{T}}$.
- `x`: The design matrix X (default = `NULL`), required only if `method` $=$ `"canonical"` is used.

We compare the accuracy of the estimations obtained by the `method ="dm"` and `"dn"`. Note that the `"dm"` method enjoys double robustness property of the estimating equations, hence the result is usually better.

```
# Calculate the distance to the true parameters
R> distance(failEDR, orthoDr.fit$B, "dist")

[1] 0.1142773

# Compare with the counting process inverse regression model
R> orthoDr.fit1 = orthoDr_surv(dataX, Y, Censor, method = "dn", ndr = 2)
R> distance(failEDR, orthoDr.fit1$B, "dist")

[1] 0.1631814
```

Semiparametric dimension reduction models for regression

The `orthoDr_reg` function implements the semiparametric dimension reduction methods proposed in Ma and Zhu (2012b). A routine call of the function `orthoDr_reg` proceed as

```
orthoDr_reg(x, y, method, ndr, B.initial, bw, keep.data, control,
            maxitr, verbose, ncore)
```

- `x`: A matrix or data.frame for features (numerical only).
- `y`: A vector of observed continuous outcome.
- `method`: We currently implemented two methods: the semiparametric sliced inverse regression method (`"sir"`), and the semiparametric principal Hessian directions method (`"phd"`).
  - `"sir"`: semiparametric sliced inverse regression method solves the sample version of the estimating equation

$$E\Big(\big[E(X|Y) - E\{E(X|Y)|\mathbf{B}^{\mathrm{T}}X\}\big]\big[X - E(X|\mathbf{B}^{\mathrm{T}}X)\big]^{\mathrm{T}}\Big) = 0$$

  - `"phd"`: semiparametric principal Hessian directions method that estimates $\mathbf{B}$ by solving the sample version of

$$E\big[\{Y - E(Y|\mathbf{B}^{\mathrm{T}}X)\}\{XX^{\mathrm{T}} - E(XX^{\mathrm{T}}|\mathbf{B}^{\mathrm{T}}X)\}\big] = 0$$

- `ndr`: The number of structural dimensional (default is 2).
- `B.initial`: Initial `B` values. For each method, the initial values are taken from the corresponding traditional inverse regression approach using the **dr** package. The obtained matrix will be processed by Gram-Schmidt for orthogonality.
- `bw`, `keep.data`, `control`, `maxitr`, `verbose` and `ncore` are exactly the same as those in the `orthoDr_surv` function.

To demonstrate the usage of `orthoDr_reg`, we consider the problem of dimension reduction by fitting a semi-PHD model proposed by Ma and Zhu (2012b).

```
R> set.seed(1)
R> N = 100; P = 4; dataX = matrix(rnorm(N*P), N, P)
R> Y = -1 + dataX[,1] + rnorm(N)
R> orthoDr_reg(dataX, Y, ndr = 1, method = "phd")

Subspace for regression model using phd approach:
            [,1]
[1,]  0.99612339
[2,]  0.06234337
[3,] -0.04257601
[4,] -0.04515279
```

Parallelled gradient approximation through OpenMP

The estimation equations of the dimension reduction problem in the survival and regression settings usually have a complicated form. Especially, multiple kernel estimations are involved, which results in difficulties in taking derivatives analytically. As an alternative, numerically approximated gradients are implemented using OpenMP. A comparison between a single core and multiple cores (4 cores) is given in the following example. Results from 20 independent simulation runes are summarized in Table 1. The data generating procedure used in this example is the same as the survival data used in Section Semiparametric dimension reduction models for survival data. All simulations are performed on an i7-4770K CPU.

```
R> t0 = Sys.time()
R> dn.fit = orthoDr_surv(dataX, Y, Censor, method = "dn", ndr = ndr,
 + ncore = 4, control = list(ftol = 1e-6))
R> Sys.time() - t0
```

Table 1: Computational cost of different numbers of cores

	# of cores	
	1	4
$n=350$, $p=6$	3.9831	1.2741
$n=350$, $p=12$	12.7780	3.4850

General solver for orthogonality constrained optimization

`ortho_optim` is a general purpose optimization function that can incorporate any user defined objective function f (and gradient function if supplied). The usage of `ortho_optim` is similar to the widely used `optim()` function. A routine call of the function proceed as

```
ortho_optim(B, fn, grad, ..., maximize, control, maxitr, verbose)
```

- `B`: Initial `B` values. Must be a matrix, and the columns are subject to the orthogonality constrains. It will be processed by Gram-Schmidt if not orthogonal.
- `fn`: A function that calculates the objective function value. The first argument should be `B`. Returns a single value.
- `grad`: A function that calculate the gradient. The first argument should be `B`. Returns a matrix with the same dimension as `B`. If not specified, a numerical approximation is used.
- `...`: Arguments passed to `fn` and `grad` besides `B`.
- `maximize`: By default, the solver will try to minimize the objective function unless `maximize = TRUE`.
- The parameters `maxitr`, `verbose` and `ncore` works in the same way as introduced in the previous sections.

To demonstrate the simple usage of `ortho_optim` as a drop-in function of `optim()`, we consider the problem of searching for the first principle component for a data matrix.

```
# an example of searching for the first principal component
R> set.seed(1)
R> N = 400; P = 100; X = scale(matrix(rnorm(N*P), N, P), scale = FALSE)
R> w = gramSchmidt(matrix(rnorm(P), P, 1))$Q
R> fx <- function(w, X) t(w) %*% t(X) %*% X %*% w
R> gx <- function(w, X) 2*t(X) %*% X %*% w

# fit the model
R> fit = ortho_optim(w, fx, gx, X = X, maximize = TRUE, verbose = 0)
R> head(fit$B)

            [,1]
[1,]  0.01268226
[2,] -0.09065592
[3,] -0.01471700
[4,]  0.10583958
[5,] -0.02656409
[6,] -0.04186199

# compare results with the prcomp() function
R> library(pracma)
R> distance(fit$B, as.matrix(prcomp(X)$rotation[, 1]), type = "dist")

[1] 1.417268e-05
```

The **ManifoldOptim** (Martin et al., 2016) package is known for solving optimization problems on manifolds. We consider the problem of optimizing Brockett cost function (Huang et al., 2018) on the Stiefel manifold with objective and gradient functions written in R. The problem can be stated as

$$\min_{\mathbf{B}^{\mathrm{T}}\mathbf{B}=\mathbf{I}_p,\ \mathbf{B}\in\mathbb{R}^{n\times p}} \operatorname{trace}(\mathbf{B}^{\mathrm{T}}X\mathbf{B}D), \tag{13}$$

where $X \in \mathbb{R}^{n\times n}$, $X = X^{\mathrm{T}}$, $D = \operatorname{diag}(\mu_1, \mu_2, ..., \mu_p)$ with $\mu_1 \geq \mu_2 \geq ... \geq \mu_p$. We generate the data with exactly the same procedure as the documentation file provided in the **ManifoldOptim** package, with only a change of notation. For our **orthoDr** package, the following code is used to specify the objective and gradient functions and solve for the optimal $\mathbf{B}$.

```
R> n = 150; p = 5; set.seed(1)

R> X <- matrix(rnorm(n*n), nrow=n)
R> X <- X + t(X)
```

```
R> D <- diag(p:1, p)

R> f1 <- function(B, X, D) { Trace( t(B) %*% X %*% B %*% D ) }
R> g1 <- function(B, X, D) { 2 * X %*% B %*% D }

R> b1 = gramSchmidt(matrix(rnorm(n*p), nrow=n, ncol=p))$Q
R> res2 = ortho_optim(b1, fn = f1, grad = g1, X, D)
R> head(res2$B)

             [,1]         [,2]         [,3]        [,4]         [,5]
[1,] -0.110048632 -0.060656649 -0.001113691 -0.03451514 -0.063626067
[2,] -0.035495670 -0.142148873 -0.011204859  0.01784039  0.129255824
[3,]  0.052141162  0.015140614 -0.034893426  0.02600569  0.006868275
[4,]  0.151239722 -0.008553174 -0.096884087  0.01398827  0.132756189
[5,] -0.001144864 -0.056849007  0.080050182  0.23351751 -0.007219738
[6,] -0.140444290 -0.112932425  0.082197835  0.18644089 -0.057003273
```

Furthermore, we compare the performence with the **ManifoldOptim** package, using four optimization methods: `"LRBFGS"`, `"LRTRSR1"`, `"RBFGS"` and `"RTRSR1"` (Huang et al., 2018). We wrote the same required functions for the Brockett problem in R. Further more, note that different algorithms implements slightly different stoping criterion, we run each algorithm a fixed number of iterations with a single core. We consider three smaller settings with $n = 150$, and $p = 5, 10$ and 15, and a larger setting with $n = 500$ and $p = 50$. Each simulation is repeated 100 times. The functional value progression (Figures 1 and 2) and the total time cost up to a certain number of iterations (Table 2) are presented.

We found that `"LRBFGS"` and our **orthoDr** package usually achieve the best performance, with functional value decreases the steepest in the log scale. In terms of computing time, `"LRBFGS"` and **orthoDr** performers similarly. Although `"LRTRSR1"` has similar computational time, its functional value falls behind. This is mainly because the theoretical complexity of second-order algorithms is similar to first order algorithms, both are of order $\mathcal{O}(p^3)$. However, it should be noted that for a semiparametric dimension reduction method, the major computational cost is not due to the parameter updates, rather, it is calculating the gradient since complicated kernel estimations are involved. Hence, we believe there is no significant advantage using either `"LRBFGS"` or our **orthoDr** package regarding the efficiency of the algorithm. However, first order algorithms may have an advantage when developing methods for penalized high-dimensional models.

Table 2: Running times with a fixed number of iterations (in seconds)

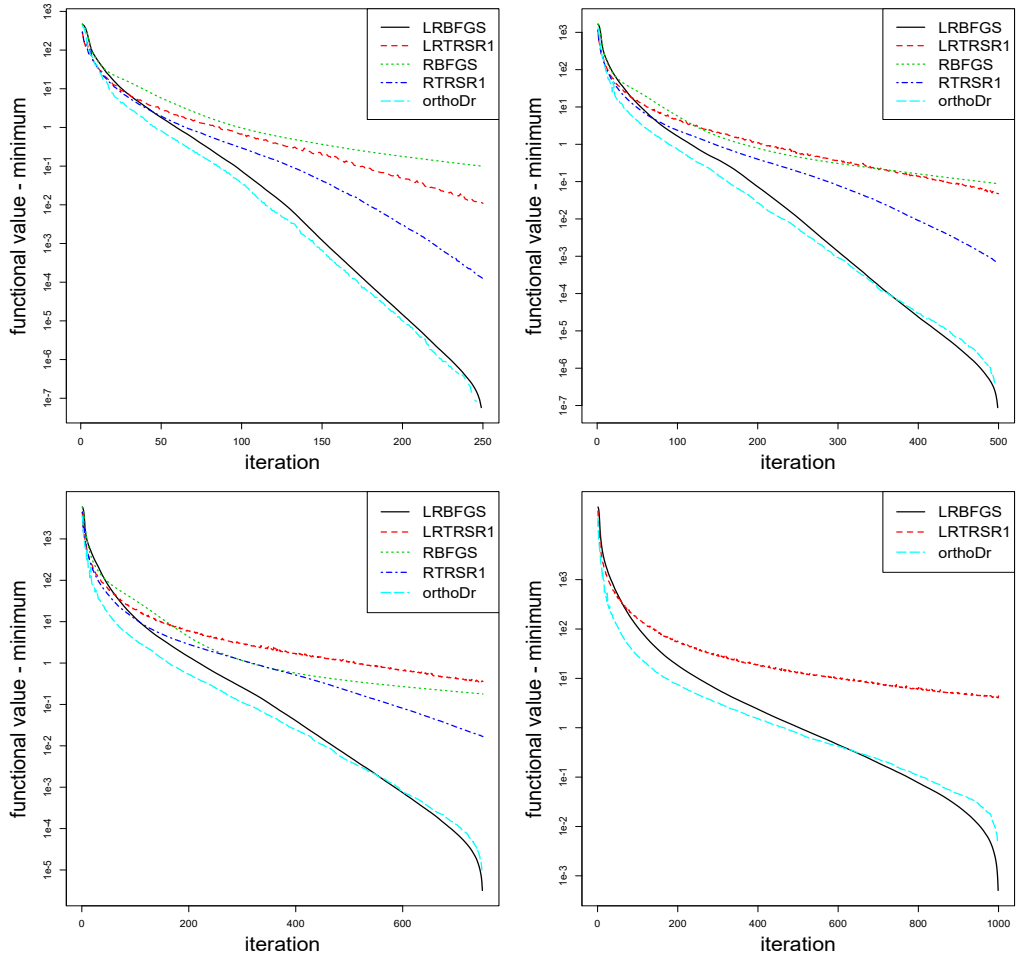
n	p	iteration	**ManifoldOpthm**				**orthoDr**
			LRBFGS	LRTRSR1	RBFGS	RTRSR1	
150	5	250	0.053	0.062	0.451	0.452	0.065
150	10	500	0.176	0.201	4.985	5.638	0.221
150	20	750	0.526	0.589	28.084	36.142	0.819
150	50	1000	2.469	2.662	–	–	6.929
500	5	250	0.403	0.414	7.382	7.426	0.423
500	10	500	1.234	1.305	57.047	67.738	1.332
500	20	750	3.411	3.6	–	–	3.974
500	50	1000	13.775	14.43	–	–	19.862

Examples

We use the *Concrete Compressive Strength* (Yeh, 1998) dataset as an example to further demonstrate the `orthoDr_reg` function and to visualize the results. The dataset is obtained from the UCI Machine Learning Repository.

Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients. These ingredients include cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate. In this dataset, we have $n = 1030$ observation, 8 quantitative input variables, and 1 quantitative output variable. We present the estimated two directions for structural dimension and further plot the observed data in these two directions. A non-parametric kernel estimation surface is further included to approximate the mean concrete strength.

Figure 1: Log of function value vs. iteration ($n = 150$)



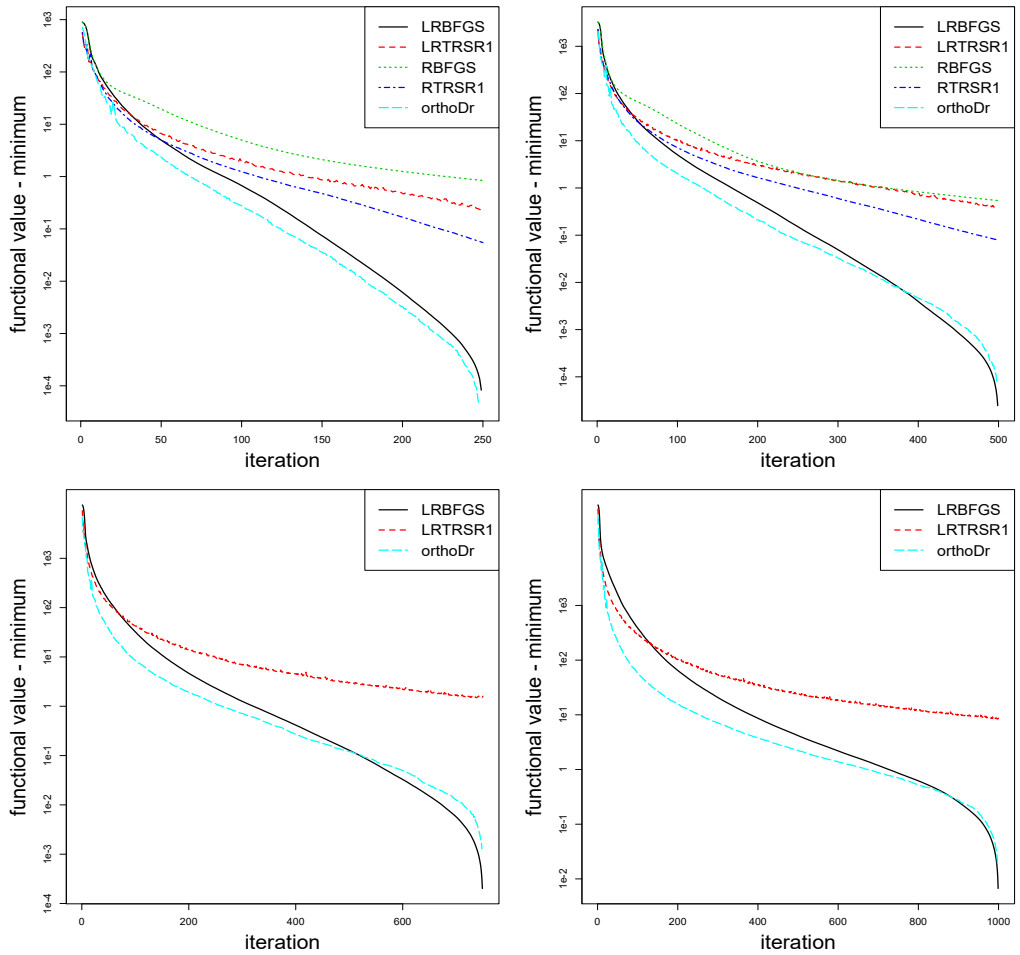
From left to right, top to bottom: $p = 5, 10, 20$ and 50 respectively.

```
R> concrete_data = read.csv(choose.files())
R> X = as.matrix(concrete_data[,1:8])
R> colnames(X) = c("Cement", "Blast Furnace Slag", "Fly Ash", "Water",
            "Superplasticizer", "Coarse Aggregate", "Fine Aggregate", "Age")
R> Y = as.matrix(concrete_data[,9])

R> result = orthoDr_reg(X, Y, ndr = 2, method = "sir", maxitr = 1000,
 + keep.data = TRUE)
R> rownames(result$B) = colnames(X)
R> result$B

                        [,1]        [,2]
Cement              0.08354280 -0.297899899
Blast Furnace Slag  0.27563507  0.320304097
Fly Ash             0.82665328 -0.468889856
Water               0.20738201  0.460314093
Superplasticizer    0.43496780  0.540733516
Coarse Aggregate    0.01141892  0.011870495
Fine Aggregate      0.02936740 -0.004718979
Age                 0.02220664 -0.290444936
```

Discussion

Using the algorithm proposed by Wen and Yin (2012) for optimization on the Stiefel manifold, we developed the **orthoDr** package that serves specifically for semi-parametric dimension reductions

Figure 2: Log of function value vs. iteration ($n = 500$)



From left to right, top to bottom: $p = 5, 10, 20$ and 50 respectively.

problems. A variety of dimension reduction models are implemented for censored survival outcome and regression problems. In addition, we implemented parallel computing for numerically appropriate the gradient function. This is particularly useful for semi-parametric estimating equation methods because the objective function usually involves kernel estimations and the gradients are difficult to calculate. Our package can also be used as a general purpose solver and is comparable with existing manifold optimization approaches. However, since the performances of different optimization approaches could be problem dependent, hence, it could be interesting to investigate other choices such as the "LRBFGS" approach in the **ManifoldOptim** package.

Our package also serves as a platform for future methodology developments along this line of work. For example, we are currently developing a personalized dose-finding model with dimension reduction structure (Zhou and Zhu, 2018). Also, when the number of covariates p is large, the model can be over-parameterized. Hence, applying a L_1 penalty can force sparsity and allow the model to handle high-dimensional data. To this end, first-order optimization approaches can have advantages over second-order approaches. However, persevering the orthogonality during the Cayley transformation while also preserve the sparsity can be a challenging task and requires new methodologies. Furthermore, tuning parameters can be selected through a cross-validation approach, which can be implemented in the future.

Acknowledgement

Xin Zhang's research was supported in part by grants DMS-1613154 and CCF-1617691 from U.S. National Science Foundation. Ruoqing Zhu's research was supported in part by grant RB19046 from UIUC.

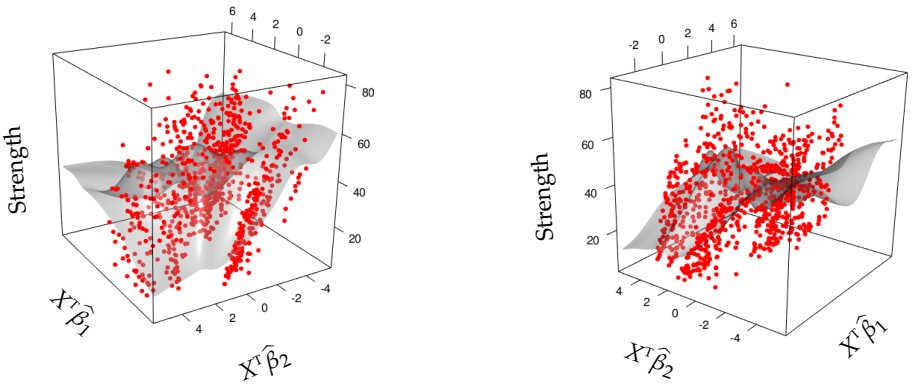


Figure 3: Response variable over learned directions

Bibliography

F. Chiaromonte, R. D. Cook, and B. Li. Sufficient dimensions reduction in regressions with categorical predictors. *The Annals of Statistics*, 30(2):475–497, 2002. URL https://doi.org/10.1214/aos/1021379862. [p24]

R. D. Cook. *Regression Graphics: Ideas for Studying Regressions through Graphics*, volume 482. John Wiley & Sons, 2009. [p24]

R. D. Cook and H. Lee. Dimension reduction in binary response regression. *Journal of the American Statistical Association*, 94(448):1187–1200, 1999. URL https://doi.org/10.1080/01621459.1999.10473873. [p24, 25]

R. D. Cook and S. Weisberg. Discussion of 'sliced inverse regression for dimension reduction'. *Journal of the American Statistical Association*, 86(414):328, 1991. URL https://doi.org/10.2307/2290564. [p24, 25]

R. D. Cook and X. Zhang. Fused estimators of the central subspace in sufficient dimension reduction. *Journal of the American Statistical Association*, 109(506):815–827, 2014. URL https://doi.org/10.1080/01621459.2013.866563. [p24]

R. D. Cook, B. Li, and F. Chiaromonte. Envelope models for parsimonious and efficient multivariate linear regression. *Statistica Sinica*, pages 927–960, 2010. [p24]

D. M. Dabrowska. Uniform consistency of the kernel conditional kaplan-meier estimate. *The Annals of Statistics*, pages 1157–1167, 1989. URL https://doi.org/10.1214/aos/1176347261. [p26]

Y. Dong and B. Li. Dimension reduction for non-elliptically distributed predictors: Second-order methods. *Biometrika*, 97(2):279–294, 2010. URL https://doi.org/10.1093/biomet/asq016. [p24]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL https://doi.org/10.18637/jss.v040.i08. [p25]

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics & Data Analysis*, 71:1054–1063, 2014. URL http://dx.doi.org/10.1016/j.csda.2013.02.005. [p25]

A. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2):303–353, 1998. URL https://doi.org/10.1137/s0895479895290954. [p25]

L. P. Hansen. Large sample properties of generalized method of moments estimators. *Econometrica: Journal of the Econometric Society*, pages 1029–1054, 1982. URL https://doi.org/10.2307/1912775. [p26]

M.-Y. Huang and C.-T. Chiang. An effective semiparametric estimation approach for the sufficient dimension reduction model. *Journal of the American Statistical Association*, pages 1–15, 2017. URL https://doi.org/10.1080/01621459.2016.1215987. [p26]

W. Huang, P.-A. Absil, K. A. Gallivan, and P. Hand. Roptlib: An object-oriented c++ library for optimization on riemannian manifolds. *ACM Transactions on Mathematical Software*, 44(4):1–21, 2018. URL https://doi.org/10.1145/3218822. [p25, 30, 31]

I. T. Jolliffe. Principal component analysis and factor analysis. In *Principal Component Analysis*, pages 115–128. Springer-Verlag, 1986. URL https://doi.org/10.1007/978-1-4757-1904-8_7. [p24]

K.-Y. Lee, B. Li, F. Chiaromonte, and others. A general theory for nonlinear sufficient dimension reduction: Formulation and estimation. *The Annals of Statistics*, 41(1):221–249, 2013. URL https://doi.org/10.1214/12-aos1071. [p24]

B. Li and Y. Dong. Dimension reduction for nonelliptically distributed predictors. *The Annals of Statistics*, pages 1272–1298, 2009. URL https://doi.org/10.1214/08-aos598. [p24]

B. Li and S. Wang. On directional regression for dimension reduction. *Journal of the American Statistical Association*, 102(479):997–1008, 2007. URL https://doi.org/10.1198/016214507000000536. [p24]

K.-C. Li. Sliced inverse regression for dimension reduction. *Journal of the American Statistical Association*, 86(414):316–327, 1991. URL https://doi.org/10.2307/2290563. [p24, 25]

K.-C. Li, J.-L. Wang, C.-H. Chen, and others. Dimension reduction for censored regression data. *The Annals of Statistics*, 27(1):1–23, 1999. URL https://doi.org/10.1214/aos/1018031098. [p24]

L. Li and X. Yin. Sliced inverse regression with regularizations. *Biometrics*, 64(1):124–131, 2008. URL https://doi.org/10.1111/j.1541-0420.2007.00836.x. [p25]

L. Li and X. Zhang. Parsimonious tensor response regression. *Journal of the American Statistical Association*, pages 1–16, 2017. URL https://doi.org/10.1080/01621459.2016.1193022. [p24]

W. Lu and L. Li. Sufficient dimension reduction for censored regressions. *Biometrics*, 67(2):513–523, 2011. URL https://doi.org/10.1111/j.1541-0420.2010.01490.x. [p24]

Y. Ma and X. Zhang. A validated information criterion to determine the structural dimension in dimension reduction models. *Biometrika*, 102(2):409–420, 2015. URL https://doi.org/10.1093/biomet/asv004. [p26]

Y. Ma and L. Zhu. Efficiency loss caused by linearity condition in dimension reduction. *Biometrika*, 99 (1):1–13, 2012a. URL https://doi.org/10.1093/biomet/ass075. [p24]

Y. Ma and L. Zhu. A semiparametric approach to dimension reduction. *Journal of the American Statistical Association*, 107(497):168–179, 2012b. URL https://doi.org/10.1080/01621459.2011.646925. [p24, 25, 27, 28, 29]

Y. Ma and L. Zhu. Efficient estimation in sufficient dimension reduction. *Annals of statistics*, 41(1):250, 2013a. URL https://doi.org/10.1214/12-aos1072. [p24, 25, 26]

Y. Ma and L. Zhu. A review on dimension reduction. *International Statistical Review*, 81(1):134–150, 2013b. URL https://doi.org/10.1111/j.1751-5823.2012.00182.x. [p24]

S. Martin, A. M. Raim, W. Huang, and K. P. Adragni. Manifoldoptim: An r interface to the roptlib library for riemannian manifold optimization, 2016. URL https://arxiv.org/abs/1612.03930. [p25, 30]

B. W. Silverman. Density estimation in action. In *Density Estimation for Statistics and Data Analysis*, pages 120–158. Springer-Verlag, 1986. URL https://doi.org/10.1007/978-1-4899-3324-9_6. [p27]

Q. Sun, R. Zhu, T. Wang, and D. Zeng. Counting process based dimension reduction methods for censored outcomes, 2017. URL https://arxiv.org/abs/1704.05046. [p24, 25, 26, 27]

Z. Wen and W. Yin. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1-2):397–434, 2012. URL https://doi.org/10.1007/s10107-012-0584-1. [p24, 25, 26, 27, 32]

Z. Wen, D. Goldfarb, and W. Yin. Alternating direction augmented lagrangian methods for semidefinite programming. *Mathematical Programming Computation*, 2(3-4):203–230, 2010. URL https://doi.org/10.1007/s12532-010-0017-1. [p25]

Y. Xia. A constructive approach to the estimation of dimension reduction directions. *The Annals of Statistics*, pages 2654–2690, 2007. URL https://doi.org/10.1214/009053607000000352. [p24]

Y. Xia, H. Tong, W. Li, and L.-X. Zhu. An adaptive estimation of dimension reduction space. *Journal of the Royal Statistical Society B*, 64(3):363–410, 2002. [p24]

Y. Xia, D. Zhang, and J. Xu. Dimension reduction and semiparametric estimation of survival models. *Journal of the American Statistical Association*, 105(489):278–290, 2010. URL https://doi.org/10.1198/jasa.2009.tm09372. [p24]

K. Xu, W. Guo, M. Xiong, L. Zhu, and L. Jin. An estimating equation approach to dimension reduction for longitudinal data. *Biometrika*, 103(1):189–203, 2016. URL https://doi.org/10.1093/biomet/asv066. [p26]

I.-C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998. URL https://doi.org/10.1016/s0008-8846(98)00165-3. [p31]

X. Yin and R. D. Cook. Dimension reduction for the conditional kth moment in regression. *Journal of the Royal Statistical Society B*, 64(2):159–175, 2002. URL https://doi.org/10.1111/1467-9868.00330. [p24]

P. Zeng and Y. Zhu. An integral transform method for estimating the central mean and central subspaces. *Journal of Multivariate Analysis*, 101(1):271–290, 2010. URL https://doi.org/10.1016/j.jmva.2009.08.004. [p24]

G. Zhao, Y. Ma, and W. Lu. Efficient estimation for dimension reduction with censored data, 2017. URL https://arxiv.org/abs/1710.05377. [p24]

W. Zhou and R. Zhu. A dimension reduction framework for personalized dose finding, 2018. URL https://arxiv.org/abs/1802.06156. [p33]

L. Zhu, B. Miao, and H. Peng. On sliced inverse regression with high-dimensional covariates. *Journal of the American Statistical Association*, 101(474):630–643, 2006. URL https://doi.org/10.1198/016214505000001285. [p24, 25]

L. Zhu, T. Wang, L. Zhu, and L. Ferré. Sufficient dimension reduction through discretization-expectation estimation. *Biometrika*, 97(2):295–304, 2010a. URL https://doi.org/10.1093/biomet/asq018. [p24]

L.-P. Zhu, L.-X. Zhu, and Z.-H. Feng. Dimension reduction in regressions through cumulative slicing estimation. *Journal of the American Statistical Association*, 105(492):1455–1466, 2010b. URL https://doi.org/10.1198/jasa.2010.tm09666. [p24]

Ruoqing Zhu
Department of Statistics
University of Illinois at Urbana-Champaign
725 S. Wright St., 116 D
Champaign, IL 61820, USA
E-mail: rqzhu@illinois.edu

Jiyang Zhang
Department of Statistics
University of Illinois at Urbana-Champaign
725 S. Wright St.
Champaign, IL 61820, USA
E-mail: jiyangz2@illinois.edu

Ruilin Zhao
School of Engineering and Applied Science
University of Pennsylvania
220 South 33rd St.
Philadelphia, PA 19104, USA

E-mail: rzhao15@seas.upenn.edu

Peng Xu
Departments of Statistics
Columbia University
1255 Amsterdam Avenue
New York, NY 10027, USA
E-mail: px2132@columbia.edu

Wenzhuo Zhou
Department of Statistics
University of Illinois at Urbana-Champaign
725 S. Wright St.
Champaign, IL 61820, USA
E-mail: wenzhuo3@illinois.edu

Xin Zhang
Department of Statistics
Florida State University
214 OSB, 117 N. Woodward Ave.
Tallahassee, FL 32306-4330, USA
E-mail: zhxnzx@gmail.com

coxed: An R Package for Computing Duration-Based Quantities from the Cox Proportional Hazards Model

by Jonathan Kropko and Jeffrey J. Harden

Abstract The Cox proportional hazards model is one of the most frequently used estimators in duration (survival) analysis. Because it is estimated using only the observed durations' rank ordering, typical quantities of interest used to communicate results of the Cox model come from the hazard function (e.g., hazard ratios or percentage changes in the hazard rate). These quantities are substantively vague and difficult for many audiences of research to understand. We introduce a suite of methods in the R package **coxed** to address these problems. The package allows researchers to calculate duration-based quantities from Cox model results, such as the expected duration (or survival time) given covariate values and marginal changes in duration for a specified change in a covariate. These duration-based quantities often match better with researchers' substantive interests and are easily understood by most readers. We describe the methods and illustrate use of the package.

Introduction

The Cox proportional hazards model (Cox, 1972) is frequently used for duration (survival) analysis in a myriad of disciplines including the health sciences, social sciences, operations research, and engineering. For many researchers who employ the Cox model, the chief concept of substantive interest is the duration of an event, such as the survival time of a patient or the duration of a civil war. However, the standard methods of reporting results from the Cox model—which are based in the hazard function—communicate no *specific* information about duration. As a result, standard interpretations of Cox model results are often substantively vague and difficult for many audiences of research to understand.

Here we introduce an R package implementation of *Cox proportional hazards model with expected durations*, or COX ED (Kropko and Harden, 2020). The COX ED suite of methods available in the **coxed** package provides a more intuitive approach to communicating results from the Cox model. Specifically, it computes duration-based quantities of interest, such as the expected time until event occurrence according to the estimated model. These quantities have long been available with parametric duration models, but in some instances researchers may not wish to make the distributional assumptions required of those estimators. The COX ED methods allow researchers to stay within the Cox model framework, but communicate results in the language of time. This affords more conceptual precision when conversing with other researchers and makes the results of the analysis more intuitive and accessible for general audiences.

The methodology

The goal of COX ED is to generate expected durations for individual observations and/or marginal changes in expected duration given a change in a covariate from the Cox model. Specifically, the methods can compute (1) the expected duration for each observation used to fit the Cox model, given the covariates, (2) the expected duration for a "new" observation with a covariate profile set by the analyst, or (3) the first difference, or change, in expected duration given two new observations.

There are two different methods of generating duration-based quantities in the package. The first method employs a generalized additive model (GAM) to map the model's exponentiated linear predictor values to duration times. The second method calculates expected durations by using nonparametric estimates of the baseline hazard and survivor functions. We present overviews of these methods here. See Kropko and Harden (2020) for additional details, including simulation results comparing the two methods. Importantly, both approaches use coefficient estimates from the Cox model, so researchers must first estimate the model just as they always have. COX ED is a postestimation procedure, not a new estimator. All of the choices required of applied researchers in estimating the Cox model must be made first, at the estimation stage, before proceeding to implement COX ED.[1]

[1] Additionally, because it is used after estimation, more extensive modeling features—such as non-linear effects or time-varying covariates—can be incorporated into the use of COX ED.

Method 1: GAM

The GAM approach to COX ED proceeds according to five steps. As is noted above, the first step is model estimation. Then the method computes expected values of risk for each observation by matrix-multiplying the covariates, $\boldsymbol{X}$, by the estimated coefficients from the model, $\hat{\boldsymbol{\beta}}$, then exponentiating the result. This creates $\exp(\boldsymbol{X}\hat{\boldsymbol{\beta}})$, or the exponentiated linear predictor (ELP). Then the observations are ranked from smallest to largest according to their values of the ELP. This ranking is interpreted as the expected order of failure; the larger the value of the ELP, the sooner the model expects that observation to fail, relative to the other observations.

The next step is to connect the model's expected risk for each observation (ELP) to duration time (the observed durations). A GAM fits a model to data by using a series of locally-estimated polynomial splines set by the user (Hastie and Tibshirani, 1990). It is a flexible means of allowing for the possibility of nonlinear relationships between variables. COX ED uses a GAM to model the observed durations as a function of the linear predictor ranks generated in the previous step. More specifically, the method utilizes a cubic regression spline to draw a smoothed line summarizing the bivariate relationship between the observed durations and the ranks (for more details, see Wood, 2006, 2011).[2]

The GAM fit can be used directly to compute expected durations, given the covariates, for each observation in the data. However, for most researchers it is more useful to assess how a change to a particular covariate of interest corresponds to changes in expected duration. In order to examine such marginal changes, it is necessary to create two or more "new" observations corresponding to theoretically-interesting, hypothetical covariate profiles. For example, the analyst might set an indicator variable to 0 and 1 or a continuous variable to a "low" and a "high" value. COX ED allows the covariates in the model to vary naturally over the entire data, then averages over them in the computations.[3] For instance, to estimate the effect of an increase in a covariate X_1 from 0 to 1 on the expected duration, we use the following steps:

(a) Set X_1 to 1 for the entire data (all N observations) and calculate the ELP for every observation, then take an average value of those computations (the median is the default).

(b) Repeat step (a) while setting X_1 equal to 0.

(c) Take the values obtained in steps (a) and (b) and append them to the list of ELP values from the original Cox model in which X_1 is left as exogenous data. Then compute new rankings of the linear predictor values from this list, which is now length $N + 2$.

(d) Pass the list of rankings from step (c) to the GAM as new data to generate expected values. Note that a new GAM is not estimated at this step. Rather, expected durations are generated for each observation—including the two new ones created in steps (a) and (b)—using the previously estimated GAM. This produces point estimates of the expected durations for those two new observations.

(e) Compute the difference between the two estimates obtained in step (d): the expected duration for the data in which X_1 is set to 1 and the expected duration for the data in which X_1 is set to 0. This quantity is a point estimate for the marginal effect, or first difference, corresponding to the change in X_1 from 0 to 1.

Finally, to produce estimates of uncertainty, the GAM approach repeats this process via bootstrapping. The method generates bootstrap samples of the data and re-estimates the Cox model coefficients on each bootstrap sample.[4] At each iteration, this produces a new vector of actual durations and a new ranking of ELP values, which are then used to fit a new GAM. This process results in a distribution of expected durations for each independent variable profile (e.g., step d) and a distribution of the marginal effect (step e). These distributions can be used to produce standard errors and confidence intervals for the estimates.[5] Importantly, by bootstrapping the entire process, this step incorporates the uncertainty from the Cox model estimation *and* the uncertainty from the GAM.

[2]The GAM is fit with the uncensored observations only. If the sample contains a large proportion of censored observations, the NPSF method (see below) may be preferable to the GAM method.

[3]This default option can be changed at the discretion of the analyst.

[4]Standard bootstrapping at the observation level or bootstrapping at the group level (Cameron et al., 2008) are both available.

[5]By default, the method computes the standard errors of each quantity as the standard deviation of its bootstrap distribution. The halfwidth of the confidence interval is then computed by multiplying a tunable critical value based on the standard normal distribution by the standard error. The default critical value is 1.96 (i.e., a 95% confidence interval). Fully non-parametric confidence intervals based on quantiles or bias-corrected quantiles of the bootstrap distribution are also available (see below).

Method 2: Nonparametric step-function

One drawback to the GAM approach is that it uses two statistical models (Cox model and GAM), which yields two sources of estimation uncertainty. An alternative approach comes from the method proposed by Cox and Oakes (1984, 107–109) for estimating the cumulative baseline hazard function. This method is nonparametric and results in a step-function representation of the cumulative baseline hazard; we refer to it as the nonparametric step-function (NPSF) approach.

Cox and Oakes (1984, 108) show that the cumulative baseline hazard function can be estimated after fitting a Cox model by

$$\hat{H}_0(t) = \sum_{\tau_j < t} \frac{d_j}{\sum_{l \in \Re(\tau_j)} \hat{\psi}(l)}, \tag{1}$$

where τ_j represents time points earlier than t, d_j is a count of the total number of failures at τ_j, $\Re(\tau_j)$ is the remaining risk set at τ_j, and $\hat{\psi}(l)$ represents the ELP from the Cox model for observations still in the risk set at τ_j. The NPSF method uses equation (1) to calculate the cumulative baseline hazard at all time points in the range of observed durations with the following steps.

(a) Tied durations are handled by collapsing the dataset by unique duration. The method calculates d_j, the numerator in equation (1), for all time points τ_j by summing the indicator for a non-censored failure within each unique duration ($d_j = 0$ only if all observed durations at τ_j are right-censored). Additionally, it sums the ELPs for all observations with the same duration, because these observations leave the risk set at the same time.

(b) The NPSF approach calculates a running sum, in reverse, for the collapsed ELPs. That is, at the first time point this sum includes the ELP for observations at every time point. At the second time point, this sum includes the ELP for every observation except for those with the earliest observed duration. At the last time point, this sum is equal to the sum of only the ELPs of observations with the latest observed duration. These sums represent the denominator of equation (1).

(c) For each time point, the method divides the number of failures d_j by the sum of ELPs for observations still in the risk set.

(d) Finally, the method calculates the running sum of the ratios we derived in the previous step. This running sum is the non-parametric estimate of the cumulative hazard function.

This procedure yields a stepwise function. Time points with no failures do not contribute to the cumulative hazard, so the function is flat until the next time point with observed failures.

The NPSF approach next obtains expected durations and marginal changes in expected duration by first calculating the baseline survivor function from the cumulative hazard function, using

$$\hat{S}_0(t) = \exp[-\hat{H}_0(t)]. \tag{2}$$

Each observation's survivor function is related to the baseline survivor function by

$$\hat{S}_i(t) = \hat{S}_0(t)^{\hat{\psi}(i)}, \tag{3}$$

where $\hat{\psi}(i)$ is the ELP for observation i. These survivor functions can be used directly to calculate expected durations for each observation. The expected value of a non-negative random variable can be calculated by

$$E(X) = \int_0^\infty \Big(1 - F(t)\Big) dt, \tag{4}$$

where $F(.)$ is the cumulative distribution function for X. In the case of a duration variable t_i, the expected duration is

$$E(t_i) = \int_0^T S_i(t)\, dt, \tag{5}$$

where T is the largest possible duration and $S(t)$ is the individual's survivor function. The NPSF method approximates this integral with a right Riemann-sum by calculating the survivor functions at every discrete time point from the minimum to the maximum observed durations, and multiplying these values by the length of the interval between time points with observed failures:

$$E(t_i) \approx \sum_{t_j \in [0,T]} (t_j - t_{j-1}) S_i(t_j). \tag{6}$$

To calculate a marginal effect, the NPSF approach to COX ED follows the same strategy employed in the GAM approach. It creates two new covariate profiles, setting a variable of interest to two theoretically interesting values. It calculates expected values from each profile, then computes the difference in the two estimates. Finally, the method bootstraps to obtain a standard error and/or confidence intervals for this point estimate.

Implementation in R and empirical example

The methods described above are mostly automated in the package; analysts generally need only a `coxph` model object from the **survival** package (Therneau, 2015) or a `cph` model object from the **rms** package (Harrell, 2018), and, if covariate effects are desired, the name of the variable of interest and the two values of that variable they wish to input.[6] However, the functions also allow for several changes to default settings, such as the formulation of the GAM in the first approach or the computation of confidence intervals.

We illustrate the main features of the package with an empirical example. Martin and Vanberg (2003) examine the determinants of negotiation time among political parties forming coalition governments in Western Europe. The outcome variable in this analysis is the number of days between the beginning and end of the bargaining period. The covariates include the range of government—the ideological distance between the extreme members of the coalition—the number of parties in the coalition, as well as several others. Their main hypotheses predict negative coefficients on the range of government and number of parties variables. They expect that increases in the ideological distance between the parties and the size of the coalition correspond with decreases in the risk of government formation, or longer negotiation times.

The authors demonstrate support for their hypotheses with a sample of data on bargaining in Western European democracies between 1950 and 1995. They estimate a Cox model, then interpret the covariate effects with quantities based in the hazard rate. As an alternative, we employ COX ED with these data. We use the `coxed()` function to predict bargaining duration for every case in the data. Then test the first of their hypotheses by computing estimates of bargaining duration at different values of ideological range of government.

The first step with COX ED is to estimate the model. We estimate the Cox model from Martin and Vanberg (2003) using the `Surv()` and `coxph()` functions from the **survival** package:

```
library(coxed)
data(martinvanberg)

mv.surv <- Surv(martinvanberg$formdur, event = rep(1, nrow(martinvanberg)))
mv.cox <- coxph(mv.surv ~ postel + prevdef + cont + ident + rgovm + pgovno +
                          tpgovno + minority, data = martinvanberg)
```

We report these results in Table 1.

Next we use the GAM version of `coxed()` to examine expected durations and marginal changes in duration.[7] We can calculate standard errors and confidence intervals for any of these quantities with the `bootstrap = TRUE` option. By default the bootstrapping procedure uses 200 iterations (to set this value to a different number, use the `B` argument).[8]

[6] Future versions of the software may accept Cox models estimated from other packages, such as **timereg** (Scheike and Zhang, 2011).

[7] For an example of this analysis using the NPSF method, see the vignette for the **coxed** package.

[8] Here we use 30 iterations simply to ease the computational burden of compiling this example. For more reliable results, set `B` to a higher value. There are different methods for calculating a bootstrapped confidence interval. The default method used by `coxed()` (setting the argument `confidence = "studentized"`) adds and subtracts `qnorm(level - (1 - level)/2)` times the bootstrapped standard error to the point estimate, where `level` is the analyst's chosen threshold for evaluating statistical significance. The alternative approach is to take the `(1 - level)/2` and `level + (1 - level)/2` quantiles of the bootstrapped draws, which can be done by specifying

Table 1: Cox model results from Martin and Vanberg (2003). Entries report coefficients with standard errors in parentheses. These results represent a common approach to presenting Cox model output, but the coefficients themselves are not immediately intuitive.

Range of government	-0.213^{*}
	(0.120)
Number of government parties	1.191^{***}
	(0.124)
Number of government parties	-0.432^{***}
$\times \ln(t)$	(0.035)
Do negotiations commence	-0.577^{***}
immediately after an election?	(0.169)
Did the government take a	-0.100
parliamentary defeat?	(0.230)
Continuation	1.100^{***}
	(0.240)
Identifiability	0.146
	(0.119)
Minority government	-0.428^{**}
	(0.208)
Observations	203
R^2	0.745
Max. Possible R^2	1.000
Log Likelihood	-745.478
Wald Test	218.130^{***} (df = 8)
LR Test	277.239^{***} (df = 8)
Score (Logrank) Test	279.277^{***} (df = 8)

Note: Cell entries report Cox model coefficient estimates with standard errors in parentheses. *p<0.1; **p<0.05; ***p<0.01.

```
ed <- coxed(mv.cox, method = "gam", bootstrap = TRUE, B = 30)
```

Now every predicted duration has a standard error and a 95% confidence interval. The first several cases' predicted durations are estimated as follows:

```
> head(ed$exp.dur)
    exp.dur bootstrap.se         lb        ub
1 48.978295    5.6915889 37.8229859 60.133605
2 42.036276    4.8132767 32.6024267 51.470125
3 55.440293    6.8188818 42.0755303 68.805056
4 15.734577    1.7119205 12.3792749 19.089880
5  1.530695    0.3512462  0.8422652  2.219125
6 64.449942    7.7421823 49.2755433 79.624340
```

The `summary()` function, when applied to `coxed()` output, reports either the mean or median estimated duration along with the bootstrapped standard error and confidence interval for the statistic:

```
> summary(ed, stat = "mean")
```

`confidence = "empirical"`. We recommend a higher number of bootstrap iterations for empirical confidence intervals. Additionally, the nonparametric bias corrected and accelerated (BC_a) method can be computed with `confidence = "bca"`, which implements the bias correction and acceleration procedure in DiCiccio and Efron (1996) using code modified from the **mediation** package (Tingley et al., 2014).

```
   mean bootstrap.se     lb    ub
 28.034        1.998 24.119 31.95
> summary(ed, stat = "median")
 median bootstrap.se     lb     ub
 21.208        2.263 16.773 25.643
```

`coxed()` can be used to provide duration predictions for observations outside of the estimation sample. Suppose that we observe five new cases and place them inside a data frame:

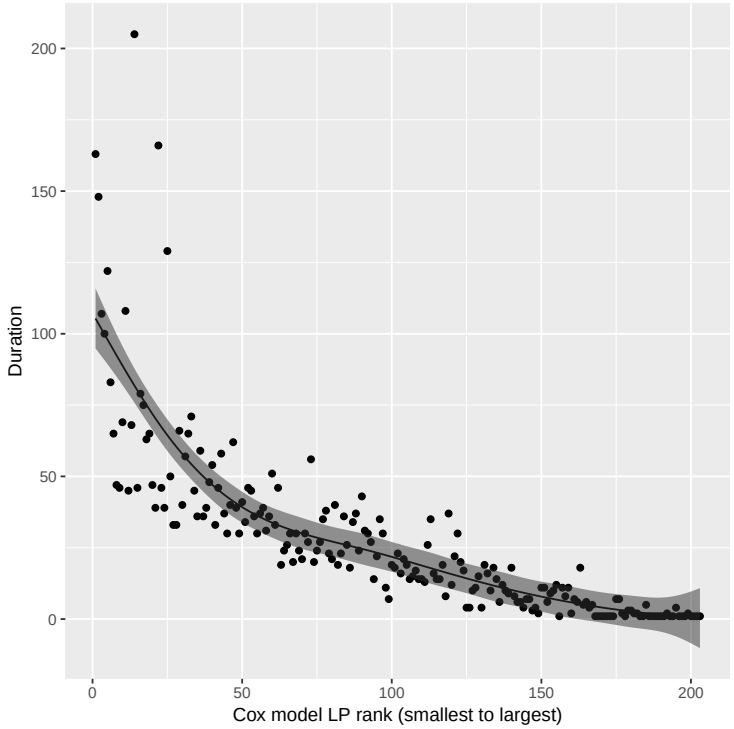
```
new.coalitions <- data.frame(postel = c(1, 1, 1, 0, 1),
                             prevdef = c(0, 0, 1, 1, 0),
                             cont = c(1, 0, 1, 0, 1),
                             ident = c(1, 2, 2, 3, 3),
                             rgovm = c(.3, .8, 1.1, .2, .35),
                             pgovno = c(2, 3, 3, 2, 4),
                             tpgovno = c(3.2, 0, 5, 0, 2.6),
                             minority = c(0, 0, 1, 0, 0))
```

To forecast durations for these cases along with standard errors and confidence intervals, we use the `coxed()` function and place `new.coalitions` into the `newdata` argument:

```
forecast <- coxed(mv.cox, newdata = new.coalitions, method = "gam",
                                                    bootstrap = TRUE, B = 30)
> forecast$exp.dur
    exp.dur bootstrap.se         lb       ub
1 4.5845636    2.7517846 -0.8088352 9.977962
2 0.9542265    0.5656203 -0.1543688 2.062822
3 5.2816962    1.1323172  3.0623953 7.500997
4 1.2358600    0.4684499  0.3177151 2.154005
5 0.5924056    0.7286877 -0.8357961 2.020607
```

The data used by `coxed()` to map rankings to durations are stored in the `gam.data` attribute, and can be used to visualize the fit of the GAM, as in Figure 1.

Figure 1: Mapping duration rankings to observed durations using a GAM. The x-axis plots the ranks of the linear predictor from smallest to largest and the y-axis plots the observed durations. The downward trend shows a non-linear relationship between the model's expectation and the observed data.



We use `coxed()` to provide an answer to the key question, "how much longer will negotiations take for an ideologically polarized coalition as compared to an ideologically homogeneous one?" Specifically, we call `coxed()` and specify two new datasets, one in which `rgovm = 0` indicating that all political parties in the governing coalition have the same ideological position (i.e., a coalition of one party), and one in which `rgovm = 1.24`, indicating that the parties have very different ideological positions.[9] We use `mutate()` from the **dplyr** package (Wickham et al., 2018) to quickly create new data frames in which `rgovm` equals 0 or 1.24 for all cases, and set these two data frames as `newdata` and `newdata2` inside `coxed()`.

```
me <- coxed(mv.cox, method = "gam", bootstrap = TRUE, B = 30,
            newdata = mutate(martinvanberg, rgovm = 0),
            newdata2 = mutate(martinvanberg, rgovm = 1.24))
```

`coxed()` calculates expected durations for all cases under each new data frame and subtracts the durations for each case. To obtain point estimates we can request the mean or median difference.

```
> summary(me, stat = "mean")
             mean bootstrap.se     lb     ub
newdata2   28.927        3.285 22.489 35.365
newdata    25.321        2.632 20.163 30.480
difference  3.605        2.417 -1.133  8.343
> summary(me, stat = "median")
           median bootstrap.se     lb     ub
newdata2   22.392        3.234 16.053 28.730
newdata    19.692        3.449 12.932 26.451
difference  2.928        1.931 -0.857  6.714
```

These results demonstrate that a coalition in which the parties have average ideological differences will take 3.6 more days on average (with a median of 2.9 days) to conclude negotiations than a coalition in which all parties have the same position (i.e., a single-party government).

The NPSF method can be used to compute estimates of these same quantities simply by specifying `method = "npsf"` in the `coxed()` function. Additionally, the package includes a function called `sim.survdata()` designed for simple simulations of duration data that do not assume a distributional form for the baseline hazard. This method, which is fully described in Harden and Kropko (2019), can be useful in several applied and computational settings that involve the Cox model.

Conclusions

The Cox model is popular among applied researchers in a wide range of disciplines due to its inherent flexibility. However, this flexibility makes conveying the substantive meaning of results challenging. By using only the rank ordering of the observed duration times, the Cox model limits researchers to interpreting results in the language of hazard and changes in risk. This yields two key problems. First, it is substantively vague because hazard does not have a meaningful scale. This hinders researchers' capacity to determine whether an estimated effect is substantively "large" or "small." Furthermore, hazard-based interpretations require specialized knowledge to understand. This makes the research less accessible to general audiences, who may be able to learn from the work but cannot due to the means by which results are communicated.

The COX ED methods provide a solution to these problems by allowing researchers to compute duration-based quantities from the Cox model. Communicating results in the language of time allows for more substantive precision and is intuitive to a broad audience of readers. We demonstrate above that COX ED is straightforward to implement in R. The **coxed** package contains functions that allow researchers to use the methods even with minimal knowledge of R. Additionally, the functions are flexible; users can make several changes to many of their features to suit the problem at hand. Finally, the output from the functions provide point estimates, standard errors, and confidence intervals, so researchers can report their results with appropriate measures of uncertainty.

In sum, the **coxed** package provides a useful alternative for researchers to communicate results from the Cox model. It gives them the benefits of the intuitive quantities available in parametric models while retaining the desirable estimation properties of the Cox model. Thus, the analysis can be guided by appropriate modeling choices, but reported in an intuitive, accessible manner.

[9] Martin and Vanberg (2003) select these values in making hazard rate comparisons. The value `rgovm = 1.24` reflects the average ideological range of coalition governments in the sample.

Bibliography

A. C. Cameron, J. B. Gelbach, and D. L. Miller. Bootstrap based improvements for inference with clustered errors. *Review of Economics and Statistics*, 90(3):414–427, 2008. URL https://doi.org/10.1162/rest.90.3.414. [p39]

D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972. URL https://doi.org/10.1111/j.2517-6161.1972.tb00899.x. [p38]

D. R. Cox and D. Oakes. *Analysis of Survival Data*. Monographs on Statistics & Applied Probability. Chapman & Hall/CRC, Boca Raton, FL, 1984. [p40]

T. J. DiCiccio and B. Efron. Bootstrap confidence intervals. *Statistical Science*, 11(3):189–228, 1996. URL https://doi.org/10.1214/ss/1032280214. [p42]

J. J. Harden and J. Kropko. Simulating duration data for the Cox model. *Political Science Research and Methods*, 7(4):921–928, 2019. URL https://doi.org/10.1017/psrm.2018.19. [p44]

F. E. Harrell. *rms: Harrell Miscellaneous*, 2018. R package version 5.1–2. http://biostat.mc.vanderbilt.edu/wiki/Main/Rrms. [p41]

T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman & Hall/CRC, Boca Raton, FL, 1990. [p39]

J. Kropko and J. J. Harden. Beyond the hazard ratio: Generating expected durations from the Cox proportional hazards model. *British Journal of Political Science*, 50(1):303–320, 2020. URL https://doi.org/10.1017/S000712341700045X. [p38]

L. W. Martin and G. Vanberg. Wasting time? The impact of ideology and size on delay in coalition formation. *British Journal of Political Science*, 33(2):323–344, 2003. URL https://doi.org/10.1017/S0007123403000140. [p41, 42, 44]

T. H. Scheike and M.-J. Zhang. Analyzing competing risk data using the R timereg package. *Journal of Statistical Software*, 38(2):1–15, 2011. URL http://dx.doi.org/10.18637/jss.v038.i02. [p41]

T. Therneau. *survival: A Package for Survival Analysis in S*, 2015. R package version 2.38. [p41]

D. Tingley, T. Yamamoto, K. Hirose, L. Keele, and K. Imai. mediation: R package for causal mediation analysis. *Journal of Statistical Software*, 59(5):1–38, 2014. URL http://dx.doi.org/10.18637/jss.v059.i05. [p42]

H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2018. URL https://CRAN.R-project.org/package=dplyr. R package version 0.7.6. [p44]

S. N. Wood. *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC, Boca Raton, FL, 2006. [p39]

S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semi-parametric generalized linear models. *Journal of the Royal Statistical Society. Series B (Methodological)*, 73(1):3–36, 2011. URL https://doi.org/10.1111/j.1467-9868.2010.00749.x. [p39]

Jonathan Kropko
University of Virginia
School of Data Science
Dell 1 Building
Charlottesville, VA 22904
jkropko@virginia.edu

Jeffrey J. Harden
University of Notre Dame
Department of Political Science
2055 Jenkins Nanovic Halls
Notre Dame, IN 46556
jeff.harden@nd.edu

Modeling regimes with extremes: the bayesdfa package for identifying and forecasting common trends and anomalies in multivariate time-series data

by Eric J. Ward, Sean C. Anderson, Luis A. Damiano, Mary E. Hunsicker, Michael A. Litzow

Abstract The **bayesdfa** package provides a flexible Bayesian modeling framework for applying dynamic factor analysis (DFA) to multivariate time-series data as a dimension reduction tool. The core estimation is done with the Stan probabilistic programming language. In addition to being one of the few Bayesian implementations of DFA, novel features of this model include (1) optionally modeling latent process deviations as drawn from a Student-t distribution to better model extremes, and (2) optionally including autoregressive and moving-average components in the latent trends. Besides estimation, we provide a series of plotting functions to visualize trends, loadings, and model predicted values. A secondary analysis for some applications is to identify regimes in latent trends. We provide a flexible Bayesian implementation of a Hidden Markov Model — also written with Stan — to characterize regime shifts in latent processes. We provide simulation testing and details on parameter sensitivities in supplementary information.

Overview

A goal of many multivariate statistical techniques is to reduce dimensionality in observed data to identify shared or latent processes. Factor analysis models represent a general class of models used to relate multiple observations to a lower dimension (factors), while also considering different covariance structures of the observed data. Factors are not directly observed, but represent a hidden, shared process among variables. Though goals of factor analysis are sometimes similar to techniques such as principal component analysis (PCA), factor analysis models explicitly estimate residual error terms, whereas PCA does not (Anderson and Rubin, 1956; Jolliffe, 1986). These factor models are written as $y_i = u_i + \mathbf{Z}\, f_i + \varepsilon_i$, where observed data y_i is a linear combination of an intercept u_i and the product of latent factors f_i and loadings $\mathbf{Z}$ (loadings are sometimes referred to in the literature as $\mathbf{L}$).

In a time-series setting, factor models may be extended to dynamic factor analysis (DFA) models. DFA models aim to reduce the dimensionality of a collection of time series by estimating a set of shared trends and factors, representing the linear effects of each trend on the observed data (Molenaar, 1985; Zuur et al., 2003; Stock and Watson, 2005). The number of trends m is chosen to be less or equal than the number of time series n. The general form of the DFA model can be formulated as a state-space model (Petris, 2010). The latent processes (also referred to as 'trends') are generally modeled as random walks, so that trend i is modeled as $x_{i,t+1} = x_{i,t} + w_{i,t}$ where $x_{i,t}$ is the value of the i-th latent trend at time t, and the deviations $w_{i,t}$ are modeled as white noise. Across trends, these deviations are modeled as $\mathbf{w}_t \sim \text{MVN}(0, \mathbf{Q})$. The latent trends $x_{i,t}$ are linked to data via a loadings matrix $\mathbf{Z}$ whose values do not evolve through time, $y_t = \mathbf{Z}x_t + \mathbf{a} + \mathbf{B}d_t + \mathbf{e}_t$. The loadings matrix $\mathbf{Z}$ is dimensioned $n \times m$ so that $Z_{j,i}$ represents the effect of trend i on time series j. The parameters $\mathbf{a}$ and $\mathbf{B}$ are optional parameters, representing time-series-specific intercepts and effects of covariates, d_t. Finally, the residual errors are assumed to be $\mathbf{e}_t \sim \text{MVN}(0, \mathbf{R})$, where $\mathbf{R}$ is an estimated covariance matrix.

Estimation of DFA models is typically done in a maximum likelihood framework, using the expectation-maximization (EM) algorithm or other optimization tools. Implementation of these methods is available in multiple R packages including **dlm** (Petris, 2010), **KFAS** (Helske, 2017), **MARSS** (Holmes et al., 2012b), and **tsfa** (Gilbert and Meijer, 2005). Challenges in parameter estimation and interpretation for DFA models have been well studied. Without constraints, parameters in the DFA model are not identifiable (Harvey, 1990; Zuur et al., 2003). To ensure identifiability of variance parameters, for example, the covariance matrix $\mathbf{Q}$ is generally fixed as an identity matrix (Harvey, 1990). To avoid confounding the latent trends and loadings matrix $\mathbf{Z}$, elements of $\mathbf{Z}$ must also be constrained. A common choice of constraints is for the elements in the first $m - 1$ rows of $\mathbf{Z}$ to be set to zero if the column index is greater than the row index, $j > i$ (Harvey, 1990), though other constraints have been proposed (Bai and Wang, 2015). For a 3-trend DFA model for instance, these constraints

would mean that the **Z** matrix parameters would be configured as

$$\begin{bmatrix} Z_{1,1} & 0 & 0 \\ Z_{2,1} & Z_{2,2} & 0 \\ Z_{3,1} & Z_{3,2} & Z_{3,3} \\ \dots & \dots & \dots \end{bmatrix}.$$

Several previous approaches to DFA estimation in a maximum likelihood framework also center (subtract the sample means) or standardize (subtract the sample means and divide by the sample standard deviations) data prior to fitting DFA models and set the intercepts a equal to zero to avoid potential confounding of level parameters (Holmes et al., 2012a). We adopt a similar approach, allowing users to either center or standardize data before estimation, and not including the intercepts as estimated parameters.

Label switching

We developed our DFA model in a Bayesian framework, using Stan and the package **rstan** (Stan Development Team, 2016), which implements Markov chain Monte Carlo (MCMC) using the No-U Turn Sampling (NUTS) algorithm (Hoffman and Gelman, 2014; Carpenter et al., 2017). Although estimation of the DFA model in a Bayesian setting is not new (Aguilar and West, 2000; Koop and Korobilis, 2010; Stock and Watson, 2011), it presents several interesting challenges over the EM algorithm. In addition to the constraints on **Q** and **Z**, Bayesian estimation suffers from a problem of label switching. In particular, elements of **F** or **Z** may flip sign within an MCMC chain, or multiple chains may converge on parameters that are identical in magnitude but with different signs.

To minimize issues with label switching, previous work on Bayesian factor analysis has proposed additional constraints on the loadings matrix, including setting the elements of **Z** to be constrained (-1, 1), or adding a positive constraint to the diagonal, $Z_{ii} > 0$ (Aguilar and West, 2000; Geweke and Zhou, 1996). Though these constraints generally help, there may be situations where MCMC chains still do not converge. To address this issue, we adopt the parameter-expanded priors for the loadings and trends proposed by Ghosh and Dunson (2009). To ensure that the sign of the estimated quantities is the same across MCMC chains, we created the function `flip_trends()` to flip the posterior samples of MCMC chains relative to the first chain as needed.

The Bayesian dynamic factor model with extremes

There are several approaches for modeling extreme deviations in time series models. Techniques include modeling deviations as a two-component mixture (Ward et al., 2007; Evin et al., 2011), or modeling deviations with non-Gaussian distributions including the Student-t distribution (Praetz, 1972; Anderson et al., 2017; Anderson and Ward, 2018). There are several existing packages to include Student-t distributions; these include **heavy** for applications to regression and mixed effects models (Osorio and F., 2018), **bsts** for univariate time series models (Scott, 2018), and **stochvol** for stochastic volatility models (Kastner, 2016). Because switching from a Gaussian to Student-t distribution only introduces a single parameter, ν, the degrees of freedom, we extend the latter approach to a multivariate setting to model extreme events in the latent trends, so that deviations in the trends are modeled as $w_t \sim \text{MVT}(\nu, 0, \mathbf{Q})$. As before, **Q** is fixed as an identity matrix **I**. Our parameterization constrains DFA models to have the same degrees of freedom ν in the residuals of the multiple trends, which may be fixed *a priori* or treated as a free parameter with a gamma(shape = 2, rate = 0.1)[2,∞] prior (Juárez and Steel, 2010).

Including autoregressive and moving average components

The trends of the dynamic factor model are most commonly modeled as non-stationary random walks, $x_{i,t+1} = x_{i,t} + w_{i,t}$, where the $w_{i,t} \sim N(0,1)$ are Gaussian white noise. Like with other vector autoregressive time series models, this framework can be easily extended to include optional autoregressive (AR) or moving average (MA) components (Chow et al., 2011). We allow for AR(1) and MA(1) processes to be specified with boolean arguments to the `fit_dfa()` function. For both the AR(1) and MA(1) components, we assume separate parameters for each trend. Including the AR(1) component ϕ_i makes the trend process become $x_{i,t+1} = \phi_i x_{i,t} + w_{i,t}$, where values of ϕ_i close to 1 make the trend behave as a random walk, and small values of ϕ_i close to 0 make the trend behave as white noise. Similarly, we model the MA(1) component as an AR(1) process on the error terms $w_{i,t}$. Instead of being independent at each time step, θ_i controls the degree of autocorrelation among deviations, $w_{i,t} \sim \text{N}\left(\theta_i w_{i,t-1}, 1\right)$. For stationarity and invertability, we constrain $|\phi_i| < 1$ and $|\theta_i| < 1$.

Rotation of trends and loadings

Like factor analysis models, there are many solutions from a DFA model capable of producing the same fit to the data. Following previous authors, we use a varimax rotation of the loadings matrix $\mathbf{Z}$ to transform the posterior loadings and trends (Kaiser, 1958; Harvey, 1990; Holmes et al., 2012a). If $\widehat{\mathbf{Z}}$ is the posterior mean of the loadings matrix from a DFA model of 4 time series and 2 trends for example, the rotation matrix $\mathbf{W}^* = \text{varimax}(\widehat{\mathbf{Z}})$ is dimensioned 2×2. The rotated loadings matrix can then be calculated as $\widehat{\mathbf{Z}}^* = \widehat{\mathbf{Z}}\,\mathbf{W}^*$ and rotated trends calculated as $\widehat{\mathbf{x}}^* = \mathbf{W}^{*-1}\widehat{\mathbf{x}}$, where $\widehat{\mathbf{x}}$ is the posterior mean of the trends.

Identifying data support for the number of trends

Since the number of trends in a DFA model is not a parameter, comparing data support across models is often necessary. Using model selection tools to identify data support is available via Akaike's Information Criterion (AIC) in packages implementing maximum likelihood for estimation of state-space models (Petris, 2010; Holmes et al., 2012b). In addition to comparing the relative support of different number of trends, model selection for Bayesian dynamic factor models may be useful for evaluating the error structure for the residual error covariance matrix $\mathbf{R}$, whether covariates should be included, whether latent trends are better modeled with a distribution allowing for extremes (MVT versus MVN), and whether the latent trends support estimation of AR or MA components. For our Bayesian DFA models, we extend the **loo** package (Vehtari et al., 2016a,b) to generate estimates of LOOIC (Leave-One-Out Information Criterion) for fitted models. To ease the selection process, **bayesdfa** includes the function `find_dfa_trends()` to run multiple models specified by the user. It returns a table of LOOIC values (denoting which of those failed convergence criteria) and the model with the lowest LOOIC value.

Anomalies or black-swan events

As a diagnostic tool, we include the function `find_swans()` to fitted DFA models. We adopt the same approach and terminology for 'black-swan events' as in Anderson et al. (2017), where black-swan events are rare and unexpected extremes. Our `find_swans()` function first-differences the posterior mean estimates of each DFA trend and evaluates the probability of observing a difference that is more extreme than expected under a normal distribution with the same scale parameter. Events beyond a user-defined threshold (e.g. 1 in 100, or 1 in 10,000) are then classified as outliers and plotted.

Simulation tests

To evaluate the ability of the Bayesian DFA model to identify anomalies in latent processes, we created simulated data using our `sim_dfa()` function. We generated simulated multivariate time series ($n = 4$ time series with $T = 20$ time steps each) with $m = 2$ underlying latent trends. Extremes were included as a step-change in the midpoint of the first trend in each simulated dataset. We varied the value of the step from -4 to -8, which represent unlikely events under the assumption that temporal deviations in the latent trends are distributed according to $N(0, 1)$. Because increased observation error may corrupt inference about anomalies in the trends, we considered three levels of observation error ($\sigma = 0.25, 0.75, 1.25$). We generated 200 simulated samples for each permutation of parameters, resulting in a total of 3000 datasets.

We fit the Bayesian DFA model with Student-t errors to each simulated dataset. As expected, the posterior estimates from these simulations illustrate that the ability to estimate low degrees of freedom is related to the magnitude of extremes (Figure 1). Similarly, higher observation error corrupts the ability to estimate extreme events, even when they are large in magnitude (Figure 1).

Using HMMs to classify regimes in latent DFA trends

An alternative approach to DFA for dimension reduction of multivariate time series data are Hidden Markov Models (HMMs). Like DFA models, they model a latent process for a time series (or collection of multivariate time series). Instead of the latent process being modeled continuously (e.g. as a random walk in DFA), HMMs conceive the latent process as a series of discrete-time, discrete-state first-order Markov chains $s_t \in \{1, \ldots, G\}$ with the number of possible states G specified *a priori*. State transition is characterized by the $G \times G$ transition matrix with simplex rows $\mathbf{A} = \left\{a_{ig}\right\}$ where $a_{ig} = p(s_t = g | s_{t-1} = i)$ represents the probability of transitioning from state i to g. Useful quantities

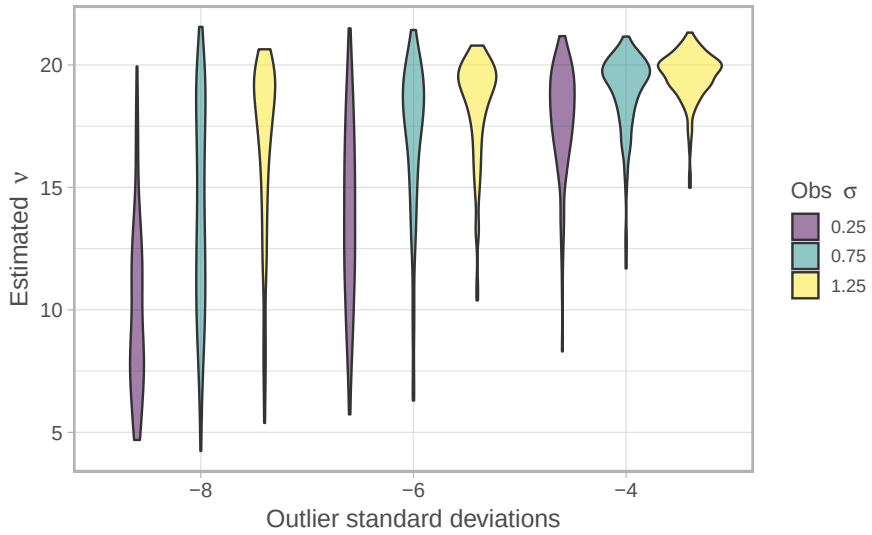


Figure 1: Results for simulated data illustrating support for the Student-t distribution (low values of nu), varying the magnitude of extremes (standard deviations from the mean) and magnitude of observation error.

from HMMs include the transition probabilities between latent states, and the probability of being in a given lantent state at each point in time (Zucchini et al., 2017).

HMMs can be applied to raw multivariate data to identify latent states; however, they may also be linked with DFA to identify regimes and transitions in the latent DFA trends. Similar to DFA, applications of HMMs are widely available in R, including via the packages **depmixS4** (Visser and Speekenbrink, 2010), **HMM** (Himmelmann, 2010), and **msm** (Jackson, 2011). Consistent with our implementation of the Bayesian DFA model, we include fully Bayesian inference in Stan based on Damiano et al. (2018). We apply independent HMM models to each DFA trend to identify alternate states or regimes. Like with the estimation of DFA models, we use the LOOIC metric to evaluate the relative support for HMMs with different numbers of underlying states, selecting the converged model with the lowest LOOIC. By default, we assume the observation model of the input time series to be normally distributed with the scale parameter equal to the estimated residual variance. However, for some applications, such as datasets with changing sampling frequencies over time, uncertainty in DFA trends may also vary through time. To propogate this uncertainty forward, we also allow the residual variance to be entered as a known quantity for every data point in our `find_regimes()` function.

Example application: identifying common patterns in sea surface temperatures in the Northeast Pacific Ocean

To illustrate an example application of the **bayesdfa** package to real data, we use monthly anomalies of sea surface temperature (SST, measured in $C°$). SST is observed from satellite and buoy data at fixed locations, and model-based interpolations are used to generate estimates at additional gridded locations[1]. We used estimates generated at the locations of 4 observing stations used by the Pacific Fisheries Environmental Laboratory[2] from the west coast of North America (USA). The four stations have some degree of correlation with one another, and are separated by approximately 6 degrees of latitude from one another. In summary, we work with $n = 4$ monthly time series with $T = 167$ observations each (from 2003–01 to 2016–05) and no missing values.

Initially, we fit a DFA model with 2 hidden trends, and will assume the 4 time series to have the same error variances **R**. We will fit the DFA model with possible extremes, modeling process error with a Student-t distribution by using the argument `estimate_nu()`. To evaluate whether these data support an extreme DFA with trends modeled as a t-distribution, we will fit two competing forms: one modeling the random walks with a Gaussian distribution, and the other using a Student-t distribution. Generating posterior samples for each model takes approximately 7 minutes per chain, when MCMC chains aren't run in parallel.

[1] https://coastwatch.pfeg.noaa.gov/erddap/info/osuSstAnom/index.html
[2] https://www.pfeg.noaa.gov/

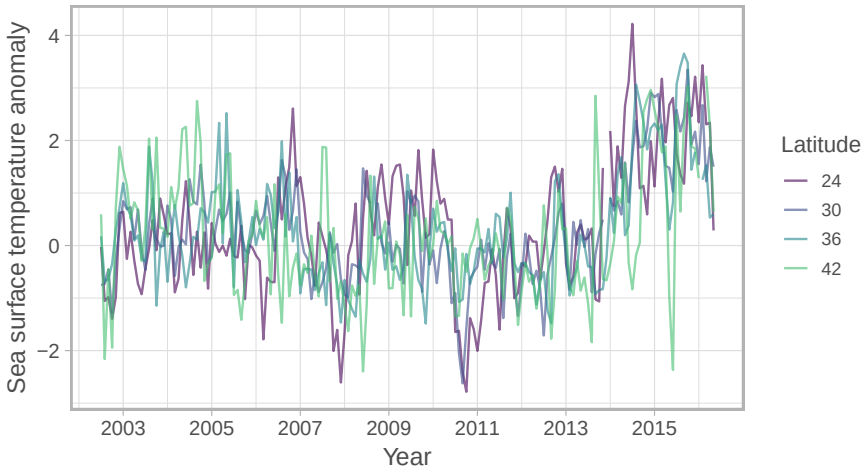


Figure 2: Sea surface temperature anomalies, at four stations on the west coast of the USA ordered by increasing latitude. The station coordinates are (113W, 24N), (119W, 30N), (122W, 36N), (125W, 42N).

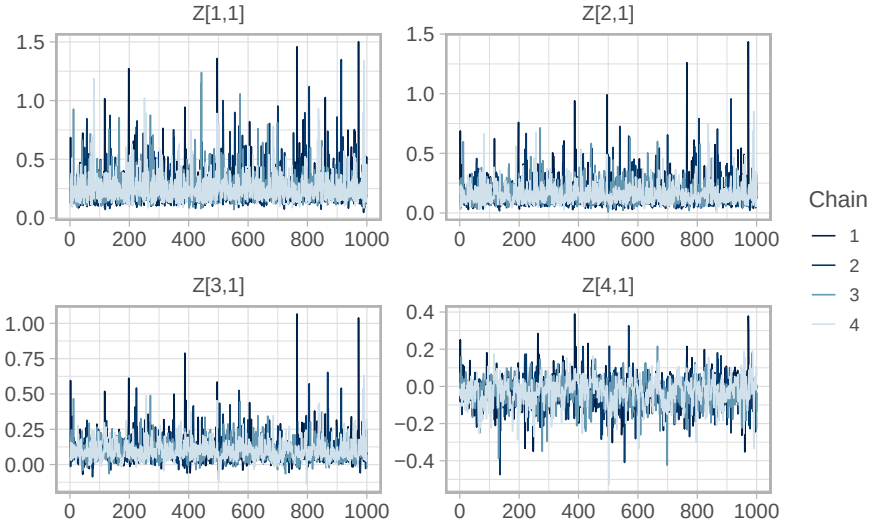


Figure 3: MCMC trace plots of loading parameters (Z) in the DFA model with Student-t errors.

After fitting the models, we confirm whether the MCMC chains are consistent with convergence using a threshold value of $\hat{R} = 1.05$ (Gelman et al., 2014) using our `is_converged()` function. We also visually inspect chain traceplots (e.g. Figure 3) and check the minimum effective sample size across parameters: NaN.

As a consistency diagnostic, we also retrieve the estimated degrees of freedom from the Student-t model ν. By visual inspection, Figure 4 shows that the posterior distribution on ν is lower than the prior distribution.

Visualizing the trends and loadings

We will focus the remaining portion of our analysis on the results from the DFA model with Student-t deviations. In Figure 5, we observe that Trend 1 and Trend 2 both support SST anomalies increasing over the latter half of the time series. Both trends appear to have reversed direction (reverting to the mean in the last 2–3 years) and this pattern is more evident in Trend 1. Because we do not model seasonality explicitly, for example by including a covariate effect for the month, each of the estimated trends also includes the within-year variability that describes seasonal patterns in observed sea surface temperature.

In the violin plot of Figure 6, we note that more southern stations (24 and 30N) contribute largely to Trend 1, while the more northern stations appear to load more heavily on Trend 2.

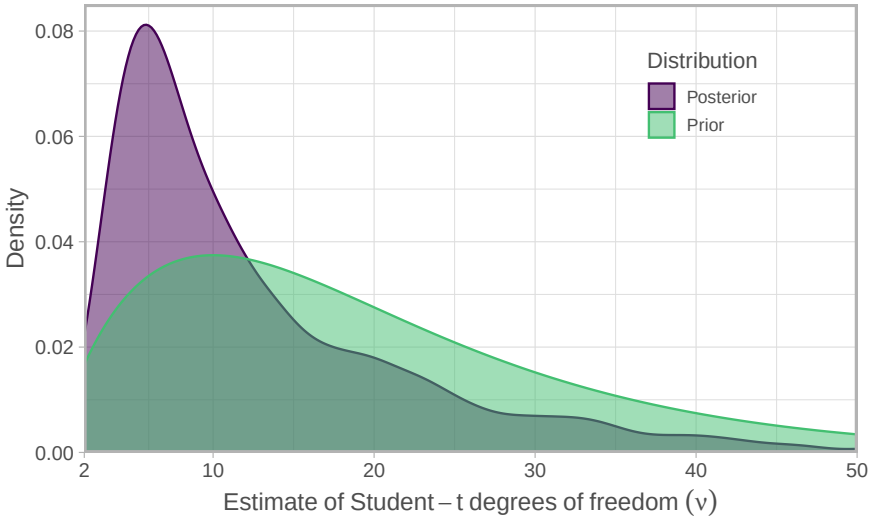


Figure 4: Posterior and prior degrees of freedom in the DFA model with Student-t errors.

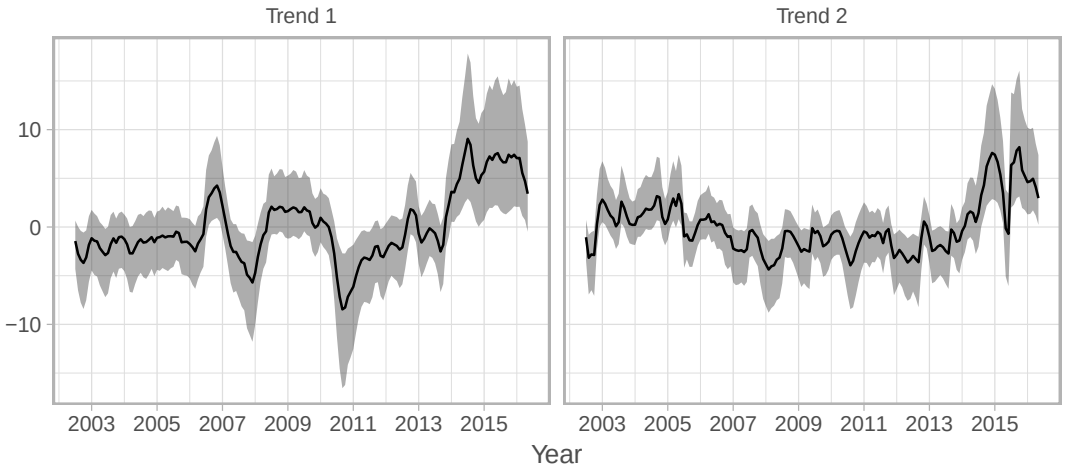


Figure 5: Latent trends from the DFA model with Student-t process deviations. Trends are rotated using the `stats::varimax()` rotation.

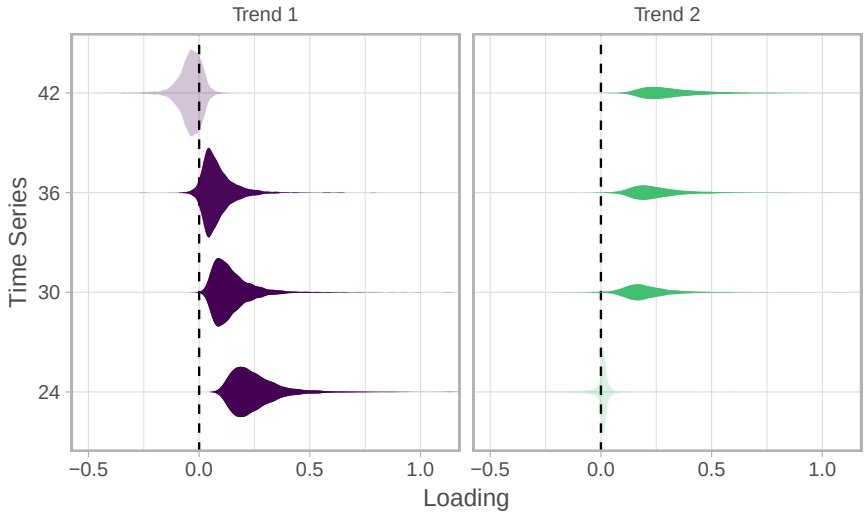


Figure 6: Loadings from the DFA model with Student-t process deviations. Loadings are rotated using the `stats::varimax()` rotation.

Regimes	LOOIC Trend 1	LOOIC Trend 2
1	855.5	756.7
2	31.0	30.3
3	69.1	99.9
4	139.7	164.6

Table 1: LOOIC estimates across different numbers of regimes for each latent DFA trend. LOOIC is calculated using the `loo::loo()` function.

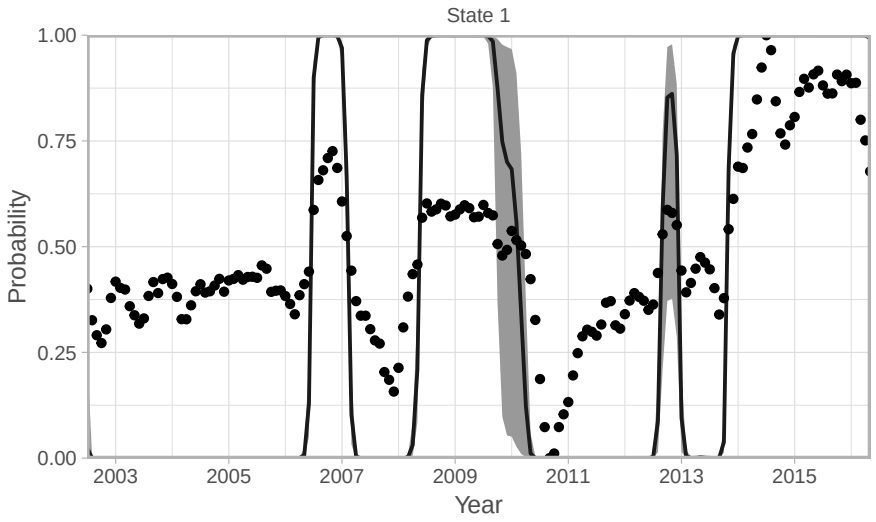


Figure 7: Estimated regimes from the 2-regime HMM in Trend 1 of the DFA model fit to the sea surface temperature anomaly data. The visualization summarizes the assignment probabilities $p(s_t = 1|x_T)$ of Trend 1 being in State 1 (for the sea surface temperature case study, State 1 is associated with warm periods). Dots represent the latent DFA trend scaled to an interval [0, 1]. The black line represents the median and the shaded area uncertainty (90% posterior interval).

Identifying regimes in the latent DFA trends with Hidden Markov Models

For each trend, we apply independent HMMs to examine the support for differing numbers of underlying regimes. Both the posterior mean and standard deviation (optional argument) will be the inputs to the HMM.

Using LOOIC as a metric of support for the number of regimes, the estimates reported in Table 1 support the inclusion of 2 regimes for both Trends 1 and 2.

Our `fit_regimes()` function computes the probability of each time point being in one of the regime states, which may also be visualized using `plot_regime_model()`. For example, the output of the 2-regime model for Trend 1 in Figure 7 suggests a change in the middle of the time series, then changing back again to State 1. Similarly, by the end of the series, the HMM assigns Trend 1 to being in State 1.

Extensions

There are a number of extensions to our implementation of the Bayesian DFA model with extremes that could make the model more applicable to a wider range of problems. Examples for the process model include adopting a skew-t distribution for asymmetric extremes. For models estimating multiple trends, multiple parameters may be treated hierarchically (e.g. covariate effects, variance parameters). For the observation or data model, our implementation of the Bayesian DFA model only includes data arising from a Gaussian or Student-t distribution, though this could be extended to include discrete or other continuous densities. Finally, spatial dynamic factor models (sDFA) have emerged as a useful tool for complicated multivariate spatial datasets (Lopes et al., 2011; Thorson et al., 2015), and could be similarly implemented in Stan.

Conclusion

This paper presents the **bayesdfa** package for applying Bayesian DFA to multivariate time series as a dimension reduction tool, particularly if extreme events may be present in observed data. In addition to allowing for the inclusion of covariates, we also extend the conventional dynamic factor model to include optionial moving average and autoregressive components in the latent trends. Applying this package to a dataset of sea surface temperature from the Northeast Pacific Ocean, we fit DFA models with Gaussian and Student-t errors. Though the model with Student-t errors has slightly lower LOOIC, the results from the two models are similar. Output from these 2-trend DFA models of sea surface temperature are useful in demonstrating a north-to-south gradient in temperature anomalies (Figure 6). Standardized temperature data from southern stations experience more interannual variability and temperatures that are greater in magnitude compared to northern stations (Figure 5). We also illustrate how latent trends from DFA models can be analyzed in a HMM framework to identify regimes and transitions; applied to the sea surface temperature data, both Trend 1 and Trend 2 support 2-regime models (roughly interpreted as 'warm' and 'cool' regimes; Figure 7).

Acknowledgements

This work was funded by NOAA's Fisheries and the Environment (FATE) Program. Development of this package benefitted from discussions with other members of our working group (Jin Gao, Chris Harvey, Sam McClatchie, Stepahni Zador) and scientists at the Northwest Fisheries Science Center (including Mark Scheuerell, James Thorson, Eli Holmes, and Kelly Andrews). 2 anonymous reviewers helped improve the clarity and plots of this paper.

Bibliography

O. Aguilar and M. West. Bayesian Dynamic Factor Models and Portfolio Allocation. *Journal of Business & Economic Statistics*, 18(3):338–357, July 2000. doi: 10.1080/07350015.2000.10524875. [p47]

S. C. Anderson and E. J. Ward. Black swans in space: Modelling spatiotemporal processes with extremes. *Ecology*, In press, 2018. doi: 10.1002/ecy.2403. [p47]

S. C. Anderson, T. A. Branch, A. B. Cooper, and N. K. Dulvy. Black-swan events in animal populations. *Proceedings of the National Academy of Sciences*, 114(12):3252–3257, 2017. doi: 10.1073/pnas.1611525114. [p47, 48]

T. W. Anderson and H. Rubin. Statistical inference in factor analysis. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 5: Contributions to Econometrics, Industrial Research, and Psychometry*, pages 111–150, Berkeley, California, 1956. University of California Press. [p46]

J. Bai and P. Wang. Identification and Bayesian Estimation of Dynamic Factor Models. *Journal of Business & Economic Statistics*, 33(2):221–240, Apr. 2015. doi: 10.1080/07350015.2014.941467. [p46]

B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software, Articles*, 76(1):1–32, 2017. ISSN 1548-7660. doi: 10.18637/jss.v076.i01. URL https://www.jstatsoft.org/v076/i01. [p47]

S.-M. Chow, N. Tang, Y. Yuan, X. Song, and H. Zhu. Bayesian estimation of semiparametric nonlinear dynamic factor analysis models using the Dirichlet process prior. *The British journal of mathematical and statistical psychology*, 64(Pt 1):69–106, Feb. 2011. doi: 10.1348/000711010X497262. [p47]

L. Damiano, B. Peterson, and M. Weylandt. A tutorial on hidden Markov models using Stan. 2018. doi: 10.5281/zenodo.1284341. URL https://doi.org/10.5281/zenodo.1284341. [p49]

G. Evin, J. Merleau, and L. Perreault. Two-component mixtures of normal, gamma, and Gumbel distributions for hydrological applications. *Water Resources Research*, 47(8), Aug. 2011. doi: 10.1029/2010WR010266. [p47]

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. Chapman & Hall, Boca Raton, FL, third edition, 2014. [p50]

J. Geweke and G. Zhou. Measuring the pricing error of the arbitrage pricing theory. *The Review of Financial Studies*, 9(2):557–587, 1996. doi: 10.1093/rfs/9.2.557. URL http://dx.doi.org/10.1093/rfs/9.2.557. [p47]

J. Ghosh and D. B. Dunson. Default prior distributions and efficient posterior computation in Bayesian factor analysis. *Journal of Computational and Graphical Statistics*, 18(2):306–320, 2009. doi: 10.1198/jcgs.2009.07145. URL https://doi.org/10.1198/jcgs.2009.07145. PMID: 23997568. [p47]

P. D. Gilbert and E. Meijer. Time Series Factor Analaysis with an Application to Measuring Money. Technical Report 05F10, University of Groningen, SOM Research School, 2005. [p46]

A. C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, 1990. ISBN 978-0-521-40573-7. [p46, 48]

J. Helske. KFAS: Exponential Family State Space Models in R. *Journal of Statistical Software*, 78(10):1–39, 2017. doi: 10.18637/jss.v078.i10. [p46]

L. Himmelmann. *HMM: HMM - Hidden Markov Models*. 2010. R package version 1.0. [p49]

M. D. Hoffman and A. Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014. [p47]

E. E. Holmes, E. J. Ward, and M. D. Scheuerell. Analysis of multivariate time-series using the MARSS package. Technical report, NOAA Fisheries, Northwest Fisheries Science Center, 2725 Montlake Blvd E., Seattle, WA 98112., 2012a. [p47, 48]

E. E. Holmes, E. J. Ward, and K. Wills. MARSS: Multivariate autoregressive state-space models for analyzing time-series data. *R Journal*, 4(1):11–19, 2012b. [p46, 48]

C. H. Jackson. Multi-State Models for Panel Data: The msm Package for R. *Journal of Statistical Software*, 38(8):1–29, 2011. [p49]

I. T. Jolliffe. Principal Component Analysis and Factor Analysis. In *Principal Component Analysis*, Springer Series in Statistics, pages 115–128. Springer, New York, NY, 1986. ISBN 978-1-4757-1906-2 978-1-4757-1904-8. doi: 10.1007/978-1-4757-1904-8_7. [p46]

M. A. Juárez and M. F. J. Steel. Model-based clustering of non-Gaussian panel data based on skew-t distributions. *J. Bus. Econ. Stat.*, 28(1):52–66, 2010. doi: 10.1198/jbes.2009.07145. [p47]

H. F. Kaiser. The varimax criterion for analytic rotation in factor analysis. *Psychometrika*, 23(3):187–200, Sept. 1958. doi: 10.1007/BF02289233. [p48]

G. Kastner. Dealing with stochastic volatility in time series using the R package stochvol. *Journal of Statistical Software*, 69(5):1–30, 2016. doi: 10.18637/jss.v069.i05. [p47]

G. Koop and D. Korobilis. Bayesian Multivariate Time Series Methods for Empirical Macroeconomics. *Foundations and Trends® in Econometrics*, 3(4):267–358, July 2010. doi: 10.1561/0800000013. [p47]

H. F. Lopes, D. Gamerman, and E. Salazar. Generalized spatial dynamic factor models. *Computational Statistics & Data Analysis*, 55(3):1319–1330, Mar. 2011. doi: 10.1016/j.csda.2010.09.020. [p52]

P. C. M. Molenaar. A dynamic factor model for the analysis of multivariate time series. *Psychometrika*, 50(2):181–202, June 1985. doi: 10.1007/BF02294246. [p46]

Osorio and F. *heavy: Robust estimation using heavy-tailed distributions*, 2018. URL https://CRAN.R-project.org/package=heavy. R package version 0.38.19. [p47]

G. Petris. An R Package for Dynamic Linear Models. *Journal of Statistical Software*, 36(12):1–16, 2010. [p46, 48]

P. D. Praetz. The Distribution of Share Price Changes. *The Journal of Business*, 45(1):49–55, 1972. [p47]

S. L. Scott. bsts: Bayesian structural time series. 2018. URL https://CRAN.R-project.org/package=bsts. R package version 0.8.0. [p47]

Stan Development Team. RStan: The R interface to Stan. 2016. R package version 2.14.1. [p47]

J. H. Stock and M. W. Watson. Implications of Dynamic Factor Models for VAR Analysis. Working Paper 11467, National Bureau of Economic Research, July 2005. [p46]

J. H. Stock and M. W. Watson. Dynamic Factor Models. *The Oxford Handbook of Economic Forecasting*, July 2011. doi: 10.1093/oxfordhb/9780195398649.013.0003. [p47]

J. T. Thorson, M. D. Scheuerell, A. O. Shelton, K. E. See, H. J. Skaug, and K. Kristensen. Spatial factor analysis: A new tool for estimating joint species distributions and correlations in species range. *Methods in Ecology and Evolution*, 6(6):627–637, June 2015. doi: 10.1111/2041-210X.12359. [p52]

A. Vehtari, A. Gelman, and J. Gabry. Loo: Efficient leave-one-out cross-validation and WAIC for Bayesian models. 2016a. R package version 1.1.0. [p48]

A. Vehtari, A. Gelman, and J. Gabry. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 2016b. doi: 10.1007/s11222-016-9696-4. [p48]

I. Visser and M. Speekenbrink. depmixS4: An R Package for Hidden Markov Models. *Journal of Statistical Software*, 36(7):1–21, 2010. [p49]

E. J. Ward, R. Hilborn, R. G. Towell, and L. Gerber. A state–space mixture approach for estimating catastrophic events in time series data. *Canadian Journal of Fisheries and Aquatic Sciences*, 64(6): 899–910, June 2007. doi: 10.1139/f07-060. [p47]

W. Zucchini, I. L. MacDonald, and R. Langrock. *Hidden Markov Models for Time Series: An Introduction Using R, Second Edition*. CRC Press, Dec. 2017. ISBN 978-1-4822-5384-9. [p49]

A. F. Zuur, R. J. Fryer, I. T. Jolliffe, R. Dekker, and J. J. Beukema. Estimating common trends in multivariate time series using dynamic factor analysis. *Environmetrics*, 14(7):665–685, Nov. 2003. doi: 10.1002/env.611. [p46]

Eric J. Ward
Conservation Biology Division, Northwest Fisheries Science Center, National Marine Fisheries Service, National Oceanic and Atmospheric Administration
2725 Montlake Blvd E, Seattle WA, 98112, USA
eric.ward@noaa.gov

Sean C. Anderson
Pacific Biological Station, Fisheries and Oceans Canada
3190 Hammond Bay Rd, Nanaimo, BC, V6T 6N7, Canada
sean.anderson@dfo-mpo.gc.ca

Luis A. Damiano
Iowa State University
2438 Osborn Drive, Snedecor Hall, Ames IA, 50011, USA
ldamiano@iastate.edu

Mary E. Hunsicker
Fish Ecology Division, Northwest Fisheries Science Center, National Marine Fisheries Service, National Oceanic and Atmospheric Administration
2725 Montlake Blvd E, Seattle WA, 98112, USA
mary.hunsicker@noaa.gov

Michael A. Litzow
University Alaska Fairbanks, College of Fisheries and Ocean Sciences,
118 Trident Way, Kodiak Seafood and Marine Science Center, Kodiak, AK 99615, USA
mlitzow@alaska.edu

Fitting Tails by the Empirical Residual Coefficient of Variation: The ercv Package

by Joan del Castillo, Isabel Serra, Maria Padilla and David Moriña

Abstract This article is a self-contained introduction to the R package **ercv** and to the methodology on which it is based through the analysis of nine examples. The methodology is simple and trustworthy for the analysis of extreme values and relates the two main existing methodologies. The package contains R functions for visualizing, fitting and validating the distribution of tails. It also provides multiple threshold tests for a generalized Pareto distribution, together with an automatic threshold selection algorithm.

Introduction and overview

Extreme value theory (EVT) is one of the most important statistical techniques for the applied sciences. A review of the available software on extreme value analysis appears in Gilleland et al. (2013). R software (R Core Team, 2017) contains some useful packages for dealing with EVT. The R package **evir** (Pfaff and McNeil, 2012) provides maximum likelihood estimation (MLE) at the same time for the block maxima and threshold model approaches. The R package **ismev** (Heffernan and Stephenson, 2018) allows fitting parameters of a generalized Pareto distribution depending on covariates and offers diagnostics such as qqplots and return level plots with confidence bands. The R package **poweRlaw** (Gillespie, 2015) enables power laws and other heavy tailed distributions to be fitted using the techniques proposed by Clauset et al. (2009).This approach had been used to describe sizes of cities and word frequency and is linked to the physics of phase transitions and to complex systems.

This paper shows that the R package **ercv** (del Castillo et al., 2017a), based on the coefficient of variation (CV), is a complement, and often an alternative, to the available software on EVT. The mathematical background is shown in Section Mathematical Background, including threshold models and the relationship between power law distribution and the generalized Pareto distributions (GPD), which is the relationship between the two different approaches followed by the aforementioned R packages **evir**, or **ismev**, and **poweRlaw**.

Section Exploratory data analysis with cvplot function introduces the tools for the empirical residual coefficient of variation developed in the papers del Castillo et al. (2014), del Castillo and Serra (2015) and del Castillo and Padilla (2016). Section Examples also shows the exploratory data analysis of nine examples, some of them from the R packages **evir** and **poweRlaw**, with the `cvplot` function, see Figure 1.

Section Estimation and Model diagnostics with Tm function explains the `Tm` function in the R package **ercv** that provides a multiple thresholds test that truly reduces the multiple testing problem in threshold selection and provides clearly defined p-values. The function includes an estimation method of the extreme value index. An automatic threshold selection algorithm provided by the `thrselect` function is explained in Section 12.5 to determine the point above which GPD can be assumed for the tail distribution.

Section Transformation from heavy to light tails (tdata) shows how the methodology developed in the previous sections can be extended with the `tdata` function to all GPD distributions, even with no finite moments. This technique is applied to the MobyDick example and to the Danish fire insurance dataset, a highly heavy-tailed, infinite-variance model. Finally, Section Fitting PoT parameters and tail plots (fitpot ccdfplot) describes the functions of the R package **ercv** that allow estimation of the parameters (`fitpot`) and drawing of the adjustments (`ccdfplot`) for the peak-over-threshold method.

Mathematical Background

Extreme value theory is widely used to model exceedances in many disciplines, such as hydrology, insurance, finance, internet traffic data and environmental science. The underlying mathematical basis is now thoroughly established in Leadbetter et al. (1983), Embrechts et al. (1997), de Haan and Ferreira (2007), Novak (2012) and Resnick (2013). Statistical tools and methods for use with a single time series of data, or with a few series, are well developed in Coles (2001), Beirlant et al. (2006) and Markovich (2007).

Threshold models

The first fundamental theorem on EVT by Fisher and Tippett (1928) and Gnedenko (1943) characterizes the asymptotic distribution of the maximum in observed data. Classical analyses now use the generalized extreme value family of distribution functions for fitting to block maximum data provided the number of blocks is sufficiently large. Another point of view emerged in the 1970's with the fundamental theorem by Pickands (1975) and Balkema and de Haan (1974). The Pickands-Balkema-DeHaan (PBdH) theorem, see McNeil et al. (2005, chap 7), initiated a new way of studying extreme value theory via distributions above a threshold, which use more information than the maximum data grouped into blocks.

Let X be a continuous non-negative r.v. with distribution function $F(x)$. For any threshold, $t > 0$, the r.v. of the conditional distribution of threshold excesses $X - t$ given $X > t$, denoted as $X_t = \{X - t \mid X > t\}$, is called the *residual distribution* of X over t. The cumulative distribution function of X_t, $F_t(x)$, is given by

$$1 - F_t(x) = (1 - F(x + t))/(1 - F(t)). \tag{1}$$

The quantity $M(t) = E(X_t)$ is called the *residual mean* and $V(t) = var(X_t)$ the *residual variance*. The plot of sample mean excesses over increasing thresholds is a commonly used diagnostic tool in risk analysis called ME-plot (`meplot` function in **evir** R package).

The *residual coefficient of variation* is given by

$$\mathrm{CV}(t) \equiv \mathrm{CV}(X_t) = \sqrt{V(t)}/M(t), \tag{2}$$

like the usual CV, the function $\mathrm{CV}(t)$ is independent under change of scale.

The PBdH theorem characterizes the asymptotic distributions of the residual distribution over a high threshold under widely applicable regularity conditions, see Coles (2001). The result essentially says that GPD is the canonical distribution for modelling excess over high thresholds. The probability density function for a $\mathrm{GPD}(\xi, \psi)$ is given by

$$g(x; \xi, \psi) = \begin{cases} \psi^{-1}(1 + \xi x/\psi)^{-(1+\xi)/\xi}, & \xi \neq 0, \\ \psi^{-1} \exp(-x/\psi), & \xi = 0, \end{cases} \tag{3}$$

where $\xi \in \mathbb{R}$ is called the *extreme value index* (evi) and $\psi > 0$ is a scale parameter, $0 \leq x \leq -\psi/\xi$ if $\xi < 0$, and $x \geq 0$ if $\xi \geq 0$. The value of ξ determines the tail type. If $\xi < 0$, we say that the distribution is *light tailed*, if $\xi = 0$ we say it is *exponential tailed*. If $\xi > 0$ a GPD has finite moments of order n if $\xi < 1/n$ and it is called *heavy tailed*. The mean of a GPD is $\psi/(1 - \xi)$ and the variance is $\psi^2/[(1 - \xi)^2(1 - 2\xi)]$ provided $\xi < 1$ and $\xi < 1/2$, respectively. Then, the coefficient of variation is

$$c_\xi = \sqrt{1/(1 - 2\xi)}, \tag{4}$$

the `cvevi` and `evicv` functions of the R package **ercv** correspond to this function and its inverse.

The residual distribution of a GPD is again GPD with the same extreme value index ξ, for any threshold $t > 0$, in fact

$$\mathrm{GPD}_t(\xi, \psi) = \mathrm{GPD}(\xi, \psi + \xi t). \tag{5}$$

Therefore, the residual CV for GPD is independent of the threshold and the scale parameter and is given by equation (4).

The probability density functions (3) are monotone decreasing (L-shaped) for $\xi > -1$, covering practically all the applications. Therefore, we are mainly concerned with the subset of data that indicate this behaviour. For example, if the dataset is concentrated in the centre and decreases on either side (bell-shaped) we will study the upper and lower part (changed sign) of the distribution separately, taking the median or some other location statistic as the origin.

The power law distribution and GPD

The power law distribution is the model, introduced by Pareto,

$$p(x; \alpha, \sigma) = \frac{\alpha}{\sigma} \left(\frac{\sigma}{x}\right)^{\alpha+1}, \quad x > \sigma \tag{6}$$

where $\alpha > 0$ is the tail index and $\sigma > 0$ the minimum value parameter. The model corresponds to the distribution functions F with the linear relation

$$log\left[1 - F\left(x\right)\right] = -\alpha\, log(x) + \alpha\, log\left(\sigma\right), \tag{7}$$

see also Gillespie (2015).

Note that if X is a r.v. with probability density function $p\left(x;\alpha,\sigma\right)$, given by (6), $Z = X - \sigma$ has probability density function

$$g(z;1/\alpha,\sigma/\alpha) = \frac{\alpha}{\sigma}\left(\frac{\sigma}{z+\sigma}\right)^{\alpha+1}, \quad z > 0, \tag{8}$$

that is, there is a one to one correspondence between power law distributions and GPD distributions with heavy tails ($\xi > 0$), where $\xi = 1/\alpha$ and $\sigma = \psi/\xi$. However, the two statistical models (3) and (6), with $\xi > 0$, are different since there is no unique transformation for all functions of the model (the transformation $Z = X - \sigma$ depends on the minimum value parameter σ of the same variable X).

The MLE for model (6) leads to the Hill estimator and Hill-plot (`hill` function in **evir** R package). The support of the distributions in (6) depends on the minimum value parameter σ. Hence, the MLE has no standard regularity conditions and the minimum value parameter σ is estimated with alternative methods, see Clauset et al. (2009) and its implementation in the **poweRlaw** R package by Gillespie (2015).

However, the support of the distributions in (3), with $\xi > 0$, does not depend on parameters and MLE existing for large samples provided $\xi > -1$ and is asymptotically efficient provided $\xi > -0.5$, see del Castillo and Serra (2015) and the references therein for details. The `gdp` function in the **evir** R package provides the MLE for (3).

Note that model (3) includes all the limit distributions (heavy or not) of the residual distribution over a high threshold and comes from a mathematical result (the PBdH theorem) and often (6) comes from empirical evidence of the linear relationship (7) and comparison with other models. Moreover, the linear relationship (7) is also obtained from the relationship between the parameters (8), see the `ccdfplot` function in Section Fitting PoT parameters and tail plots (fitpot ccdfplot).

The residual CV approach

Gupta and Kirmani (2000) show that the residual CV characterizes the distribution in univariate and bivariate cases, provided threre is a finite second moment ($\xi < 1/2$). In the case of GPD, the residual CV is constant and is a one to one transformation of the extreme value index suggesting its use to estimate this index. The residual CV can also be expressed in terms of probabilities, rather than the threshold, through the inverse of the distribution function or the *quantile function* defined by $Q\left(p\right) = inf\left[x : F(x) \geq p\right]$, then the CV can be drawn, for $0 \leq p < 1$, for the threshold $t = Q\left(p\right)$, that is to plot the function $p \to CV(Q(p))$. This representation makes it possible to draw on the same scale for the x axis the residual CV of distributions with different supports.

Exploratory data analysis with `cvplot` function

In this section the `cvplot` function of the R package **ercv** is introduced as a graphical tool for use in a exploratory data analysis, through the nine examples described in Section 3.2. The `cvplot` function is essentially the empirical residual CV whose asymptotic distribution as a stochastic process is explained by del Castillo et al. (2014) and del Castillo and Padilla (2016).

The empirical residual CV and confidence intervals.

Assume that the raw data consist of a sequence of independent and identically distributed measurements $x_1, ..., x_n$. Extreme events are identified by defining a high threshold t for which the *exceedances* are $\{x_j : x_j > t\}$. Hence, we first identify a threshold t such that its exceedances correspond to a constant residual CV (equivalently a GPD). We denote the ordered sample $x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n)}$. The `cvplot` function provides the function $cv(t)$ of the sample coefficient of variation of the *threshold excesses* $(x_j - t)$ given by

$$t \to cv(t) = sd\{x_j - t \mid x_j > t\}/\text{mean}\{x_j - t \mid x_j > t\}, \tag{9}$$

in practice $t = x_{(k)}$ are the order statistics, where, k $(1 \leq k \leq n)$ is the size of the sub-sample excluded. Hereinafter the graph of this function is called CV-plot. Figure 1 shows the CV-plots of nine examples

(blue lines) that we comment on the next section.

Point-wise error limits for $cv(t)$ under GPD(ξ, ψ) (provided $\xi < 1/4$) follow from the asymptotic distribution of the empirical residual CV, by del Castillo and Padilla (2016), in particular for a fixed threshold t, the asymptotic confidence intervals in Figure 1 (solid orange lines) are obtained by

$$\sqrt{n(t)}(cv(t) - c_\xi) \xrightarrow{d} N(0, \sigma_\xi^2), \tag{10}$$

where c_ξ is in (4), $n(t) = \sum_{j=1}^{n} 1_{(x_j > t)}$. For an exponential distribution $(\xi = 0)$, $c_0 = 1$ and $\sigma_0^2 = 1$, and for a uniform distribution $(\xi = -1)$, $c_{-1} = 1/\sqrt{3}$ and $\sigma_{-1}^2 = 8/45$.

By default, if $\sqrt{2}$ is in the range of y's then the `cvplot` function draws the line $y = \sqrt{2}$ (black dotted line), which corresponds to $\xi = 1/4$ (finite fourth moment). Hence, CV-plot larger than this value for high thresholds lead to very heavy tailed distribution and we suggest to switch to transformed data through function `tdata` (Section 12.6). Alternatively, finite moments can be checked by a confidence interval for the MLE estimator of evi, or the methods in the R package **RobExtremes** (Ruckdeschel et al., 2019) and the references cited therein can be used.

The CV-plot is an alternative tool to Hill-plot an to ME-plot. It has two advantages over ME-plot: first, it depends on a scale parameter and CV-plot does not; second, linear functions are defined by two parameters and the constants by only one. So the uncertainty is reduced from three to one single parameter. On the other hand, the Hill-plot can only be used for heavy tailed distributions.

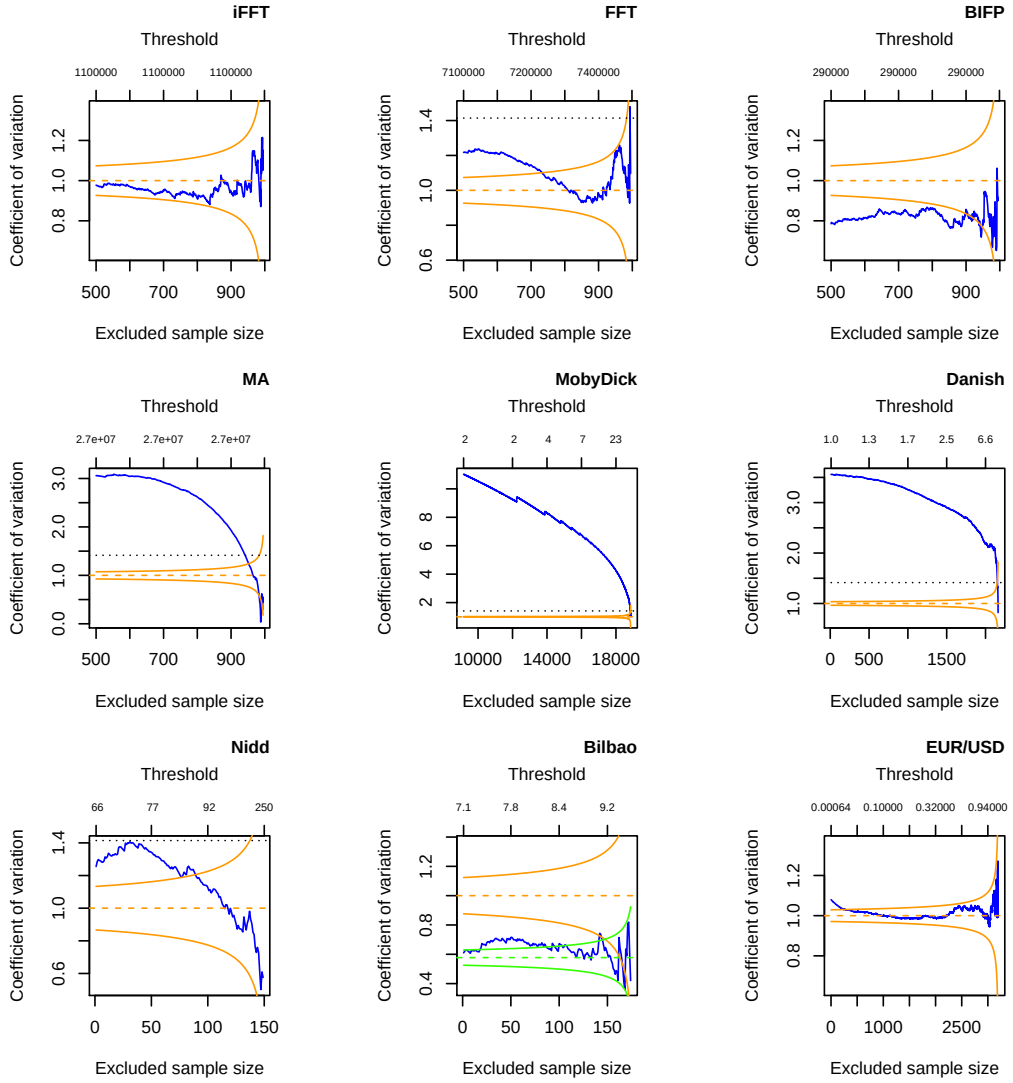


Figure 1: CV-plots stowed from left to right and top to bottom: four different types of execution time distributions of automotive applications, the frequency of words in the novel Moby Dick, Danish fire insurance data, River Nidd exceedances above value 65, Bilbao waves dataset and positive daily returns of euro/dollar exchange rates between 1999 and 2014.

Examples

The use of the `cvplot` function and its options is described using nine examples. The first four (iFFT, FFT, BIFP and MA) correspond to different types of execution time distributions observed for a set of representative programs for the analysis of automotive applications. Three others are in R packages: MobyDick ("moby" in R package **poweRlaw**), Danish and Nidd ("danish" and '"nidd.thresh" in the R package **evir**). The Bilbao waves dataset (bilbao) was originally analysed by Castillo and Hadi (1997). EURUSD is the dataset of euro/dollar daily exchange rates between 1999 and 2016.

We collect samples with $n = 1,000$ observations for 4 of the 16 benchmarks in the EEMBC AutoBench suite (Poovey, 2007), which is a well-known suite for real-time systems that includes a number of programs used in embedded automotive systems. Hereinafter, these datasets will be called iFFT (idctrn), FFT (aifftr), BIFP (basefp) and MA (matrix), leaving the real names in parentheses, they correspond respectively to *Inverse Fast Fourier Transform, Fast Fourier Transform, Basic Integer and Floating Point* and *Matrix Arithmetic*, see Abella et al. (2017) and del Castillo et al. (2017b). The histograms of the four datasets are bell-shaped. Hence, when searching for L-shaped distributions, we start the exploratory data analysis of the upper part of the distribution by taking the median as origin. Note also that large samples increase the precision of the estimates, provided that the fitted model is validated. The CV-plots for these four datasets are obtained, for instance, with:

```
library("ercv")
data(iFFT)
cvplot(iFFT,thr=median(iFFT))
```

The plots in Figure 1 are stowed from left to right and top to bottom. For iFFT, the CV-plot is inside the confidence interval of the exponential distribution ($evi = 0$). Hence, it can be assumed that the CV is constant equal to 1 (dashed orange line). For FFT, the CV-plot is inside the confidence interval for the last 250 observations. For BIFP, the CV-plot looks like a constant with CV lower than 1, hence a light tailed GPD is suggested. For MA, the CV-plot suggests a heavy tailed distribution.

The following three CV-plots in Figure 1 are made from the MobyDick, Danish and Nidd datasets, which can be directly loaded from the R packages. The three plots are made with the default `cvplot` function options, but including title, for instance:

```
data("moby", package = "poweRlaw")
cvplot(moby,main="MobyDick")
```

The second row of Figure 1 shows three examples that suggest heavy tailed distributions. In the centre is MobyDick and on the right is the Danish fire insurance dataset, which is a highly heavy-tailed infinite-variance example used to illustrate the basic ideas of extreme value theory, see Embrechts et al. (1997), McNeil et al. (2005, Example 7.23) and Novak (2012, Example 9.8). Section Transformation from heavy to light tails (tdata) shows how to analyse these examples, with the `tdata` function, using the methodology developed in del Castillo and Padilla (2016).

Nidd is the dataset of high levels of the River Nidd above a threshold value of 65. Its CV-plot is always lower than $\sqrt{2}$, begins in the area of heavy tails and goes into the confidence interval of exponentially. The Bilbao waves dataset was originally analysed by Castillo and Hadi (1997). The Nidd and Bilbao datasets are two of the most commented examples of extreme values theory, which were also analysed by del Castillo and Serra (2015) from the MLE point of view.

By default, the `cvplot` function draws a 90% confidence interval of CV-plot from exponential distribution ($evi = 0$). The *evi* parameter of the function provides confidence intervals of the corresponding GPD ($evi < 1/4$). The *conf.level* parameter allows for changing confidence levels. Both *evi* and *conf.level* may be a vector. For light tailed distributions, as is presumably the case with the wave levels, it is also advisable to draw a confidence interval from the uniform distribution ($evi = -1$). Hence, the Bilbao CV-plot in Figure 1 has confidence intervals for exponential (orange) and uniform (green) distributions.

```
data(bilbao)
cvplot(bilbao,evi = c(0,-1),main="Bilbao")
```

EURUSD is the data frame object of the euro/dollar daily exchange rates between 1999 and 2016, including the financial crisis of 2007-08, which was obtained from the R package **quantmod** (Ryan, 2016). Various parts of the EURUSD series have been studied by several authors, see Gomes and Pestana (2007) and del Castillo and Padilla (2016). The last plot in Figure 1 shows the CV-plot of the positive log-returns of the euro/dollar daily prices, obtained from

```
data("EURUSD")
prices<-ts(EURUSD$EUR.USD,frequency=365,start=1999)
```

```
#plot(prices,col="blue",main="euro/dollar daily prices(1999-2016)")
return <- 100*diff(log(prices));
pos.return <- subset(return, return >0);
cvplot(pos.return,main="pos.returns EUR/USD 1999-2016")
```

The dynamics of the daily return can be described by a GARCH(1,1) model. One might then hope that for sufficiently high values of t the subset of daily returns that are above t is so well separated in time that independence can reasonably be assumed. Then, the CV-plot clearly shows that the tail of the distribution looks like an exponential.

Estimation and Model diagnostics with `Tm` function

Following the exploratory analysis, we would like to confirm or deny some of the previous observations. It is known that in order to make optimum decisions, it is necessary to quantify the uncertainty of information extracted from data. Statistics provides mechanisms to ensure a controlled probability of error, but there is always the risk of misuse for multiple testing, especially in EVT where quite small changes can be greatly magnified on extrapolation. The asymptotic distribution of the residual coefficient of variation for GPD as a random process indexed by the threshold by del Castillo and Padilla (2016) provides pointwise error limits for CV-plot, used in the last section, and a *multiple thresholds test* that truly reduces the multiple testing problem, hence, the p-values are clearly defined.

Using the building blocks given by (10) the multiple threshold test T_m (the `Tm` function of the R package **ercv**) for a (supplementary) number of thresholds m as large as necessary for practical applications is constructed from

$$T_m(\xi) = n \sum_{k=0}^{m} p^k (cv(q_k) - c_\xi)^2, \tag{11}$$

where c_ξ is in (4), q_k are the empirical quantiles corresponding to probabilities $1 - p^k$ and probability p is chosen so that $n\, p^m \approx omit$, where $omit$ is the smaller sample size used to calculate CV. This statistic can be used to test whether a sample is distributed as a GPD with parameter ξ.

The `Tm` function makes it possible to see whether the 75 largest values of Nidd can be assumed to be exponentially distributed.

```
data("nidd.thresh",package = "evir")
Tm(nidd.thresh,evi=0, nextremes = 75)

nextremes   cvopt   evi    tms   pvalue
       75   1.000   0.000  0.981 0.310
```

The `Tm` function provides $tms = Tm(evi)/(m+1)$, which is stable on vary the number of thresholds m, the p-value says that it can not be rejected exponenciality (the number of simulations can be increased with *nsim*). Moreover, by default the `Tm` function assumes that the parameter ξ is unknown ($evi = NA$), then the *cvopt* is estimated as the value $\widetilde{c}_{\widetilde{\xi}}$ such that achieves the minimum of $T_m(\xi)$, and reversing (4) provides an estimator $\widetilde{\xi}$.

The following code shows that the assumption of constant CV (GDP) is rejected for the complete sample.

```
Tm(nidd.thresh)

nextremes   cvopt  evi    tms   pvalue
      154   1.225  0.167  1.214 0.030
```

It is rejected that Bilbao dataset is uniform distributed. However, It can not be rejected GPD as the following code shows

```
Tm(bilbao,evi=-1,nsim=1000)

nextremes   cvopt     evi    tms     pvalue
      179   0.577   -1.000   0.629   0.003

Tm(bilbao,nsim=1000)

nextremes   cvopt     evi    tms     pvalue
      179   0.650   -0.685   0.254   0.172
```

The confidence interval for the parameter estimation $evi = -0.685$ can be obtained with

```
cievi(nextremes=length(bilbao),evi=-0.685)

   5%     95%
-0.778 -0.549
```

Using a small threshold, (0.1%), the `Tm` function shows that the positive and negative returns of the euro/dollar between 1999 and 2016 can be assumed exponentially distributed.

```
Tm(pos.return,m=50,evi=0,thr=0.1,nsim=1000)

nextremes   cvopt   evi   tms    pvalue
     2207   1.000  0.000  0.392  0.780

neg.return <- -subset(return, return <0);
Tm(neg.return,m=50,evi=0,thr=0.1,nsim=1000)

nextremes   cvopt   evi   tms    pvalue
     2187   1.000  0.000  1.160  0.231
```

The last statement with Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz takes *elapsed=4.73* (R> proc.time()).

Threshold selection algorithm (`thrselect`)

There are two different approaches to the question of threshold choice. The first approach is to regard the free choice of the threshold as an advantageous feature of the procedure. By varying the threshold, the data can be explored, and if a single estimate is needed it can be obtained by subjective choice. It may well be that such a subjective approach is in reality the most useful one.

The other, to some extend opposing, view is that there is a need for an automatic method whereby the threshold is chosen by the data. It is fairer to use the word automatic rather than objective for such a method, because there are arbitrary decisions involved in the choice of the method itself. Nevertheless, it is of course the case that conditional on the automatic method being used, the threshold is indeed objective. Automatic methods need not be used in an uncritical way; they can of course be used as a starting point for fine tuning.

The `thrselect` function in the R package **ercv** starts with the $T_m(\xi)$ calculation (11) where the number of thresholds m must be fixed by the researcher. This determines the thresholds where the CV is calculated, $0 = q_0 < q_1 < \cdots < q_m$, which are fixed throughout the procedure. We accept or reject the null hypothesis for the shape parameter using all the thresholds. If the hypothesis is rejected, the threshold excesses $\left(x_j - q_1\right)$ are calculated for the sub-sample $\left\{x_j > q_1\right\}$. The previous steps are repeated, but removing one threshold, to accept or reject the null hypothesis that the sample comes from a GPD with parameter ξ, see del Castillo and Padilla (2016).

If we apply the function `thrselect` on the Nidd dataset the code shows

```
DF <- thrselect(nidd.thresh,m=10, nsim=1000)

    m nextremes threshold   rcv   cvopt   evi     tms    pvalue
5   6        63     87.85   1.193 1.073   0.0656  0.408  0.102
```

This means that the algorithm need 5 steps to achieve a p-value larger than 0.10 and it is using in this step $m = 6$ thresholds. Then, constant CV can be accepted for the last 63 extremes over the threshold 87.85, with the CV $cvopt = 1.0728$ and the corresponding $evi = 0.0656$.

The output of `thrselect` is in the data frame DF, the printed values are in DF$solution and DF$options provides complementary information that can be used for a more personal approach.

```
print(DF$options,digits=4)

    m nextremes threshold   rcv  cvopt      evi     tms    pvalue
1  10       154     65.08 1.2486 1.2249  0.166758 1.33553  0.023
2   9       123     74.38 1.4082 1.2183  0.163112 1.47158  0.012
3   8        99     77.80 1.3163 1.1634  0.130594 0.93927  0.034
4   7        79     81.40 1.2587 1.1175  0.099606 0.64548  0.064
5   6        63     87.85 1.1933 1.0728  0.065559 0.40795  0.102
```

```
6   5        50     92.82 1.1328 1.0320  0.030493 0.24415  0.217
7   4        40     99.14 1.0714 0.9945 -0.005584 0.12917  0.457
8   3        32    107.94 1.0054 0.9619 -0.040406 0.05888  0.609
9   2        26    115.93 0.9006 0.9396 -0.066323 0.04218  0.637
10  1        21    131.87 0.9473 0.9667 -0.034986 0.01755  0.597
```

Transformation from heavy to light tails (tdata)

It is possible to extend the previous methodology based on CV to all distributions, even without finite moments. For CV-plots above the straight line $y = \sqrt{2}$, like the three examples in the second row of Figure 1, the datasets are transformed by the strictly increasing function that applies $(0, \infty)$ to $(0, \sigma)$,

$$y(x) = \sigma x/(x + \sigma),$$

where $\sigma > 0$, using the tdata function in the R package **ercv**.

This technique is founded on the following result: if X is a random variable $GPD(\xi, \psi)$ distributed and $\xi > 0$, then for $\sigma = \psi/\xi$ the transformed random variable $Y = y(X)$ is $GPD(-\xi, \psi)$ distributed. Furthermore, the converse is also true, as evidenced by applying the inverse transformation $x(y) = \sigma y/(\sigma - y)$, see also del Castillo and Padilla (2016). The $\sigma > 0$ parameter is estimated by tdata, using MLE with the internal function egpd, (see del Castillo and Serra (2015)) or may be provided by the researcher as a preliminary estimate.

The CV-plots for Danish and MobyDick transformed by tdata function are obtained with:

```
data("danish",package = "evir")
tdanish<- tdata(danish)
cvplot(tdanish,main="transformed Danish")
tmoby<- tdata(moby)
cvplot(tmoby,main="transformed MobyDick")
```

The CV-plots in Figure 2 for the transformed datasets are more stable than the original CV-plots in Figure 1 and actually look light tailed. The CV-plot of the transformed MobyDick has a sawtooth profile because the original dataset only takes positive integer values and the smaller values have a high frequency (among the 18,855 values, 1 appears 9,161 times, 2 appears 3,085, ...). In order to use a GPD approach for this example we assume that the data correspond to positive values rounded to the nearest integer.

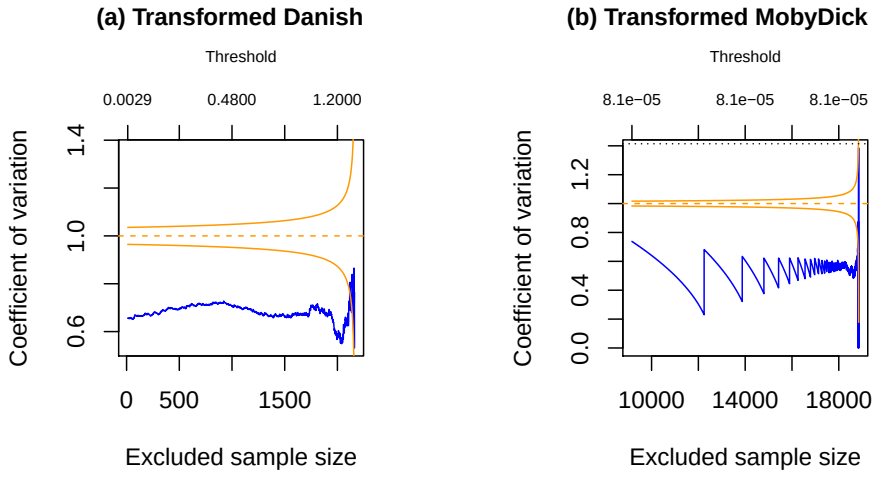


Figure 2: CV-plots under *tdata* transformation of Danish fire insurance data and frequencies of words in the novel Moby Dick.

The Tm function rejects GPD for the complete transformation of MobyDick. The same result is obtained with the transformation of the dataset on the thresholds 2 and 3. However, GPD is not rejected on threshold 4, hence the frequencies of words that appear four or more times in the novel Moby Dick (4,980 observations) can be approximated by a GPD distribution with $evi = 0.982$, as the following code shows (changing the sign of *evi*):

```
t4moby<-tdata(moby,thr=4)
Tm(t4moby,m=50,nsim=1000)

nextremes   cvopt    evi     tms    pvalue
     4980   0.581   -0.982   0.198  0.293
```

The Danish example was studied by del Castillo and Padilla (2016). The results obtained are validated by the `Tm` function after the transformation `tdata`

```
Tm(tdanish,m=20,nextremes = 951,omit = 8, nsim = 1000)

nextremes   cvopt     evi     tms      pvalue
      951   0.676    -0.595   0.256    0.253
```

Applying the `thrselect` function to Danish after the transformation by `tdata` we obtain

```
DF<-thrselect(tdanish,m=30,nsim=1000)

    m  nextremes  threshold  rcv    cvopt     evi    tms     pvalue
19 12    116      1.283      0.589  0.6747   -0.598  0.265   0.11
```

The automatic algorithm chooses the threshold 1.283 (116 extremes) with the estimate $evi = 0.598$ (changing the sign of *evi*) really close to the previous one $evi = 0.595$. The result is different from that obtained by McNeil et al. (2005) by MLE $evi = 0.50$ (109 extremes). However the *cievi* function shows that $evi = 0.50$ can not be rejected, as shown by the confidence interval provided by the following code (changing the sign of *evi* again),

```
cievi(116,evi=-0.596)

    5%    95%
-0.714 -0.440
```

In the next section we will discuss these results with new features of the R package **ercv**.

Fitting PoT parameters and tail plots (`fitpot` `ccdfplot`)

The tools described in the previous sections provide an asymptotic model for threshold exceedances over a high quantile, the so-called *peak-over-threshold* (PoT) method, see McNeil et al. (2005). The PoT method is based on determining a high enough threshold from which the distribution of the observations above this value, adjusted to zero, approaches to a GPD distribution. Then, given a threshold t, for $x > t$ the *complementary cumulative distribution function* (ccdf) is estimated by

$$1 - \hat{F}(x) = \hat{p}_t \left(1 - G\left(x - t; \hat{\xi}_t, \hat{\psi}_t\right)\right) \tag{12}$$

where $G(x; \xi, \psi)$ is the cumulative distribution function of the GDP, whose probability density function was introduced in (3), and $\left(\hat{\xi}_t, \hat{\psi}_t\right)$ are their estimated parameters for the n_t threshold exceedances over t adjusted to zero, from a sample of size n with $\hat{p}_t = n_t/n$. Alternatively, given n_t the estimated parameter is t.

The `ppot` function is the cumulative distribution function for the PoT method. That is, given an estimate of the four parameters in (12), $\left(\hat{\xi}, \hat{\psi}, \hat{t}, \hat{p}\right)$, the right hand part of (12) is provided by $1-\texttt{ppot}\left(x, \left(\hat{\xi}, \hat{\psi}, \hat{t}, \hat{p}\right)\right)$. The `qpot` is the quantile function for the PoT method that assigns to each probability p attained by `ppot` the value x for which $\texttt{ppot}(x) = p$, given the same vector of four parameters. The `qpot` function can be used in the estimation of high quantiles, that in terms of risk is expressed as the *value at risk* (VaR). For a small p, $VaR_p = q$ if and only if $1 - F(q) = p$. Hence, if $\varepsilon < \hat{p}$,

$$VaR_\epsilon = \hat{t} + qpot\left(\left(1 - \varepsilon/\hat{p}\right), \left(\hat{\xi}, \hat{\psi}, \hat{t}, \hat{p}\right)\right).$$

The `fitpot` function of the R package **ercv** provides an estimate of the four parameters in (12) that allow approximating the empirical cumulative distribution function of a dataset. It is assumed that the threshold t, or the number of extremes, has been chosen based on the tools of the previous sections. By default `fitpot` uses MLE. However, since parameter ξ (evi) can be estimated minimizing (11) by the `Tm` function, this value can be entered into the function `fitpot` and then it uses MLE by the restricted model to a single parameter. From now on this method of estimation will be called CV method.

The two methods of estimation of `fitpot` applied to Danish explain the differences between the results obtained by us and by other researchers, which we have discussed in the previous section, as we can see with the code

```
fit1<-fitpot(danish,nextremes =116);fit1                #MLE

  evi       psi     threshold   prob
0.446      7.462    9.200      0.054

fit2<-fitpot(danish,evi=0.598,nextremes =116);fit2      #CV

  evi       psi     threshold   prob
0.598      6.450    9.200      0.054
```

Naturally, different estimation methods provide different estimates, but the question of identifying the best approach still remains. To clarify this point, we can use the `ccdfplot` function, which draws the empirical complementary cumulative distribution function with the approximations provided by the parameters estimated by `fitpot`. The `ccdfplot` function allows to draw several approaches at several scales. The approximation is linear in the log-log scale for datasets with heavy tails, although it is linear in log scale for datasets with exponential tails (log = "y", by default). To draw the approach on natural scale the option *log* = "" has to be used.

The plots of Figure 3 have been obtained with `ccdfplot` function applied to Danish data with the estimates obtained by MLE (orange) and CV method (green) on logarithmic and double logarithmic scales, with

```
ccdfplot(danish,pars=list(fit1,fit2),main="Danish (log scale)")
ccdfplot(danish,pars=list(fit1,fit2),log="xy",main="Danish (log-log
scale)")
```

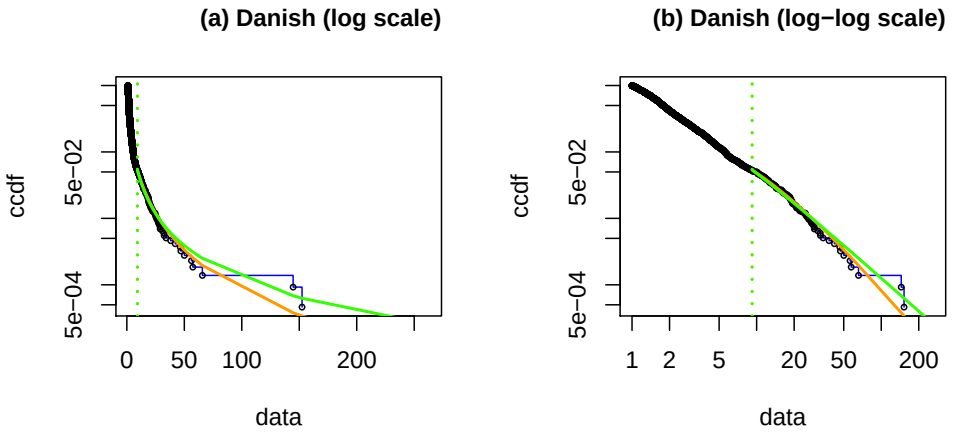


Figure 3: Complementary cumulative distribution function of Danish fire insurance data adjusted by MLE and CV methods. in log scale and log-log scale.

Figure 3 shows that both adjustments are reasonable. The CV method is not worse than MLE, perhaps less optimistic or more realistic. The previous PoT approach can be validated using the Clauset et al. (2009) point of view.

Based on the four parameters estimated by `fitpot` $(\hat{\xi}, \hat{\psi}, \hat{t}, \hat{p})$ for heavy tailed models (evi > 0), the linear relationship (7) can be obtained for the dataset values over the threshold, with the new threshold $\hat{\sigma} = \hat{\psi}/\hat{\xi}$ and the probability 1, see the following code.

```
fit1<-as.numeric(fit1$coeff);sg1<- fit1[2]/fit1[1];sg1
fit2<-as.numeric(fit2$coeff);sg2<- fit2[2]/fit2[1];sg2
exDanish<-danish[danish>fit1[3]]-fit1[3]  #origin to zero
exDanish1<- exDanish+sg1                  #origin to sg1
exDanish2<-exDanish+sg2                   #origin to sg2
exfit1<-c(fit1[1],fit1[2],sg1,1)
exfit2<-c(fit2[1],fit2[2],sg2,1)
ccdfplot(exDanish, pars=c(exfit1),log="xy",main="adjusted by MLE")
ccdfplot(exDanish2, pars=c(exfit2),log="xy",main="adjusted by CV")
```

The Figure 4 plot (a) shows the linear relationship (7) for the 116 upper extremes of Danish adjusted by MLE. Changing the previous *fit1* by *fit2* the linear relationship is obtained by the CV method and is

shown in plot (b). Notice that the linear relationship (7) begins at the threshold $sg1 = 16.727$ for MLE and at a threshold $sg2 = 10.787$ for the CV method, so we can not overlay them in the same graph. The goodness of fit can now be measured by the correlation between the logarithm of the complementary empirical distribution function, $log(1 - F_n)$ and the logarithm of the data, $\log(x + sg)$, where $(x + sg)$ are the 116 upper extremes of Danish, adjusted to sigma. The results are $correlation = -0.981$ using MLE, plot (a), and $correlation = -0.990$ using CV-method, plot (b).

We can also calculate the threshold th having a maximum correlation between $log(1 - F_n)$ and $log(x + th)$, obtaining $th = 6.996$ and $correlation = -0.992$. Thus, the correlation on which the goodness of the CV method adjustment is based on is very close to the best that can be obtained by this procedure, which is in line with Clauset et al. (2009) and the **poweRlaw** R package by Gillespie (2015) (although here the estimation of evi is different). This shows that the methodology provided by the R package **ercv** complements and connects the contributions of **evir** (Pfaff and McNeil, 2012) and **poweRlaw** by Gillespie (2015).

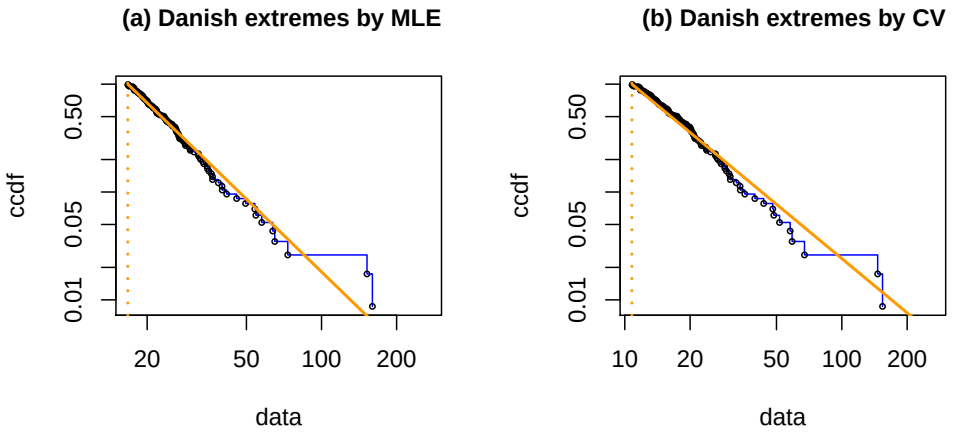


Figure 4: The linear relationship for the 116 upper extremes of Danish fire insurance data adjusted by MLE and CV method.

Acknowledgements

This work was supported by the Spanish Ministry of Economy and Competitiveness under Grant: Statistical modelling of environmental, technological and health risks, MTM2015-69493-R. David Moriña acknowledges financial support from the Spanish Ministry of Economy and Competitiveness, through the María de Maeztu Programme for Units of Excellence in R&D (MDM-2014-0445) and from Fundación Santander Universidades.

Bibliography

J. Abella, M. Padilla, J. del Castillo, and F. Cazorla. Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(4):72:1–72:29, 2017. [p60]

A. Balkema and L. de Haan. Residual life time at great age. *Annals of Probability*, 2:792–804, 1974. [p57]

J. Beirlant, Y. Goegebeur, J. Segers, and J. L. Teugels. *Statistics of Extremes: Theory and Applications*. John Wiley & Sons, Chichester, UK, 2006. [p56]

E. Castillo and A. S. Hadi. Fitting the generalized pareto distribution to data. *Journal of the American Statistical Association*, 92:1609 – 1620, 1997. [p60]

A. Clauset, C. R. Shalizi, and M. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009. [p56, 58, 65, 66]

S. Coles. *An Introduction to Statistical of Extremes Values*. Springer-Verlag, London, 2001. [p56, 57]

L. de Haan and A. Ferreira. *Extreme Value Theory: An Introduction*. Springer-Verlag, New York, 2007. [p56]

J. del Castillo and M. Padilla. Modelling extreme values by the residual coefficient of variation. *Statistics and Operations Research Transactions*, 40:303–320, 2016. [p56, 58, 59, 60, 61, 62, 63, 64]

J. del Castillo and I. Serra. Likelihood inference for generalized pareto distribution. *Computational Statistics & Data Analysis*, 83:116–128, 2015. [p56, 58, 60, 63]

J. del Castillo, J. Daoudi, and R. Lockhart. Methods to distinguish between polynomial and exponential tails. *Scandinavian Journal of Statistics*, 41:382–393, 2014. URL https://doi.org/10.1111/sjos.12037. [p56, 58]

J. del Castillo, D. Moriña, and I. Serra. **ercv***: Fitting Tails by the Empirical Residual Coefficient of Variation*, 2017a. URL https://CRAN.R-project.org/package=ercv. R package version 1.0.0. [p56]

J. del Castillo, M. Padilla, J. Abella, and F. Cazorla. Execution time distributions in embedded safety-critical systems using extreme value theory. *International Journal of Systems Control and Information Processing*, 9(4):348–361, 2017b. [p60]

P. Embrechts, C. Klüppelberg, and T. Mikosch. *Modelling Extremal Events for Insurance and Finance*. Springer-Verlag, Berlin, 1997. [p56, 60]

R. Fisher and L. Tippett. Limiting forms of the frequency distribution of the largest or smallest member of a sample. *Mathematical Proceedings of the Cambridge Philosophical Society*, 24(2):180–190, 1928. [p57]

E. Gilleland, M. Ribatet, and A. Stephenson. A software review for extreme value analysis. *Extremes*, 16:103–119, 2013. [p56]

C. Gillespie. Fitting heavy tailed distributions: The powerlaw package. *Journal of Statistical Software*, 64(2):1–16, 2015. [p56, 58, 66]

B. V. Gnedenko. Sur la distribution limite du terme maximum d'une serie aleatoire. *The Annals of Mathematics*, 44(3):423–453, 1943. [p57]

M. I. Gomes and D. Pestana. A sturdy reduced-bias extreme quantile (var) estimator. *Journal of the American Statistical Association*, 102:280–292, 2007. [p60]

R. Gupta and S. N. U. A. Kirmani. Residual coefficient of variation and some characterization results. *Journal of Statistical Planning and Inference*, 91:23–31, 2000. [p58]

J. E. Heffernan and A. G. Stephenson. *Ismev: An Introduction to Statistical Modeling of Extreme Values*, 2018. URL https://CRAN.R-project.org/package=ismev. [p56]

R. Leadbetter, G. Lindgren, and H. Rootzén. *Extremes and Related Properties of Random Sequences and Processes*. Springer-Verlag, New York, 1983. [p56]

N. Markovich. *Nonparametric Analysis of Univariate Heavy-Tailed Data*. John Wiley & Sons, Chichester, UK, 2007. [p56]

A. McNeil, R. Frey, and P. Embrechts. *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton Series in Finance, New Jersey, 2005. [p57, 60, 64]

S. Novak. *Extreme Value Methods with Applications to Finance*. CRC Press, Boca Raton, 2012. [p56, 60]

B. Pfaff and A. McNeil. **evir***: Extreme Values in R*, 2012. URL https://CRAN.R-project.org/package=evir. R package version 1.7-3. [p56, 66]

J. Pickands. Statistical inference using extreme order statistics. *The Annals of Statistics*, 3:119–131, 1975. [p57]

J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, Raleigh, NC, 2007. [p60]

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/. [p56]

S. Resnick. *Extreme Values,Regular Variation,and Point Processes*. Springer-Verlag, New York, 2013. [p56]

P. Ruckdeschel, M. Kohl, and N. Horbenko. *RobExtremes: Optimally Robust Estimation for Extreme Value Distributions*, 2019. URL http://robast.r-forge.r-project.org/. Contributions by S. Desmettre, G. Kroisandt, E. Massini, D. Pupashenko and B. Spangl; R package version 1.2.0. [p59]

J. A. Ryan. **quantmod**: *Quantitative Financial Modelling Framework*, 2016. URL https://CRAN.R-project.org/package=quantmod. R package version 0.4-7. [p60]

Joan del Castillo
Departament de Matemàtiques, Universitat Autònoma de Barcelona (UAB)
Edifici C, E-08193 Barcelona
Spain
castillo@mat.uab.cat

Isabel Serra
Centre de Recerca Matemàtica (CRM)
Edifici C, E-08193 Barcelona
Spain
iserra@crm.cat

Maria Padilla
Departament de Matemàtiques, Universitat Autònoma de Barcelona (UAB)
Edifici C, E-08193 Barcelona
Spain
mpadilla@mat.uab.cat

David Moriña
Barcelona Graduate School of Mathematics (BGSMath), Departament de Matemàtiques, Universitat Autònoma de Barcelona (UAB)
Edifici C, E-08193 Barcelona
Spain
dmorina@mat.uab.cat

biclustermd: An R Package for Biclustering with Missing Values

by John Reisner, Hieu Pham, Sigurdur Olafsson, Stephen Vardeman and Jing Li

Abstract Biclustering is a statistical learning technique that attempts to find homogeneous partitions of rows and columns of a data matrix. For example, movie ratings might be biclustered to group both raters and movies. **biclust** is a current R package allowing users to implement a variety of biclustering algorithms. However, its algorithms do not allow the data matrix to have missing values. We provide a new R package, **biclustermd**, which allows users to perform biclustering on numeric data even in the presence of missing values.

Introduction

Traditional (one-way) clustering (such as with complete-link hierarchical clustering or k-means) aims to partition only rows (or columns) of a data matrix into homogeneous subsets. Rows or columns are clustered simply based upon their relational similarity to other observations. Biclustering simultaneously groups rows and columns to identify homogeneous "cells". Biclustering is known to be NP-hard; as such, every existing algorithm approaches this problem heuristically. This methodology was first investigated by Hartigan (1972) but was not given much attention until applied to gene expression data (Cheng and Church, 2000). Today, biclustering is applied across many areas such as biomedicine, text mining, and marketing (Busygin et al., 2008).

For our purposes, we consider rearranging a data matrix to obtain a checkerboard-like structure where each cell is as homogeneous as possible. In this regard, our algorithm has the same goal as spectral biclustering (Kluger et al., 2003), but approaches the problem in a different way. In contrast to clustering with the end goal being a checkerboard-like structure, other techniques have been proposed based on the singular value decomposition (Lazzeroni and Owen, 2002; Bergmann et al., 2003) and others are based on a graph-theoretic approach (Tan and Witten, 2014). Although each technique is different, each has the goal of finding substructure within the data matrix. In Figure 1 we provide a visual suggestion of our biclustering goal. The color scheme represents similar numeric values and our goal is to rearrange the data matrix so that these values form homogeneous cells.

RawDataMatrix0805ShuffledDataMatrix0805

Figure 1: Biclustering with *checkerboard-like* structure

A publicly available R package for biclustering is **biclust** by Kaiser and Leisch (2008). This appears to be a commonly used package developed with the intent of allowing users to choose from a variety of algorithms and renderable visualizations. Other biclustering packages include **superbiclust**, **iBBiG QUBIC**, **s4vd**, **BiBitR** which each provide unique algorithms and implementations (Khamiakova, 2014; Gusenleitner and Culhane, 2019; Zhang et al., 2017; Sill and Kaiser, 2015; Ewoud, 2017). However, from an implementation and algorithmic standpoint, the methods implemented in these packages fail when given a data matrix with missing values. This is clearly a limitation since there exist many rectangular datasets with missing values. For handling missing data, many imputation methods exist in the literature. While this does produce a complete two-way data table, which can subsequently be fully analyzed using existing biclustering algorithms, it has inherent limitations. When large percentages of data are missing, such as is, for example, common in plant breeding and movie rating applications to be discussed later, it is difficult and impossible to reasonably infer missing values. Even if a small number of values are missing values those are potentially missing not-at-random due to non-random and unknown devices. For example, in plant breeding, observation may be missing because it is unreasonable to plant a crop in a particular environment or simply because a plant breeder decides to not plant in certain environments. In these cases, imputing missing values would imply that one can confidently estimate the performance of (say) crop yield in an environment where it was never observed growing. There is a large body of literature on the difficult nature of this problem. With this as motivation, our goal was to produce a biclustering algorithm which can successfully deal with data with missing values without applying imputation or making any assumptions about why data are missing.

Biclustering with missing data

The package described in this paper, **biclustermd**, implements the biclustering algorithm of Li et al. (2020) and their paper gives a thorough explanation of the proposed biclustering algorithm as well as its applicability. For completeness we give an overview of their algorithm here.

Notation

- X is a data matrix with I rows and J columns. X_{ij} is a response measure of row i in column j for $i \in \{1, 2, \ldots, I\}$ and $j \in \{1, 2, \ldots, J\}$.
- Row index set $\mathcal{I} = \{1, 2, \ldots, I\}$ is partitioned into r mutually exclusive and exhaustive sets $T_1, T_2, \ldots, T_r$. $\mathcal{Q} \equiv$ partition of the row index set.
- Column index set $\mathcal{J} = \{1, 2, \ldots, J\}$ is partitioned into c mutually exclusive and exhaustive sets $S_1, S_2, \ldots, S_c$. $\mathcal{P} \equiv$ partition of the column index set.

Our goal for biclustering is to generate a rearranged data matrix with a checkerboard structure such that each "cell" of the matrix defined by $\mathcal{Q}$ and $\mathcal{P}$ is as homogeneous as possible. Depending on specifics of a real problem, "homogeneous" can have different subject matter meanings, and hence optimization of different objective functions can be appropriate. We present our algorithm here with the goal of optimizing a total within-cluster sum of squares given both the row groups in $\mathcal{Q}$ and column groups in $\mathcal{P}$. This can be interpreted as the total sum of squared errors between cell means and data values within cells. Hence we refer to this as SSE. Using the above notations we have r row groups (or row clusters) and c column groups (or column clusters). Let A denote an $r \times c$ "cell-average matrix" with entries

$$A_{mn} \equiv \frac{1}{|\{X_{ij} : i \in T_m;\ j \in S_n;\ X_{ij} \neq NA\}|} \sum_{\{X_{ij}: i \in T_m;\ j \in S_n;\ X_{ij} \neq NA\}} X_{ij} \tag{1}$$

for $m \in 1, 2, \ldots, r$ and $n \in 1, 2, \ldots, c$. Here, $|\cdot|$ is the set cardinality function and NA denotes a missing value. Then, the within-cluster sum of squares function to be minimized is

$$\text{SSE} \equiv \sum_{m,n} \sum_{\substack{X_{ij} \neq NA \\ i \in T_m \\ j \in S_n}} \left(X_{ij} - A_{mn}\right)^2. \tag{2}$$

Biclustering with missing data algorithm

1. Randomly generate initial partitions $\mathcal{Q}^{(0)}$ and $\mathcal{P}^{(0)}$ with respectively r row groups and c column groups.
2. Create a matrix $A^{(0)}$ using Equation (1) and the initial partitions. In the event that a "cell" (m, n) defined by $\{(i, j) \,|i \in T_m \text{ and } j \in S_n\}$ is empty, A_{mn} can be set to some pre-specified constant or some function of the numerical values corresponding to the non-empty cells created by the partition. (For example, the mean of the values coming from non-empty cells in row m or in column n can be used.) This algorithmic step should not be seen as imputation of responses for the cell under consideration, but rather only a device to keep the algorithm running.
3. At iteration s of the algorithm, with partitions $\mathcal{P}^{(s-1)}$ and $\mathcal{Q}^{(s-1)}$ and corresponding matrix $A^{(s-1)}$ in hand, for $i = 1, 2, \ldots, I$ let

   $$M_{in}^R = \frac{1}{\left|\left\{j \in S_n | X_{ij} \neq NA\right\}\right|} \sum_{\substack{j \in S_n \\ \text{s.t. } X_{ij} \neq NA}} X_{ij}$$

   for each $n = 1, 2, \ldots, c$ and compute for $m = 1, 2, \ldots, r$

   $$d_{im}^R = \sum_{n=1}^{c} \left(A_{mn} - M_{in}^R\right)^2 \cdot \left|\left\{j \in S_n | X_{ij} \neq NA\right\}\right|.$$

   Then create $\mathcal{Q}^{(s)*}$ by assigning each row i to T_m with minimum d_{im}^R.
4. If for $\mathcal{Q}^{(s)*}$ every T_m is non-empty, proceed to Step 5. If at least one $T_m = \emptyset$ do the following:

(a) Randomly choose a row group $T_{m'}$ with $|T_{m'}| > k^R_{\min}$ (a user-specified positive integer parameter) and choose $k^R_{\text{move}} < k^R_{\min}$ row indices to move to one empty T_m. Choose those indices i from $T_{m'}$ with the largest k^R_{move} corresponding values of the sum of squares

$$\sum_{n=1}^{c} \sum_{\substack{j \in S_n \\ \text{s.t. } X_{ij} \neq NA}} \left(X_{ij} - M^R_{in}\right)^2.$$

(b) If after the move in (a) no empty row group remains, proceed to Step 5. Otherwise return to (a).

5. Replace $\mathcal{Q}^{(s-1)}$ in Step 3 with the updated version of $\mathcal{Q}^{(s)*}$ and cycle through Steps 3 and 4 α times, where α is a user-specified integer parameter. **If** `row_shuffles` > 1**, replace** $\mathcal{Q}^{(s-1)}$ **in 3. with the updated version of** $\mathcal{Q}^{(s)*}$ **and cycle through steps 3. and 4.** `row_shuffles`-1 **times.**
6. Set $\mathcal{Q}^{(s)} = \mathcal{Q}^{(s)*}$. Then update $A^{(s-1)}$ to $A^{(s)*}$ using the partitions $\mathcal{Q}^{(s)}$ and $\mathcal{P}^{(s-1)}$ in Equation (1).
7. For $j = 1, 2, \ldots, J$ let

$$M^C_{jm} = \frac{1}{\left|\left\{i \in T_m | X_{ij} \neq NA\right\}\right|} \sum_{\substack{i \in T_m \\ \text{s.t.} X_{ij} \neq NA}} X_{ij}$$

for each $m = 1, 2, \ldots, r$ and compute for $n = 1, 2, \ldots, c$

$$d^C_{jn} = \sum_{m=1}^{r} \left(A_{mn} - M^C_{jm}\right)^2 \cdot \left|\left\{i \in T_m | X_{ij} \neq NA\right\}\right|.$$

Then create $\mathcal{P}^{(s)*}$ by assigning each column j to S_n with minimum d^C_{jn}.

8. If for $\mathcal{P}^{(s)*}$ every S_n is non-empty, proceed to Step 9. If at least one $S_n = \emptyset$ do the following:

(a) Randomly choose a column group $S_{n'}$ with $|S_{n'}| > k^C_{\min}$ (a user-specified positive integer parameter) and choose $k^C_{\text{move}} < k^C_{\min}$ column indices to move to one empty S_n. Choose those indices j from $S_{n'}$ with the largest k^C_{move} corresponding values of the sum of squares

$$\sum_{m=1}^{r} \sum_{\substack{i \in T_m \\ \text{s.t. } X_{ij} \neq NA}} \left(X_{ij} - M^C_{jm}\right)^2.$$

(b) If after the move in (a) no empty column group remains, proceed to Step 9. Otherwise return to (a).

9. Replace $\mathcal{P}^{(s-1)}$ in Step 3 with the updated version of $\mathcal{P}^{(s)*}$ and cycle through Steps 7 and 8 β times, where β is a user-specified integer parameter. If `col_shuffles` > 1, replace $\mathcal{P}^{(s-1)}$ in 3. with the updated version of $\mathcal{P}^{(s)*}$ and cycle through steps 7. and 8. `col_shuffles`-1 times.
10. Set $\mathcal{P}^{(s)} = \mathcal{P}^{(s)*}$ and we have new partitions $\mathcal{Q}^{(s)}$ and $\mathcal{P}^{(s)}$. Then update $A^{(s)*}$ to $A^{(s)}$ using the partitions $\mathcal{Q}^{(s)}$ and $\mathcal{P}^{(s)}$ in Equation (1).
11. Steps 3–10 are executed N times or until the algorithm converges, which is when the Rand Indices for successive row and column partitions are both 1. (See the description of the Rand Index below.)

Intuitively, our proposed algorithm is nothing more than a rearrangement of rows and columns with the objective to minimize the objectives given in Steps 3 and 7. We consider Step 1 (the random generation of initial cluster assignments) to be of high importance to avoid any bias in the original structure of the data. As a quantitative way to measure the effectiveness of our biclustering, we consider the sum of squared errors (SSE) as the measure of within cell homogeneity. Paired with the SSE, we allow for three different convergence criteria, the Rand Index (Rand, 1971), the Adjusted Rand Index (Hubert and Arabie, 1985), and the Jaccard Index (Goodall, 1966). These indices provide measures for the similarity between two clusterings.

Overview of biclustermd

The **biclustermd** package consists of six main functions with the most important being `bicluster()`. This function is where the algorithmic process is embedded and contains numerous tunable parameters.

- **data**: dataset to bicluster. Must be a data matrix/table with only numbers and missing values in the dataset. It should have row names and column names.
- **row_clusters**: The number of clusters to partition the rows into. Default is $\lfloor \sqrt{I} \rfloor$
- **col_clusters**: The number of clusters to partition the columns into. Default is $\lfloor \sqrt{J} \rfloor$
- **missing_val**: Value or function used to represent empty cells of the data matrix. If a value, a random normal variable centered at itself with standard deviation `miss_val_sd` is used each iteration. Note that this is not data imputation but a temporary value used by the algorithm.
- **missing_val_sd**: Standard deviation of the normal distribution `miss_val` follows if `miss_val` is a number. By default this equals 1.
- **similarity**: The metric used to compare two successive clusterings. Can be "Rand" (default), "HA" for the Hubert and Arabie adjusted Rand index or "Jaccard". See **clues** for details.
- **row_min_num**: Minimum row cluster size in order to be eligible to be chosen when filling an empty row cluster. Default is $\lfloor I/r \rfloor$.
- **col_min_num**: Minimum column cluster size in order to be eligible to be chosen when filling an empty column cluster. Default is $\lfloor J/c \rfloor$.
- **row_num_to_move**: Number of rows to remove from the sampled cluster to put in an empty row cluster. Default is 1.
- **col_num_to_move**: Number of columns to remove from the sampled cluster to put in an empty column cluster. Default is 1.
- **row_shuffles**: Number of times to shuffle rows in each iteration. Default is 1.
- **col_shuffles**: Number of times to shuffle columns in each iteration. Default is 1.
- **max.iter**: Maximum number of iterations to let the algorithm run.
- **verbose**: Logical. If TRUE, will report iteration progress.

In the following sections, we provide an overview of the functionality of **biclustermd**. For the first dataset, we display the array of visualizations available, in the second example we demonstrate the impact of numerous tunable parameters, our final example demonstrates the computational times of our algorithm.

Example with NYCflights13

For a first example, we will utilize the flights dataset from Wickham's package **nycflights13** (Wickham, 2017). Per the package documentation, `flights` contains data on all flights in 2013 that departed NYC via JFK, LaGuardia, or Newark. The variables of interest are `month`, `dest`, and `arr_delay` these are the rows, columns and response value, respectively. In a dataset such as this, an application of biclustering would be to determine if there exist subsets of months and airports with similar numbers of delays. From a pragmatic perspective, this discovery may allow for air officials to investigate the connection between these airports and months and why delays are occurring.

Using functions from **tidyverse** (Wickham, 2016), we generate a two-way data table such that rows represent months, columns represent destination airports, and the numeric response values are the average arrival delays in minutes. This data matrix contains 12 rows (months), 105 columns (destination airports), and approximately 11.7% missing observations. Below is a snippet of our data matrix.

```
> flights[1:5,1:5]
               ABQ      ACK      ALB ANC       ATL
January         NA       NA 35.17460  NA  4.152047
February        NA       NA 17.38889  NA  5.174092
March           NA       NA 17.16667  NA  7.029286
April    12.222222       NA 18.00000  NA 11.724280
May      -6.516129 3.904762 10.19643  NA  8.187036
```

The first step is to determine the number of clusters for months and the number of clusters for destination airports. Since we are clustering months, in this analysis, choosing $r = 4$ row clusters seems reasonable (create a group for each season/quarter of the year). Although this is arbitrary, we choose $c = 6$ column clusters. Since this algorithm incorporates purposeful randomness (by row and column cluster initialization), `biclustermd()` should be run multiple times keeping the result with the lowest sum of squared errors (SSE) since it may be expected that for different initialization one can obtain a different local minimum (Li et al., 2020).

```
> bc <- biclustermd(data = flights, col_clusters = 6, row_clusters = 4,
+                   miss_val = mean(flights, na.rm = TRUE), miss_val_sd = 1,
+                   col_min_num = 5, row_min_num = 3,
+                   col_num_to_move = 1, row_num_to_move = 1,
+                   col_shuffles = 1, row_shuffles = 1,
+                   max.iter = 100)
> bc

Data has 1260 values, 11.75% of which are missing
10 Iterations
Initial SSE = 186445; Final SSE = 82490
Rand similarity used; Indices: Columns (P) = 1, Rows (Q) = 1
```

The output of `biclustermd()` is a list of class "biclustermd" and "list" containing the following:

- The two-way table of data provided to the function.
- The final column and row partition matrices.
- SSE generated from the initial partitioning.
- SSE of each iteration, as an "biclustermd_sse" object.
- Similarity measures for rows and columns for each iteration, as an "biclustermd_sim" object.
- The number of iterations to convergence.
- A table of resulting cell means.

Analyzing the NYCflights13 biclustering

The list output of `biclustermd()` is used for rendering plots and to obtain cell information. One such visual aid is a plot of the convergence indices versus iteration, given in Figure 2. From this graphic, we can determine the rate at which convergence occurs for both row and column clusters. Moreover, this provides confirmation that our algorithm can indeed achieve good clusterings along both dimensions. Plotting of the similarity measures and SSE is done with `autoplot.biclustermd_sim()` and `autoplot.biclustermd_sse()`, methods added to `autoplot()` of **ggplot2** (Wickham, 2009).

```
> autoplot(bc$Similarities, ncol = 3) +
+   theme_bw() +
+   theme(aspect.ratio = 1) +
+   scale_x_continuous(breaks = 0:9)
```

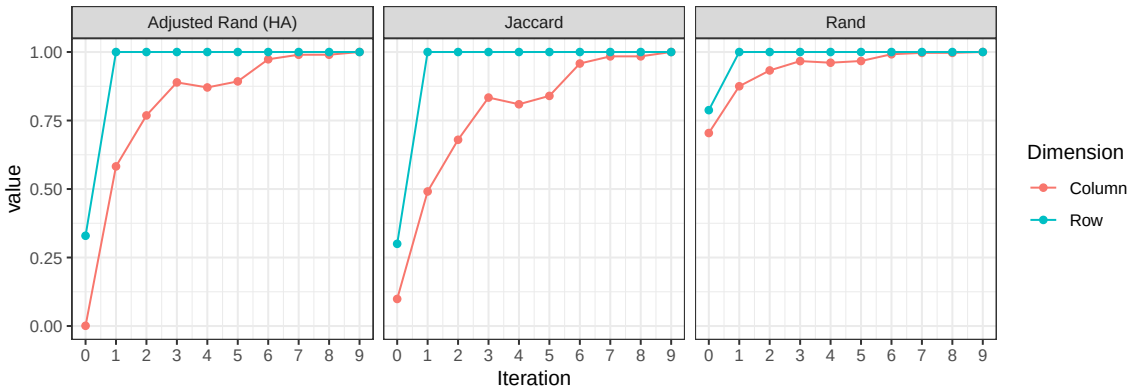


Figure 2: Plot of similarity measures for the flights biclustering

In addition to the similarity plots, one can utilize the SSE graphic as an indication of convergence to a (local) minimum biclustering. This can be seen in Figure 3. From this we can observe the rate of decrease of the SSE as well as the relative difference between the first and final iteration. Observing closely each of the three convergence criteria suddenly decrease in value along the columns, namely from iteration three to four. The algorithm is simply (attempting to) obtain a lower SSE which may result in column shuffles which differ from iteration to iteration.

```
> autoplot(bc$SSE) +
+   theme_bw() +
```

```
+   theme(aspect.ratio = 1) +
+   scale_y_continuous(labels = comma) +
+   scale_x_continuous(breaks = 0:9)
```

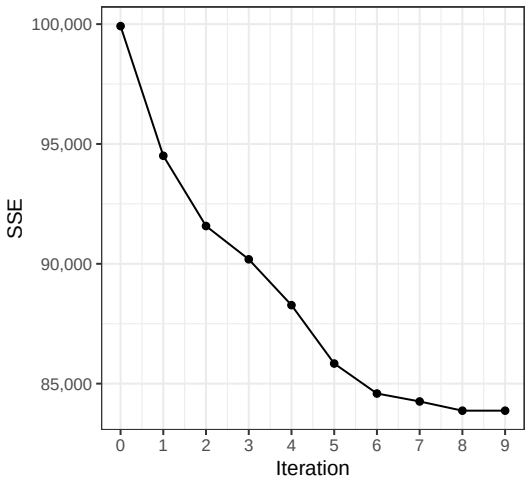


Figure 3: SSE plot of flights biclustering

Traditionally visualizations of biclustering plots are in a heat map fashion. `autoplot.biclustermd()` makes visual analysis of biclustering results easy by rendering a heat map of the biclustered data and allows for additional customization. Each of Figure 4–7 provide an example of the flexibility of this function. Recall that the algorithm uses purposeful randomness, so a replicated result may look different.

In Figure 4, we provide the default visualization without additional parameters. The white space represent cells without any observations which is directly useful for our interpretation, and the color scale is represented on the same spread as the numerical response.

```
> autoplot(bc) +
+   scale_fill_viridis_c(na.value = 'white') +
+   labs(x = "Destination Airport",
+        y = "Month",
+        fill = "Average Delay")
```

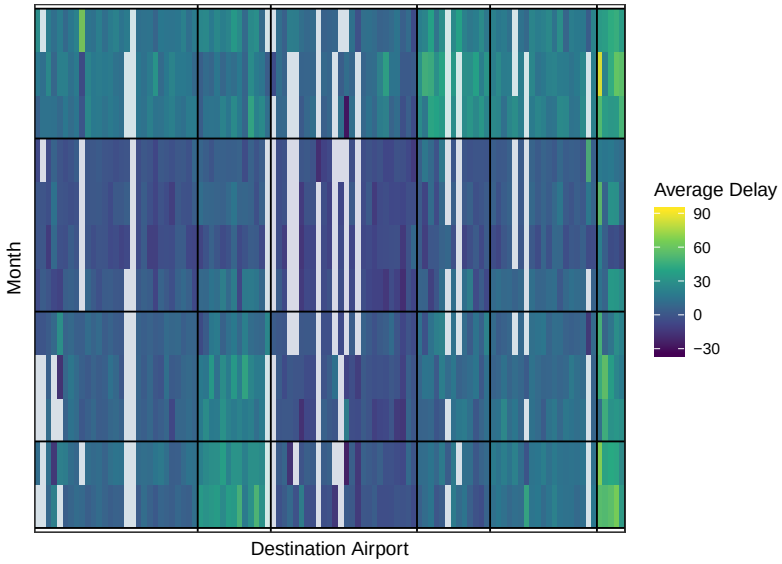


Figure 4: A heat map of the flights biclustering without transforming colors.

Often it may aid in interpretation to run the data through an *S*-shaped function before plotting. Two parameter arguments in `autoplot()` are `transform_colors = TRUE` and `c` where `c` is the constant

to scale the data by before running it through a standard normal cumulative distribution function. See Figure 5 for an illustration. Applying this transformation, one can immediately notice the distinct dissimilarity between cells that were not clearly present in Figure 4.

```
> autoplot(bc, transform_colors = TRUE, c = 1/15) +
+   scale_fill_viridis_c(na.value = 'white') +
+   labs(x = "Destination Airport",
+        y = "Month",
+        fill = "Average Delay")
```

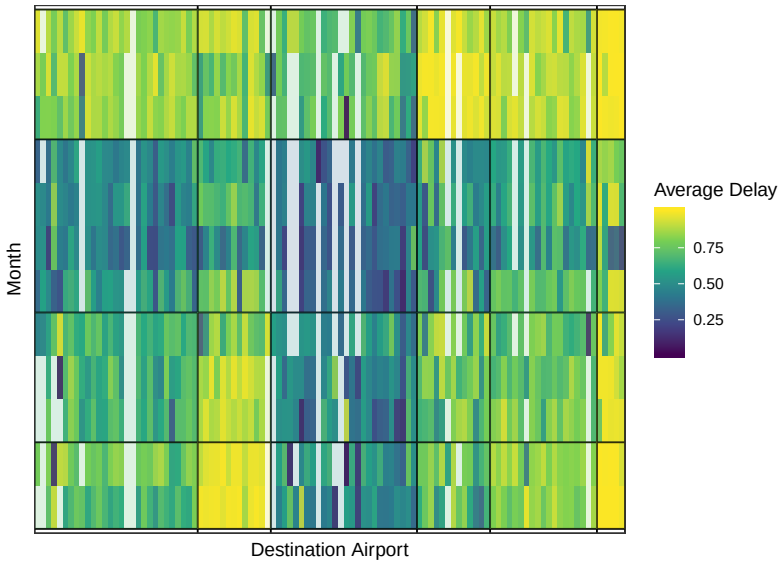


Figure 5: A heat map of the flights biclustering after transforming colors.

To further aid interpretations, we make use of `reorder_biclust` in Figure 6. This command reorders row and column clusters from increasing to decreasing mean. In our fights dataset, this may be particularly useful to determine if there is a slow shift in airport locations moving from a high to low number of delays.

```
> autoplot(bc, reorder = TRUE, transform_colors = TRUE, c = 1/15) +
+   scale_fill_viridis_c(na.value = 'white') +
+   labs(x = "Destination Airport",
+        y = "Month",
+        fill = "Average Delay")
```

Lastly, with large heat maps the authors have found it useful to zoom into selected row and column clusters. In Figure 7, row clusters three and four and column clusters one and four are shown, using the `row_clusts` and `col_clusts` arguments of `autoplot()`. Colors are not transformed.

```
> autoplot(bc, col_clusts = c(3, 4), row_clusts = c(1, 4)) +
+   scale_fill_viridis_c(na.value = 'white') +
+   labs(x = "Destination Airport",
+        y = "Month",
+        fill = "Average Delay")
```

There are two additional visualizations that provide insight into the quality of each cell: `mse_heatmap()` and `cell_heatmap()`. `mse_heatmap()` gives the mean squared error (MSE) of each cell. Here, MSE is defined as the mean squared difference between data values and the mean in each cell. Whereas `cell_heatmap()` provides a heatmap with the total number of observations in the given cell. Combined, these tools provide valuable insight into the homogeneity of each cell.

```
> mse_heatmap(bc) +
+   theme_bw() +
+   scale_fill_viridis_c() +
+   labs(fill = "Cell MSE") +
+   scale_x_continuous(breaks = 1:6)
```

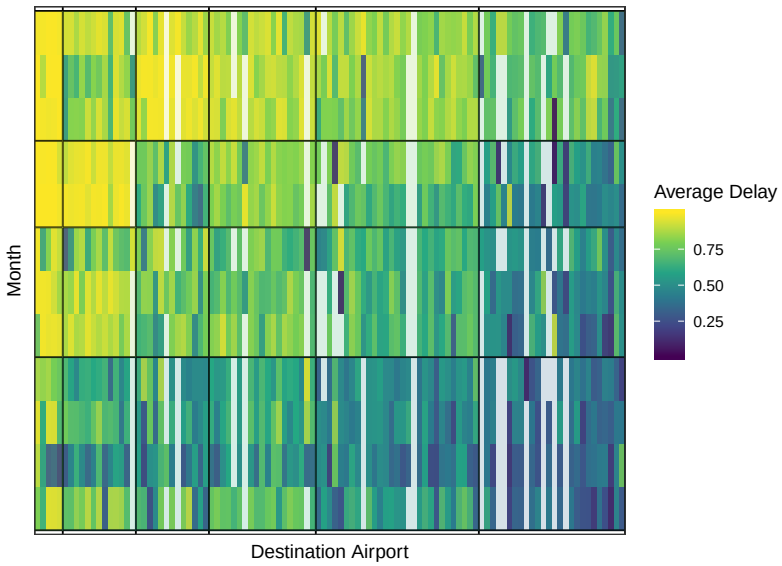


Figure 6: An ordered heat map of the flights biclustering after transforming colors.

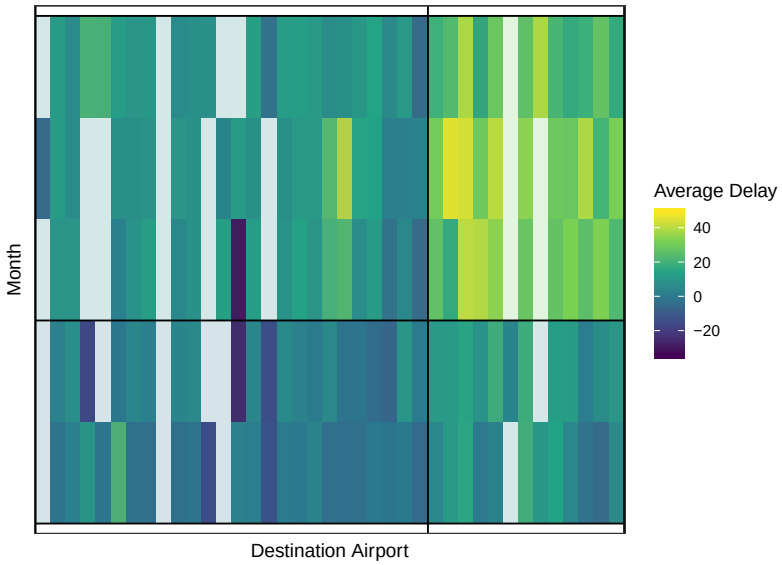


Figure 7: A zoomed in view of the heat map of the biclustering.

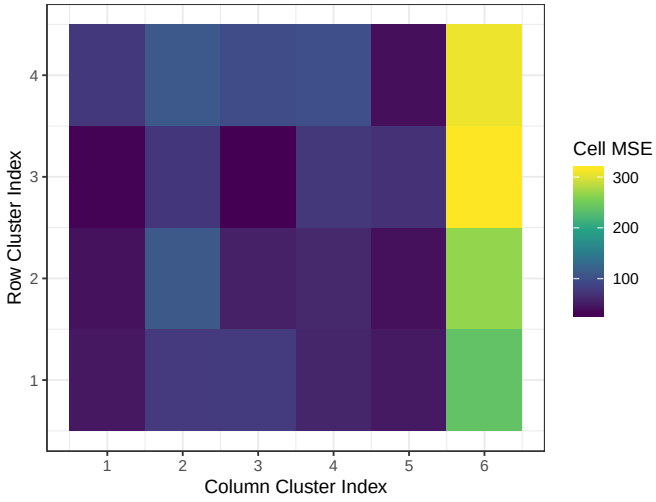


Figure 8: A heat map of cell MSEs for the flights biclustering

```
> cell_heatmap(bc) +
+   theme_bw() +
+   scale_fill_viridis_c()
```

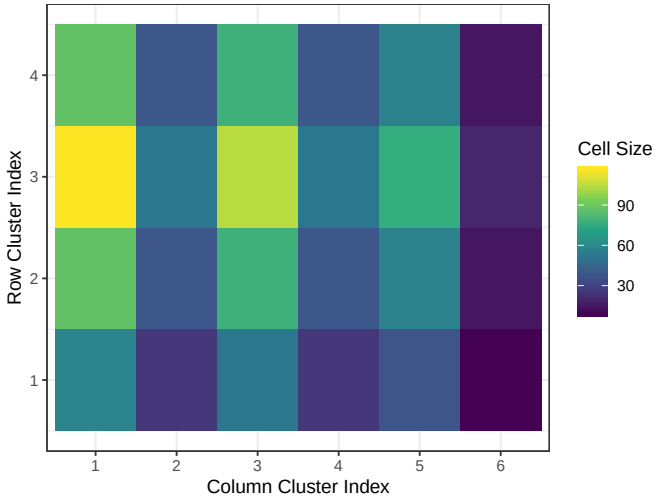


Figure 9: A heat map of cell sizes for the flights biclustering

Finally, for interpretation purposes, retrieving row or column names and their corresponding clusters is easily done using the `biclustermd` method of `row.names()` (for rows) and use of a new generic `col.names()` and its method `col.names.biclustermd()` (for columns). Two final examples are given below showing the output of each function, which have class `data.frame`.

```
> row.names(bc) %>% head()
  row_cluster     name
1           1  January
2           1    April
3           2 February
4           2    March
5           2   August
6           3      May

> col.names(bc) %>% head()
  col_cluster name
1           1  ABQ
2           1  ACK
3           1  AUS
```

```
4           1  AVL
5           1  BGR
6           1  BQN
```

Further capabilities

As previously mentioned, due to the purposeful randomness of initial row and column clusterings, multiple runs of the algorithm can produce different results. Hence it is recommended to perform several trials (with various parameters) and store the result which obtains the lowest SSE. These multiple runs can easily be done in parallel using the `tune_biclustermd()` function with the parameters as listed below. To utilize this, first a tuning grid must be defined as an input for `tune_biclustermd()`. Below we provide an illustration of the process.

- **data**: Dataset to bicluster. Must to be a data matrix with only numbers and missing values in the data set. It should have row names and column names.
- **nrep**: dataset to bicluster. The number of times to repeat the biclustering for each set of parameters. Default 10.
- **parallel** : Logical indicating if the user would like to utilize the `foreach` parallel backend. Default is FALSE.
- **ncores**: The number of cores to use if parallel computing. Default 2.
- **tune_grid**: A data frame of parameters to tune over. The column names of this must match the arguments passed to `biclustermd()`.

```
> flights_grid <- expand.grid(
+   row_clusters = 4,
+   col_clusters = c(6, 9, 12),
+   miss_val = fivenum(flights),
+   similarity = c("Rand", "Jaccard")
+ )

> flights_tune <- tune_biclustermd(
+   flights,
+   nrep = 10,
+   parallel = TRUE,
+   tune_grid = flights_grid
+ )
```

The output of `tune_biclustermd()` is a list of class "biclustermd" and "list" containing the following:

- **best_combn**: The best combination of parameters
- **best_bc**: The minimum SSE biclustering using the parameters in `best_combn`
- **grid**: `tune_grid` with columns giving the minimum, mean, and standard deviation of the final SSE for each parameter combination
- **runtime**: CPU runtime & elapsed time.

Users can easily identify which set of tuning parameters gives the best results and corresponding performance with the below code. The minimum SSE is obtained when 12 column clusters are used, the missing value used is -34, and the Rand similarity is used. A minimum SSE of 70,698 was obtained in the 10 repeats with that combination, which is a 16% reduction in SSE from our original parameter guesses above. Due to the unsupervised nature of biclustering, ultimately, it is the user's responsibility to choose reasonable number of row and column clusters for interpretations. Each domain and application of biclustering may lead to a different number of desired row or column clusters for a given array size. We simply utilize the SSE and convergence criteria as quantitative measures in determining the quality of the biclustering result.

```
 > flights_tune$grid[trimws(flights_tune$grid$best_combn) == '*',]
    row_clusters col_clusters miss_val similarity  min_sse mean_sse  sd_sse best_combn
  3            4           12      -34       Rand 70697.95 76581.85 4934.83 *
```

Any of the previously discussed exploratory functions can be used on the biclustering fit with the best tuning parameters by accessing the `best_bc` element of `flights_tune` since it is a biclustermd object:

```
> flights_tune$best_bc
Data has 1260 values, 11.75% of which are missing
8 Iterations
Initial SSE = 184165; Final SSE = 69586
Rand similarity used; Indices: Columns (P) = 1, Rows (Q) = 1
```

Finally, **biclustermd** also possesses a method for `gather()` (Wickham and Henry, 2019) which provides the name of the row and column a data point comes from as well as its corresponding row and column group association. This is particularly useful since we can easily determine the cell membership of each row and column to do further analysis. Namely, given these associations one can further analyze the quality of each cell and paired with domain knowledge of their data make informed judgments about the value of the biclustering. The following output was created from `flights_tune$best_bc`.

```
> gather(flights_tune$best_bc) %>% head()
    row_name col_name row_cluster col_cluster bicluster_no    value
  1  January      ABQ           1           1            1       NA
  2    March      ABQ           1           1            1       NA
  3    April      ABQ           1           1            1 12.22222
  4  January      ACK           1           1            1       NA
  5    March      ACK           1           1            1       NA
  6    April      ACK           1           1            1       NA
```

Example with soybean yield data

For our next example, we perform biclustering on a dataset which has a larger fraction of missing data to further show the practicability of our algorithm. Using data from a commercial soybean breeding program, we consider 132 soybean varieties as rows, 73 locations as columns, and yield in bushels per acre as the response. The locations span across the Midwestern United States and includes parts of Illinois, Iowa, Minnesota, Nebraska, and South Dakota, and each of the 132 soybean varieties represent a different genetic make-up. As one can imagine, not every soybean is grown in each location, as such we obtain a dataset with approximately 72.9% missing values. One application of a dataset such as this would be to determine if there are some subset of soybeans that perform consistently better (or worse) in some locations than others. From a plant breeding perspective, it is of vital importance to understand the relationship between the genetics and environments of crops, and identifying cells non-overlapping homogeneous cells from biclustering can provide insights into this matter (Malosetti et al., 2013).

The main purpose of this dataset is to demonstrate our algorithm on a dataset with a large amount of missing values as well as show the usefulness of the tuning parameters. Below is our first trial on the soybean yield data where we partition into 10 column clusters, 11 row clusters, and use the Jaccard similarity measure.

```
> yield_bc <- biclustermd(
+   yield,
+   col_clusters = 10,
+   row_clusters = 11,
+   similarity = "Jaccard",
+   miss_val_sd = sd(yield, na.rm = TRUE),
+   col_min_num = 3,
+   row_min_num = 3
+ )
> yield_bc

Data has 9636 values, 72.9% of which are missing
13 Iterations
Initial SSE = 239166; Final SSE = 51813, a 78.3% reduction
Jaccard similarity used; Indices: Columns (P) = 1, Rows (Q) = 1
```

In observing Figure 10, we notice that perfect convergence through the Rand Index, adjusted Rand Index, and Jaccard similarity; however, the similarities suggest that the columns converge more quickly than the rows. This may be attributed to the high percentage of missing values in the rows of the data table. That is, for each location there is more data available than there is for each soybean variety. Again we notice decreases in the values for each of the three indices, but observing Figure

11, we are assured that the algorithm is only making a column/row swap because a lower SSE is obtainable.

```
> autoplot(yield_bc$Similarities, facet = TRUE, ncol = 3, size = 0) +
+   theme_bw() +
+   theme(aspect.ratio =  1)
```

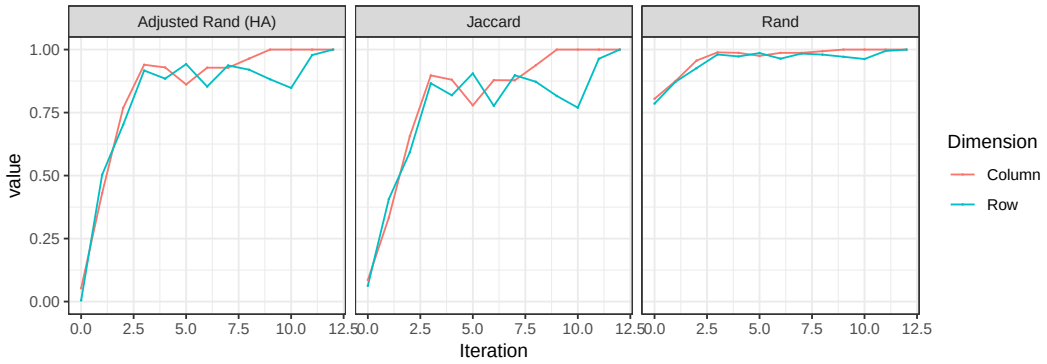


Figure 10: Plot of similarity measures for the soybean yield biclustering

```
> autoplot(yield_bc$SSE, size = 1) +
+   theme_bw() +
+   theme(aspect.ratio =  1) +
+   scale_y_continuous(labels = comma)
```

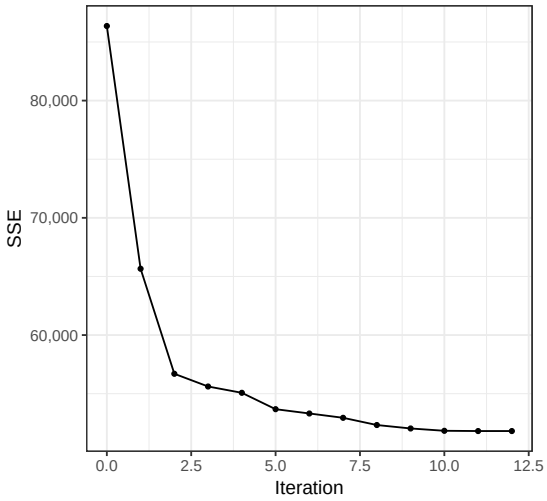


Figure 11: SSE plot of soybean yield biclustering

For the initial trial we observe that the Jaccard index converges in 13 iterations to an SSE value of 51,813. To see if it is possible to decrease this SSE even further, we test the impact of `col_shuffles` and `row_shuffles`. Recall that these parameters determine how many row and column rearrangements the algorithm makes before completing *one* iteration. Below we use `tune_biclustermd()` to test combinations of `col_shuffles` and `row_shuffles` as well as its corresponding SSE. We define the tune grid to mimic that of the `yield_bc` creation above, but let `col_shuffles` and `row_shuffles` take on values in $\{1, 3, 6\}$ independent of each other. We repeat the biclustering ten times for each parameter, specified by `nrep = 10`. Note that `parallel = TRUE` allows us to tune over the grid in parallel.

```
> yield_tbc <- tune_biclustermd(
+   yield,
+   nrep = 10,
+   tune_grid = expand.grid(
+     col_clusters = 10,
```

```
+     row_clusters = 11,
+     similarity = "Jaccard",
+     miss_val_sd = sd(yield, na.rm = TRUE),
+     col_min_num = 3,
+     row_min_num = 3,
+     row_shuffles = c(1, 3, 6),
+     col_shuffles = c(1, 3, 6)
+   ),
+   parallel = TRUE,
+   ncores = 2
+ )
> yield_tbc$grid[, c('row_shuffles', 'col_shuffles', 'min_sse', 'sd_sse', 'best_combn')]
    row_shuffles col_shuffles  min_sse   sd_sse best_combn
  1            1            1 51202.74 2640.662
  2            3            1 54073.92 2766.218
  3            6            1 52203.23 3198.391
  4            1            3 51296.99 1883.676
  5            3            3 52869.85 2118.745
  6            6            3 50530.38 2107.578 *
  7            1            6 51442.19 1895.268
  8            3            6 52111.31 2015.416
  9            6            6 52870.18 2652.400
```

Algorithm time study with movie ratings data

For our last example, we focus our attention on a movie ratings dataset obtained from MovieLens (Harper and Konstan, 2015). If we consider movie raters as defining rows, movies as defining columns, and a rating from 1–5 (with 5 being the most favorable) as a response, then biclustering can be used to determine subsets of raters who have similar preferences towards some subset of movies.

The main topic of this section will be to perform time studies to test the scalability of our proposed algorithm. In some applications, it is not uncommon to have a two-way data table with 10,000+ rows or columns. Intuitively as the dimensions of the two-way data table increases so will the computational time. In it is not uncommon for other biclustering algorithms to run for 24+ hours (Oghabian et al., 2014). We ran the biclustering over a grid of 80 combinations of I rows, J columns, r row clusters, and c column clusters with 30 replications for each combination. In addition to the four grid parameters, we consider the following metrics which are byproducts of the four parameters: the size of the dataset $N = I \times J$, average row cluster size I/r, and average column cluster size J/c. Table 1 summarizes the grid parameters, their byproducts and the defined lower and upper limits on each.

	$N = I \times J$	I	J	r	c	I/r	J/c
Lower Limit	2,500	50	50	4	4	5	5
Min	18,146	86	98	4	4	5	5
Mean	665,842	784	839	42	45	49	47
Max	1,929,708	1,495	1,457	239	258	293	346
Upper Limit	2,225,000	1,500	1,500	300	300	375	375

Table 1: Summary of the movie data runtime grid with defined lower and upper limits

Table 2 gives a five number summary and the mean runtime in seconds paired with the parameters which produced run times closest to each statistic. In all, we see that the algorithm can take less than a second to run, while in the other extreme the algorithm requires 39 minutes to converge. It is particularly interesting that for the two parameter combinations closest to the median run time, one dataset is nearly twice the size of the other. Furthermore, note than the mean run time is more than twice that of the median, but the size of the dataset is just 38% of that at the median. However, at the mean, $3744 = 72 \cdot 52$ biclusters are computed, while at the medians, only $80 = 20 \cdot 4$ and $481 = 13 \cdot 37$ biclusters are computed. For a visual summary of the results, we point the reader to Figure 12.

Figure 12 plots run times versus the five parameters controlled for in the study as well as average row cluster size, average column cluster size, and sparsity. We encourage the reader to personally explore the results; the run time data is the `runtimes` dataset in the package. Moreover, Li et al. (2020), provides further insights into the effect of sparsity on runtimes.

Finally, we address the trade-off between interpretability and computation time. Figure 13 plots

	Seconds	I	J	N	r	c	Sparsity
Min	0.4	210	98	20,580	10	9	96.2%
Q1	24.7	820	184	150,880	53	12	98.2%
Median	63.6	988	1,240	1,225,120	20	4	98.5%
Median	63.5	501	1,302	652,302	13	37	98.0%
Mean	137.5	1,084	427	462,868	72	52	98.4%
Q3	141.0	485	875	424,375	36	126	98.0%
Max	2,369.0	1,495	1,233	1,843,335	147	204	98.5%

Table 2: Five number summary and mean runtime in seconds along with parameters achieved at

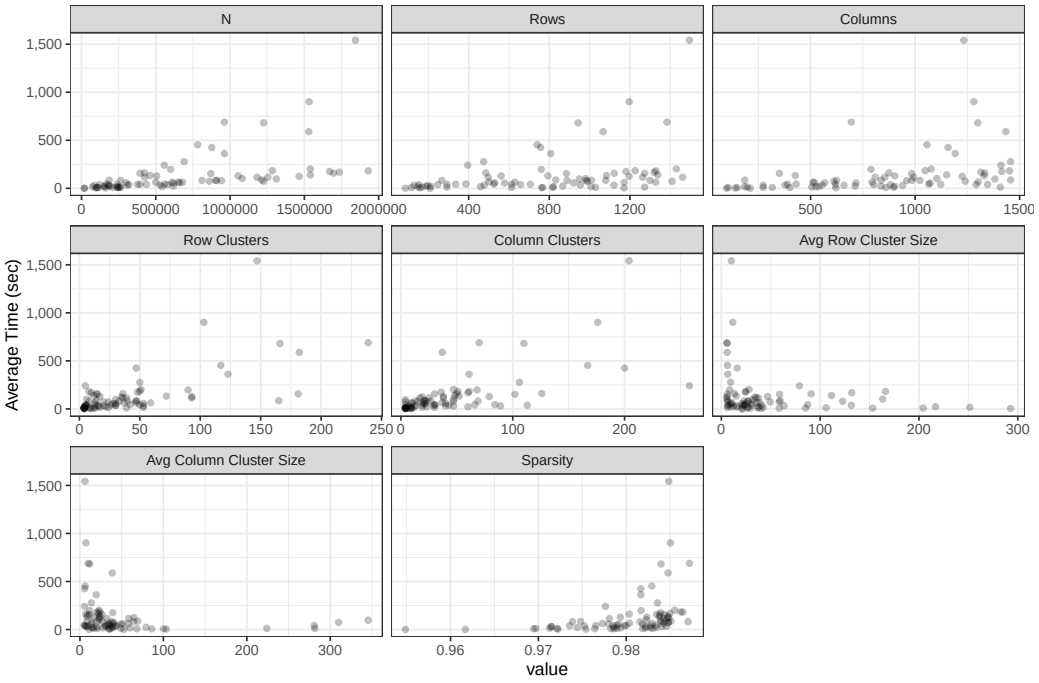


Figure 12: Relationship between movie grid parameters and elapsed time

elapsed time versus average cluster size on a doubly log 10 scales for row clusters (left) and column clusters (right). Clearly, computation time can be decreased by increasing the average cluster size, but doing so potentially reduces the interpretability of results; biclusters may be too large for certain use cases. Keeping in mind that the y-axis is on a log 10 scale, increasing average cluster size will have diminishing returns. Reviewing the plot on the right-hand side of the second row and the left-hand side of row three in Figure 12 sheds more light into this notion.

Summary

Based on the work of (Li et al., 2020) we provide a user-friendly R implementation of their proposed biclustering algorithm for missing data as well as a variety of visual aids that are helpful for biclustering in general and biclustering with missing data specifically. The unique benefit **biclustermd** provides is in its ability to operate with missing values. Compared to other packages which do not allow incomplete data or make use of some sort of imputation, we approach this problem with a novel framework that does not alter the structure of an inputted data matrix. Moreover, given the tunability of our biclustering algorithm, users are able to run trials on numerous combinations in an attempt to best bicluster their data.

Acknowledgments

This research was supported in part by Syngenta Seeds and by a Kingland Data Analytics Faculty Fellowship at Iowa State University.

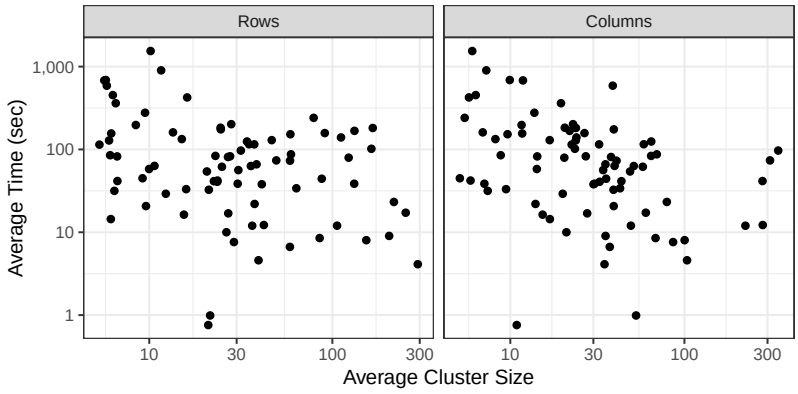


Figure 13: Relationship between average cluster sizes and elapsed time

Bibliography

S. Bergmann, J. Ihmels, and N. Barkai. Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, 2003. ISSN 1063651X. URL https://doi.org/10.1103/PhysRevE.67.031902. [p69]

S. Busygin, O. Prokopyev, and P. M. Pardalos. Biclustering in data mining. *Computers and Operations Research*, 35(9), 2008. URL https://doi.org/10.1016/j.cor.2007.01.005. [p69]

Y. Cheng and G. M. Church. Biclustering of expression data. *Proceedings International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology*, 8, 2000. [p69]

D. T. Ewoud. *BiBitR: R Wrapper for Java Implementation of BiBit*, 2017. R package version 0.4.2. [p69]

D. W. Goodall. A new similarity index based on probability. *Biometrics*, 22:882–907, 1966. [p71]

D. Gusenleitner and A. Culhane. *iBBiG: Iterative Binary Biclustering of Genesets*, 2019. URL http://bcb.dfci.harvard.edu/~aedin/publications/. R package version 1.28.0. [p69]

F. M. Harper and J. A. Konstan. The MovieLens datasets. *ACM Transactions on Interactive Intelligent Systems*, 2015. [p81]

J. A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337): 123–129, 1972. URL https://doi.org/10.1080/01621459.1972.10481214. [p69]

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 1985. ISSN 01764268. doi: 10.1007/BF01908075. [p71]

S. Kaiser and F. Leisch. A toolbox for bicluster analysis in {R}. *Compstat 2008—Proceedings in Computational Statistics*, pages 201–208, 2008. [p69]

T. Khamiakova. *superbiclust: Generating Robust Biclusters from a Bicluster Set (Ensemble Biclustering)*, 2014. URL https://CRAN.R-project.org/package=superbiclust. R package version 1.1. [p69]

Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein. Spectral biclustering of microarray data: Coclustering genes and conditions. *Genome Research*, 13(4):703–716, 2003. URL https://doi.org/10.1101/gr.648603. [p69]

L. Lazzeroni and A. Owen. Plaid models for gene expression data. *Statistica Sinica*, 12:61–86, 2002. ISSN 10170405. URL https://doi.org/10.1017/CBO9781107415324.004. [p69]

J. Li, J. Reisner, H. Pham, S. Olafsson, and S. Vardeman. Biclustering for missing data. *Information Sciences*, 510:304–316, 2020. URL https://doi.org/10.1016/j.ins.2019.09.047. [p70, 72, 81, 82]

M. Malosetti, J. M. Ribaut, and F. A. van Eeuwijk. The statistical analysis of multi-environment data: Modeling genotype-by-environment interaction and its genetic basis. *Frontiers in Physiology*, 4(44), 2013. URL https://doi.org/10.3389/fphys.2013.00044. [p79]

A. Oghabian, S. Kilpinen, S. Hautaniemi, and E. Czeizler. Biclustering methods: Biological relevance and application in gene expression analysis. *PLOS ONE*, 9(3):1–10, 03 2014. doi: 10.1371/journal.pone.0090801. URL https://doi.org/10.1371/journal.pone.0090801. [p81]

M. Rand. Objective criteria for the evaluation of methods clustering. *Journal of the American Statistical Association*, 66(336):846–850, 1971. URL https://doi.org/10.1080/01621459.1971.10482356. [p71]

M. Sill and S. Kaiser. *s4vd: Biclustering via Sparse Singular Value Decomposition Incorporating Stability Selection*, 2015. URL https://CRAN.R-project.org/package=s4vd. R package version 1.1-1. [p69]

K. M. Tan and D. M. Witten. Sparse Biclustering of Transposable Data. *Journal of Computational and Graphical Statistics*, 2014. ISSN 15372715. URL https://doi.org/10.1080/10618600.2013.852554. [p69]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN 978-0-387-98140-6. URL http://ggplot2.org. [p73]

H. Wickham. *tidyverse: Easily Install and Load 'Tidyverse' Packages*. 2016. URL https://cran.r-project.org/package=tidyverse. [p72]

H. Wickham. *nycflights13: Flights that departed NYC in 2013*, 2017. URL https://CRAN.R-project.org/package=nycflights13. R package version 0.2.2. [p72]

H. Wickham and L. Henry. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2019. URL https://CRAN.R-project.org/package=tidyr. R package version 0.8.3. [p79]

Y. Zhang, J. Xie, J. Yang, A. Fennell, C. Zhang, and Q. Ma. QUBIC: a bioconductor package for qualitative biclustering analysis of gene co-expression data. *Bioinformatics*, 33(3):450–452, 2017. URL https://doi.org/10.1093/bioinformatics/btw635. [p69]

John Reisner
Department of Statistics
Iowa State University
United States
johntreisner@gmail.com

Hieu Pham
Department of Industrial and Manufacturing Systems Engineering
Iowa State University
United States
htpham@iastate.edu

Sigurdur Olafsson
Department of Industrial and Manufacturing Systems Engineering
Iowa State University
United States
olafsson@iastate.edu

Stephen Vardeman
Department of Statistics
Department of Industrial and Manufacturing Systems Engineering
Iowa State University
United States
vardeman@iastate.edu

Jing Li
Boehringer Ingelheim Animal Health
St. Joseph, Missouri
United States
jingli2014cymail@gmail.com

auditor: an R Package for Model-Agnostic Visual Validation and Diagnostics

by Alicja Gosiewska and Przemysław Biecek

Abstract

Machine learning models have successfully been applied to challenges in applied in biology, medicine, finance, physics, and other fields. With modern software it is easy to train even a complex model that fits the training data and results in high accuracy on test set. However, problems often arise when models are confronted with the real-world data. This paper describes methodology and tools for model-agnostic auditing. It provides functinos for assessing and comparing the goodness of fit and performance of models. In addition, the package may be used for analysis of the similarity of residuals and for identification of outliers and influential observations. The examination is carried out by diagnostic scores and visual verification. The code presented in this paper are implemented in the **auditor** package. Its flexible and consistent grammar facilitates the validation models of a large class of models.

Introduction

Predictive modeling is a process using mathematical and computational methods to forecast outcomes. Many algorithms in this area have been developed and novel ones are continuously being proposed. Therefore, there are countless possible models to choose from and a lot of ways to train a new new complex model. A poorly- or over-fitted model usually will be of no use when confronted with future data. Its predictions will be misleading (Sheather, 2009) or harmful (O'Neil, 2016). That is why methods that support model diagnostics are important.

Diagnostics are often carried out only by checking model assumptions. However, they are often neglected for complex machine learning models and they may be used as if they were assumption-free. Still, there is a need to verify their quality. We strongly believe that a genuine diagnosis or an audit incorporates a broad approach to model exploration. The audit includes three objectives.

- **Objective 1:** Enrichment of information about model performance.
- **Objective 2:** Identification of outliers, influential and abnormal observations.
- **Objective 3:** Examination of other problems relating to a model by analyzing distributions of residuals, in particular, problems with bias, heteroscedasticity of variance and autocorrelation of residuals.

In this paper, we introduce the **auditor** package for R, which is a tool for diagnostics and visual verification. As it focuses on residuals[1] and does not require any additional model assumptions, most of the presented methods are model-agnostic. A consistent grammar across various tools reduces the amount of effort needed to create informative plots and makes the validation more convenient and available.

Diagnostic methods have been a subject of much research (Atkinson, 1985). Atkinson and Riani (2012) focus on graphical methods of diagnostics regression analysis. Liu et al. (2017) present an overview of interactive visual model validation. One of the most popular tools for verification are measures of the differences between the values predicted by a model and the observed values (Willmott et al., 1985). These tools include Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) (Hastie et al., 2001). Such measures are used for well-researched and easily interpretable linear model as well as for complex models such as random forests (Ho, 1995), gradient-boosted trees (Chen and Guestrin, 2016), or neural networks (Venables and Ripley, 2002).

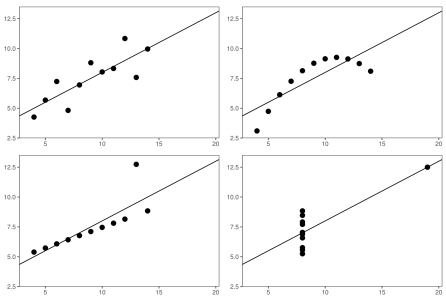


Figure 1: Anscombe Quartet data sets are identical when examined with he use of simple summary statistics. The difference is noticeable after plotting the data.

However, no matter which measure of model performance we use, it does not reflect all aspects of the model. For example, Breiman (2001) points out that a linear regression model validated only on the basis of R^2

[1]Residual of an observation is the difference between the observed value and the value predicted by a model.

may lead to many false conclusions. The best known example of this issue is the Anscombe Quartet (Anscombe, 1973). It contains four different data sets constructed to have nearly identical simple statistical properties such as mean, variance, correlation, etc. These measures directly correspond to the coefficients of the linear models. Therefore, by fitting a linear regression to the Anscombe Quartet we obtain four almost identical models (see Figure 1). However, residuals of these models are very different. The Anscombe Quartet is used to highlight that the numerical measures should be supplemented by graphical data visualizations.

The analysis of diagnostics is well-researched for linear and generalized linear models. The said analysis is typically done by extracting raw, studentized, deviance, or Pearson residuals and examining residual plots. Common problems with model fit and basic diagnostics methods are presented in Faraway (2002) and Harrell Jr. (2006)

Model validation may involve both checking the overall trend in residuals and looking at residual values of individual observations (Littell et al., 2007). Gałecki and Burzykowski (2013) discussed methods based on residuals for individual observation and groups of observations.

Diagnostics methods are commonly used for linear regression (Faraway, 2004). Complex models are treated as if they were assumption-free, which is why their diagnostics is often ignored. Considering the above, there is a need for more extensive methods and software dedicated for model auditing. Many of diagnostic tools, such as plots and statistics developed for linear models, are still useful for exploring machine learning models. Applying the same tools to all models facilitates their comparison.

The paper is organized as follows. Section 16.2 summarizes related work and state of the art. Section 16.3 contains an architecture of the **auditor** package. Section 16.4 provides the notation. Selected tools that help to validate models are presented in Section 16.5 and conclusions can be found in Section 16.6.

Related work

In this chapter, we overview common methods and tools for auditing and examining the validity of the models. There are several attempts to validate. They include diagnostics for predictor variables before and after model fit, methods dedicated to specific models, and model-agnostic approaches.

Data diagnostics before model fitting

The problem of data diagnostics is related to the Objective 2 presented in the Introduction, that is, the identification of problems with observations. There are several tools that address this issue. We review the most popular of them.

- One of the tools that supports the identification of errors in data is the **dataMaid** package (Petersen and Ekstrom, 2018). It creates a report that contains summaries and error checks for each variable in data. Package **lumberjack** (van der Loo, 2017) provides row-wise analysis. It allows for monitoring changes in data as they get processed. The **validatetools** (de Jonge and van der Loo, 2018) is a package for managing validation rules.
- The `datadist` function from **rms** package (Harrell Jr, 2018) computes distributional summaries for predictor variables. They include the overall range and certain quantiles for continuous variables, as well as distinct values for discrete variables. It automates the process of fitting and validating several models due to storing model summaries by `datadist` function.
- While above packages use pipeline approaches, there are also tools that focus on specific step of data diagnostic. The package **corrgram** (Wright, 2018) calculates a correlation of variables and displays corrgrams. Corrgrams (Friendly, 2002) are visualizations of correlation matrices, that help to identify the relationship between variables.

Diagnostics methods for linear models

As linear models have a very simple structure and do not require high computational power, they have been and still are used very frequently. Therefore, there are many tools that validate different aspects of linear models. Below, we overview the most widely known tools implemented in R packages.

- The **stats** package provides basic diagnostic plots for linear models. Function `plot` generates six types of charts for `"lm"` and `"glm"` objects, such as a plot of residuals against fitted values, a scale-location plot of $\sqrt{|residuals|}$ against fitted values and a normal quantile-quantile plot. These visual validation tools may be addressed to the Objective 3 of diagnostic, related to the

examination of model by analyzing the distribution of residuals. The other three plots, that include: a plot of Cook's distances, a plot of residuals against leverages, and a plot of Cook's distances against $\frac{leverage}{1-leverage}$ may be addressed to the identification of influential observations (Objective 1).

- Package **car** (Fox and Weisberg, 2011) extends the capabilities of **stats** by including more types of residuals, such as Pearson and deviance residuals. It is possible to plot against values of selected variables and to group residuals by levels of factor variables. What is more, **car** provides more diagnostic plots such as, among others, partial residual plot (`crPlot`), index plots of influence (`infIndexPlot`) and bubble plot of studentized residuals versus hat values (`influencePlot`). These plots allow for checking both the effect of observation and the distribution of residuals, what address to the Objective 2 and the Objective 3 respectively.
- A linear regression model is still one of the most popular tools for data analysis due to its simple structure. Therefore, there is a rich variety of methods for checking its assumptions, for example, the normality of residual distribution and the homoscedasticity of the variance.

  The package **nortest** (Gross and Ligges, 2015) provides five tests for normality: the Anderson-Darling (Anderson and Darling, 1952), the Cramer-von Mises (Cramer, 1928; Von Mises, 1928), the Kolmogorov-Smirnov (Stephens, 1974), the Pearson chi-square (F.R.S., 1900), and the Shapiro-Francia (Sanford Shapiro and S. Francia, 1972) tests. The **lmtest** package (Zeileis and Hothorn, 2002) also contains a collection of diagnostic tests: the Breusch-Pagan (Breusch and Pagan, 1979), the Goldfield-Quandt (Goldfeld and Quandt, 1965) and the Harrison-McCabe (Harrison and McCabe, 1979) tests for heteroscedasticity and the Harvey-Collier (Harvey and Collier, 1977), the Rainbow (Utts, 1982), and the RESET (Ramsey, 1969) tests for nonlinearity and misspecified functional form. A unified approach for examining, monitoring and dating structural changes in linear regression models is provided in **strucchange** package (Zeileis et al., 2002). It includes methods to fit, plot and test fluctuation processes and F-statistics. The **gvlma** implements the global procedure for testing the assumptions of the linear model (find more details in Peña and Slate (2006)).

  The Box-Cox power transformation introduced by Box and Cox (1964) is a way to transform the data to follow a normal distribution. For simple linear regression, it is often used to satisfy the assumptions of the model. Package **MASS** (Venables and Ripley, 2002) contains functions that compute and plot profile log-likelihoods for the parameter of the Box-Cox power transformation.
- The **broom** package (Robinson, 2018) provides summaries for about 30 classes of models. It produces results, such as coefficients and p-values for each variable, R^2, adjusted R^2, and residual standard error.

Other model-specific approaches

There are also several tools to generate validation plots for time series, principal component analysis, clustering, and others.

- Tang et al. (2016) introduced the **ggfortify** interface for visualizing many popular statistical results. Plots are generated with **ggplot2** (Wickham, 2009), what makes them easy to modify. With one function `autoplot` it is possible to generate validation plots for a wide range of models. It works for, among others, `lm`, `glm`, `ts`, `glmnet`, and `survfit` objects.

  The **autoplotly** (Tang, 2018) package is an extension of **ggfortify** and it provides functionalities that produce plots generated by **plotly** (Sievert et al., 2017). This allows for both modification and interaction with plots.

  However, **ggorftify** and **autoplotly** do not support some popular types of models, for instance, random forests from **randomForest** (Liaw and Wiener, 2002) and **ranger** (Wright and Ziegler, 2017) packages.
- The **hnp** package (Moral et al., 2017) provides half-normal plots with simulated envelopes. These charts evaluate the goodness of fit of any generalized linear model and its extensions. It is a graphical method for comparing two probability distributions by plotting their quantiles against each other. The package offers a possibility to extend the `hnp` for new model classes. However, this package provides only one tool for model diagnostic. In addition, plots are not based on **ggplot2**, what makes it difficult to modify them.

Model-agnostic approach

The tools presented above target specific model classes. The model-agnostic approach allows us to compare different models.

- The **DALEX** (Descriptive mAchine Learning EXplanations) (Biecek, 2018) is a methodology for exploration of black-box models. Main functionalities focus on understanding or proving how the input variables impact on final predictions. There are also two simple diagnostics: reversed empirical cumulative distribution function for absolute values of residuals and box plot of absolute values of residuals. As methods in the **DALEX** are model-agnostic, they allow for comparison of two or more models.
- The package **iml** (Molnar et al., 2018) also contains methods for structure-agnostic exploration of model. For example, a measure of a feature's importance by calculating the change of the model performance after permuting values of a variable.

Model-agnostic audit

In this paper, we present the **auditor** package for R, which fills out the part of model-agnostic validation. As it expands methods used for linear regression, it may be used to verify any predictive model.

Package Architecture

The **auditor** package works for any predictive model which returns a numeric value. It offers a consistent grammar of model validation, what is an efficient and convenient way to generate plots and diagnostic scores. A diagnostic score is a number that evaluates one of the properties of a model. That might be, for example, an accuracy of model, an independence of residuals or an influence of observation.

Figure 2 presents the architecture of the package. The **auditor** provides 2 pipelines for model validation. First of them consists of two steps. Function `audit` wraps up the model with meta-data, then the result is passed to the `plot` or `score` function. Second pipeline includes an additional step, which consists of calling one of the functions that generate computations for plots and scores. These functions are: `modelResiduals`, `modelEvaluation`, `modelFit`, `modelPerformance`, and `observationInfluence`. Further, we call them computational functions. Results of these functions are tidy data frames (Wickham, 2014).

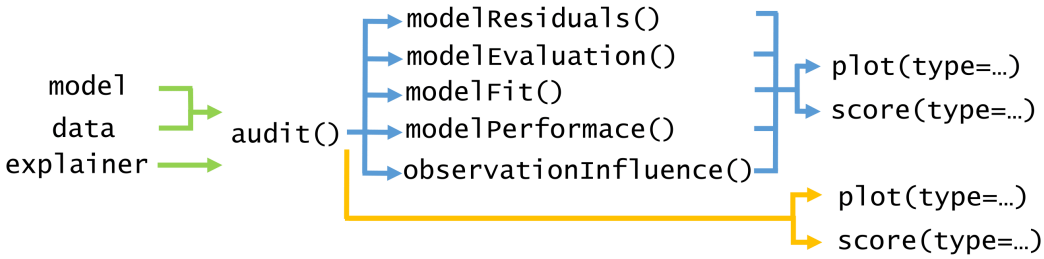


Figure 2: Architecture of the **auditor**. The blue color indicates the first pipeline, while orange indicates the second. Function `audit` takes model and data or `"explainer"` object created with the **DALEX** package.

Both pipelines for model audit are compared below.

1. **model %>% audit() %>% computational function %>% plot(type=...)**
   We recommend this pipeline. Function `audit` wraps up a model with meta-data used for modeling and creates a `"modelAudit"` object. One of the computational functions takes `"modelAudit"` object and computes the results of validation. Then, outputs may be printed or passed to functions `score` and `plot` with defined type. We describe types of plots in Chapter 16.5. This approach requires one additional function within the pipeline. However, once created output of the computational function contains all necessary calculations for related plots. Therefore, generating multiple plots is fast.
2. **model %>% audit() %>% plot(type=...)**
   This pipeline is shorter than the previous one. The only difference is that it does not include computational function. Calculations are carried out every time a generic `plot` function is called. Omitting one step might be convenient for ad-hoc model analyses.

Implemented types of plots are presented in Table 1. Scores are presented in Table 2. All plots are generated with **ggplot2**, what provides a convenient way to modify and combine plots.

Plot	Function	plot(type = ...)	Reg.	Class.
Autocorrelation Function	`modelResiduals`	`"ACF"`	+	+
Autocorrelation	`modelResiduals`	`"Autocorrelation"`	+	+
Cooks's Distances	`observationInfluence`	`"CooksDistance"`	+	+
Half-Normal	`modelFit`	`"HalfNormal"`	+	+
LIFT Chart	`modelEvaluation`	`"LIFT"`		+
Model Correlation	`modelResiduals`	`"ModelCorrelation"`	+	+
Model PCA	`modelResiduals`	`"ModelPCA"`	+	+
Model Ranking	`modelPerformance`	`"ModelRanking"`	+	+
Predicted Response	`modelPerformance`	`"ModelPerformance"`	+	+
REC Curve	`modelResiduals`	`"REC"`	+	+
Residuals	`modelResiduals`	`"Residual"`	+	+
Residual Boxplot	`modelResiduals`	`"ResidualBoxplot"`	+	+
Residual Density	`modelResiduals`	`"ResidualDensity"`	+	+
ROC Curve	`modelEvaluation`	`"ROC"`		+
RROC Curve	`modelResiduals`	`"RROC"`	+	+
Scale-Location	`modelResiduals`	`"ScaleLocation"`	+	+
Two-sided ECDF	`modelResiduals`	`"TwoSidedECDF"`	+	+

Table 1: Columns contain respectively: name of the plot, name of the computational function, value for `type` parameter of the function `plot`, indications whether the plot can be applied to regression and classification tasks.

Score	Function	score(type = ...)	Reg.	Class.
Cook's Distance	`observationInfluence`	`"CooksDistance"`	+	+
Durbin-Watson	`modelResiduals`	`"DW"`	+	+
Half-Normal	`modelFit`	`"HalfNormal"`	+	+
Mean Absolute Error	`modelResiduals`	`"MAE"`	+	+
Mean Squared Error	`modelResiduals`	`"MSE"`	+	+
Area Over the REC	`modelResiduals`	`"REC"`	+	+
Root Mean Squared Error	`modelResiduals`	`"RMSE"`	+	+
Area Under the ROC	`modelEvaluation`	`"ROC"`		+
Area Over the RROC	`modelResiduals`	`"RROC"`	+	+
Runs	`modelResiduals`	`"Runs"`	+	+
Peak	`modelResiduals`	`"Peak"`	+	+

Table 2: Columns contain respectively: name of a score, name of a computational function, value for `type` parameter of function the `score`, indications whether the score can be applied to regression and classification tasks.

Notation

Let us use the following notation: $x_i = (x_i^{(1)}, x_i^{(2)}, ..., x_i^{(p)}) \in \mathcal{X} \subset \mathcal{R}^p$ is a vector in space $\mathcal{X}$, $y_i \in \mathcal{R}$ is an observed response associated with x_i. A single observation we denote as a pair (y_i, x_i) and n is the number of observations.

Let us denote a model as a function $f : \mathcal{X} \to \mathcal{R}$. Predictions of the model f for particular observation we shall denote as

$$f(x_i) = \hat{y}_i. \tag{1}$$

The row residual, or simply the residual, is the difference between the observed value y_i and the predicted value $\hat{y}_i$. We shall denote residual of particular observation as

$$r_i = y_i - \hat{y}_i. \tag{2}$$

Illustrations

Diagnostics allows for evaluation of different properties of a model. They may be related to the following questions: Which model has better performance? Does the model fit data? Which observations are abnormal? These questions are directly related to the diagnostics objectives described in the Introduction. First of them refers to the evaluation of a model performance, which was proposed as the Objective 1. The second question concerns the examination of residuals distribution (Objective 3). The last one refers to outliers and influential observations (Objective 2).

In this Section we illustrate chosen validation tools that allow for exploration of the above issues. To demonstrate applications of the **auditor**, we use the data set `apartments` available in the **DALEX** package. First, we fit two models: simple linear regression and random forest.

```
library("auditor")
library("DALEX")
library("randomForest")

lm_model <- lm(m2.price ~ ., data = apartments)
set.seed(59)
rf_model <- randomForest(m2.price ~ ., data = apartments)
```

The next step creates `"modelAudit"` objects related to these two models.

```
lm_audit <- audit(lm_model, label = "lm",
              data = apartmentsTest, y = apartmentsTest$m2.price)
rf_audit <- audit(rf_model, label = "rf",
              data = apartmentsTest, y = apartmentsTest$m2.price)
```

Below, we create objects of class `"modelResidual"`, which are needed to generate plots. Parameter `variable` determines the order of residuals in the plot. When the `variable` argument is set to `"Fitted values"` residuals are sorted by values of predicted responses. Entering a name of a variable `"m2.price"` implies that residuals will be in order of this variable.

```
lm_res_fitted <- modelResiduals(lm_audit, variable = "Fitted values")
rf_res_fitted <- modelResiduals(rf_audit, variable = "Fitted values")

lm_res_observed <- modelResiduals(lm_audit, variable = "m2.price")
rf_res_observed <- modelResiduals(rf_audit, variable = "m2.price")
```

Model Ranking Plot

In this subsection, we propose a Model Ranking plot which compares models performance across multiple measures (see Figure 3). The implemented measures are listed in Table 2 in Chapter 16.3. The descriptions of all scores are in (Gosiewska and Biecek, 2018).

Model Ranking Radar plot consists of two parts. On the left side there is a radar plot. Colors correspond to models, edges to values of scores. Score values are inverted and rescaled to $[0, 1]$.

Let us use the following notation: $m_i \in \mathcal{M}$ is a model in a finite set of models $\mathcal{M}$, where $|\mathcal{M}| = k$, $score : \mathcal{M} \to \mathbb{R}$ is a loss function for the model under consideration. Higher values mean worse model performance. The $score(m_i)$ is a performance of model m_i.

Definition 16.5.1. We define the inverted score of model m_i as

$$invscore(m_i) = \frac{1}{score(m_i)} \min_{j=1...k} score(m_j). \tag{3}$$

Models with the larger *invscore* are closer to the centre. Therefore, the best model is located the farthest from the center of the plot. On the right side of the plot is a table with results of scoring. The third column contains scores scaled to one of the models.

Let $m_l \in \mathcal{M}$ where $l \in \{1, 2, ..., k\}$ be a model to which we scale.

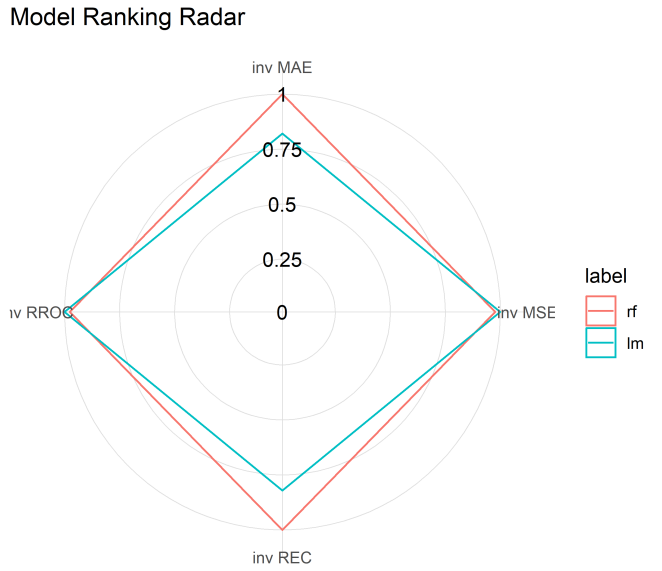
Definition 16.5.2. We define the scaled score of model m_i to model m_l as

$$scaled_l(m_i) = \frac{score(m_l)}{score(m_i)}. \tag{4}$$

As values of $scaled_l(m_l)$ are always between 0 and 1, comparison of models is easy, regardless of the ranges of scores.

The plot below is generated by `plot` function with parameter `type = "ModelRanking"` or by function `plotModelRanking`. The scores included in the plot may be specified by `scores` parameter.

```
rf_mp <- modelPerformance(rf_audit)
lm_mp <- modelPerformance(lm_audit)
plot(rf_mp, lm_mp, type = "ModelRanking")
```



name	label	values	scaled
MAE	rf	2.16e+02	1.000
MAE	lm	2.63e+02	0.821
MSE	rf	8.21e+04	1.000
MSE	lm	8.01e+04	1.025
REC	rf	2.16e+02	1.000
REC	lm	2.63e+02	0.821
RROC	rf	3.32e+12	1.000
RROC	lm	3.24e+12	1.024

Figure 3: Model Ranking Plot. Random forest (red) has better performance in aspect of MAE and REC scores, while linear model (blue) is better in aspect of MSE and RROC scores.

REC Curve Plot

Regression Error Characteristic (REC) curve (see Figure 4) is a generalization of Receiver Operating Characteristic (ROC) curve for binary classification (Swets, 1988).

REC curve estimates the Cumulative Distribution Function of the error. On the x axis of the plot there is an error tolerance. On the y axis there is an accuracy at the given tolerance level. Bi and Bennett (2003) define the accuracy at tolerance ϵ as a percentage of observations predicted within the tolerance ϵ. In other words, residuals larger than ϵ are considered as errors.

Let us consider pairs (y_i, x_i) introduced in the beginning of Chapter 16.5. Bi and Bennett (2003) define an accuracy as follows.

Definition 16.5.3. An accuracy at tolerance level ϵ is given by

$$acc(\epsilon) = \frac{|\{(x,y) : loss(f(x_i), y_i) \leq \epsilon, i = 1, ..., n\}|}{n}. \tag{5}$$

REC Curves implemented in the **auditor** are plotted for a special case of Definition 16.5.3 where the loss is defined as

$$loss(f(x_i), y_i) = |f(x_i) - y_i| = |r_i|. \tag{6}$$

The shape of the curve illustrates the behavior of errors. The quality of the model can be evaluated and compared for different tolerance levels. The stable growth of the accuracy does not indicate any problems with the model. A small increase of accuracy near 0 and the areas where the growth is fast signalize bias of the model predictions.

The plot below is generated by `plot` function with parameter `type = "REC"` or by `plotREC` function.

```
plot(rf_res_fitted, lm_res_fitted, type = "REC")
```

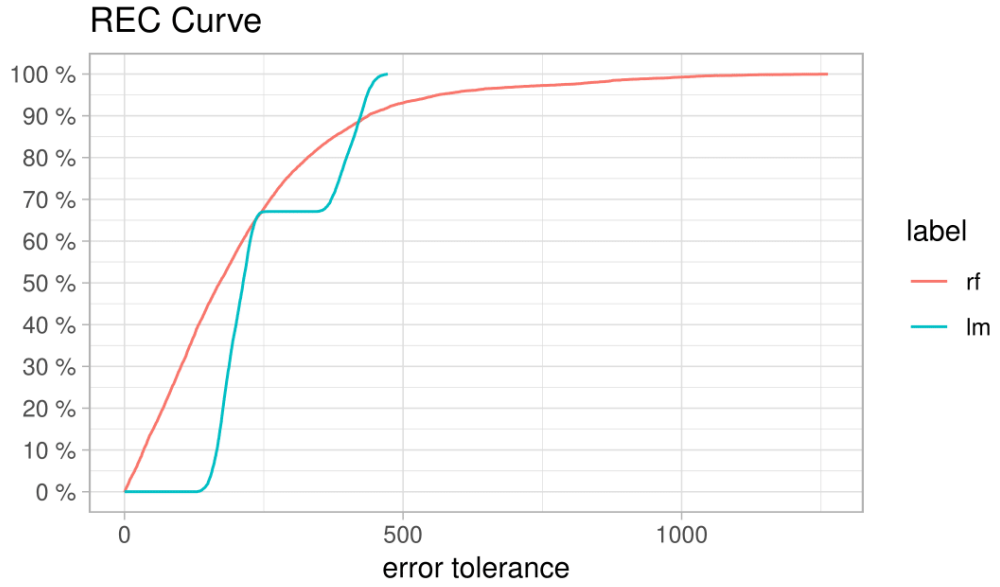


Figure 4: REC curve. Curve for linear model (blue) suggests that the model is biased. It displays poor accuracy when the tolerance ϵ is small. However, once ϵ exceeds the error tolerance 130, accuracy rapidly increases. The random forest (red) has a stable increase of accuracy when compared to the linear model. However, there is s fraction of large residuals.

As often it is difficult to compare models on the plot, there is an REC score implemented in the **auditor**. This score is the Area Over the REC Curve (AOC), which is a biased estimate of the expected error for a regression model. As Bi and Bennett (2003) proved, AOC provides a measure of the overall performance of regression model.

Scores may be obtained by `score` function with `type = "REC"` or `scoreREC` function.

```
scoreREC(lm_res_fitted)
scoreREC(rf_res_fitted)
```

Residual Boxplot Plot

Residual boxplot shows the distribution of the absolute values of residuals r_i. They may be used for analysis and comparison of residuals. Example plots are presented in Figure 5. Boxplots (Tukey, 1977) usually consist of five components. The box itself corresponds to the first quartile, median, and third quartile. The whiskers extend to the smallest and largest values, no further than 1.5 of Interquartile Range (IQR) from the first and third quartile respectively. Residual boxplots consists of a sixth component, namely a red dot which stands for Root Mean Square Error (RMSE). In case of an appropriate model, most of the residuals should lay near zero. A large spread of values indicates problems with a model.

The plot presented below is generated by `plotResidualBoxplot` or by `plot` function with parameter `type = 'ResidualBoxplot'` function.

```
plot(lm_res_fitted, rf_res_fitted, type = "ResidualBoxplot")
```

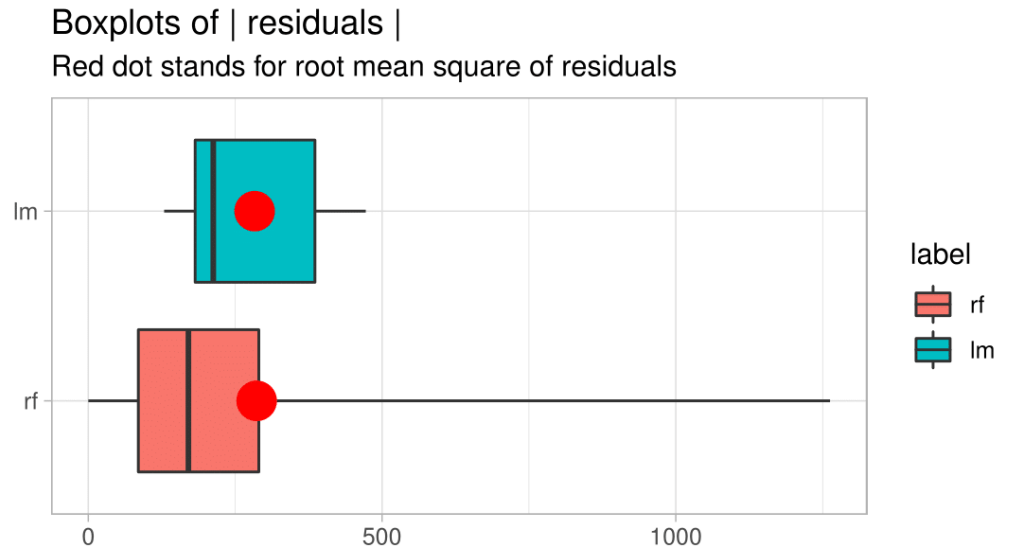


Figure 5: Boxplots of absolute values of residuals. Dots are in similar places, hence RMSE for both models is almost identical. However, the distribution of residuals of these two models is different. For the linear model (blue), most of the residuals are around the average. For the random forest (red), most residuals are small. Nevertheless, there is also a fraction of large residuals.

Residual Density Plot

Residual Density plot detects the incorrect behavior of residuals. An example is presented in Figure 6. On the plot, there are estimated densities of residuals. For some models, the expected shape of density derives from the model assumptions. For example, simple linear model residuals should be normally distributed. However, even if the model does not have an assumption about the distribution of residuals, such a plot may be informative. If most of the residuals are not concentrated around zero, it is likely that the model predictions are biased. Values of errors are displayed as marks along the x axis. That makes it possible to ascertain whether there are individual observations or groups of observations with residuals significantly larger than others.

The plot below is generated by `plotResidualDensity` function or by `plot` function with parameter `type = "ResidualDensity"`.

```
plot(rf_res_observed, lm_res_observed, type = "ResidualDensity")
```

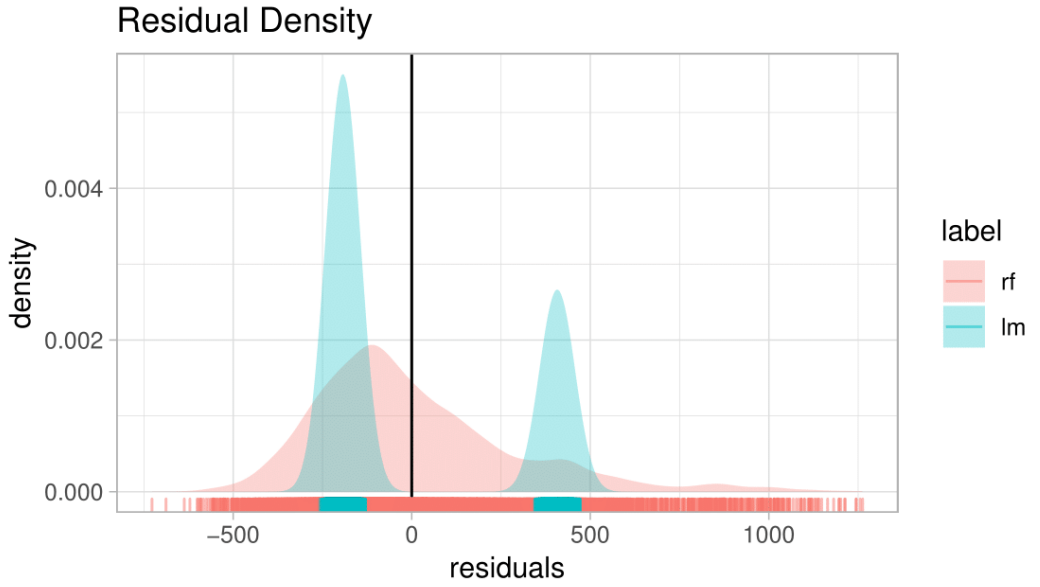


Figure 6: Residual Density Plot. The density of residuals for the linear model (blue) forms two peaks. There are no residuals with values around zero. Residuals do not follow the normal distribution, what is one of the assumptions of the simple linear regression. There is an asymmetry of residuals generated by random forest (red).

Two-sided ECDF Plot

Two-sided ECDF plot (see Figure 7) shows an Empirical Cumulative Distribution Functions (ECDF) for positive and negative values of residuals separately.

Let $x_1, ..., x_n$ be a random sample from a cumulative distribution function $F(t)$. The following definition comes from van der Vaart (2000).

Definition 16.5.4. The empirical cumulative distribution function is given by

$$F_n(t) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{x_i \leq t\}. \tag{7}$$

Empirical cumulative distribution function presents a fraction of observations that are less than or equal to t. It is an estimator for the cumulative distribution function $F(t)$.

On the positive side of the x-axis, there is the ECDF of positive values of residuals. On the negative side, there is a transformation of ECDF:

$$F_{rev}(t) = 1 - F(t). \tag{8}$$

Let n_N and n_P be numbers of negative and positive values of residuals respectively. Negative part of the plot is normalized by multiplying it by the ratio of the n_N over $n_N + n_P$. Similarly, positive part is normalized by multiplying it by the ratio of the n_P over $n_N + n_P$. Due to the applied scale, the ends of the curves add up to 100% in total. The plot shows the distribution of residuals divided into groups with positive and negative values. It helps to identify the asymmetry of the residuals. Points represent individual error values, what makes it possible to identify 'outliers'.

The plot below is generated by `plotTwoSidedECDF` function or by `plot` function with parameter `type = "TwoSidedECDF"`.

```
plot(rf_res_fitted, lm_res_fitted, type = "TwoSidedECDF")
```

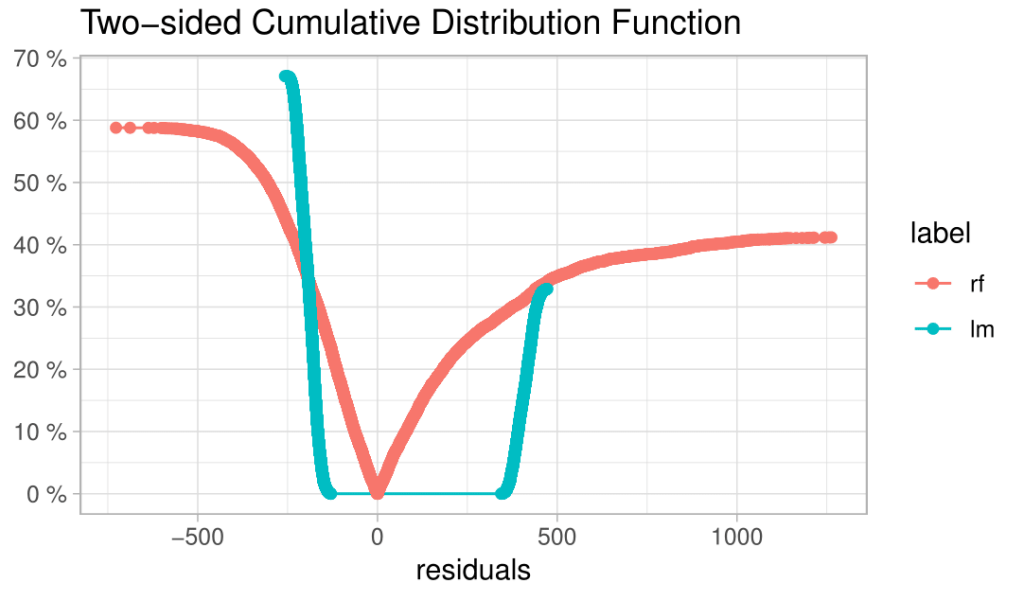


Figure 7: Two-sided ECDF plot. The plot shows that majority of residuals for the random forest (red) is smaller than residuals for the linear model (blue). However, random forest has also fractions of large residuals.

Conclusion and future work

In this article, we presented the **auditor** package and selected diagnostic scores and plots. We discussed the existing methods of model validation and proposed new visual approaches. We also specified three objectives of model audit (see Section 16.1), proposed relevant verification tools, and demonstrated their usage. Model Ranking Plot and REC Curve enrich the information about model performance (Objective 1). Residual Boxplot, Residual Density, and Two-Sided ECDF Plots expand the knowledge about the distribution of residuals (Objective 3). What is more, the latter two tools allow for identification of outliers (Objective 2). Finally, we proposed two new plots, the Model Ranking Plot and the Two-Sided ECDF Plot.

We implemented all the presented scores and plots in the **auditor** package for R. The included functions are based on a uniform grammar introduced in Figure 16.3. Documentation and examples are available at `https://mi2datalab.github.io/auditor/`. The stable version of the package is on CRAN, the development version is on GitHub (`https://github.com/MI2DataLab/auditor`). A more detailed description of methodology is available in the extended version of this paper on arXiv: `https://arxiv.org/abs/1809.07763` (Gosiewska and Biecek, 2018).

There are many potential areas for future work that we would like to explore, including more extensions of model-specific diagnostics to model-agnostic methods and residual-based methods for investigating interactions. Another potential aim would be to develop methods for local audit based on the diagnostics of a model around a single observation or a group of observations.

Acknowledgements

We would like to acknowledge and thank Aleksandra Grudziąż and Mateusz Staniak for valuable discussions. Also, we wish to thank Dr. Rafael De Andrade Moral for his assistance and help related to the **hnp** package.

The work was supported by NCN Opus grant 2016/21/B/ST6/02176.

Bibliography

T. W. Anderson and D. A. Darling. Asymptotic theory of certain goodness of fit criteria based on stochastic processes. *Ann. Math. Statist.*, 23(2):193–212, 1952. URL `https://doi.org/10.1214/aoms/1177729437`. [p]

F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, 1973. URL https://doi.org/10.1080/00031305.1973.10478966. [p]

A. Atkinson and M. Riani. *Robust Diagnostic Regression Analysis*. Springer Series in Statistics. Springer-Verlag, 2012. ISBN 9781461211600. URL https://books.google.pl/books?id=sZ3SBwAAQBAJ. [p]

A. C. Atkinson. *Plots, Transformations, and Regression: An Introduction to Graphical Methods of Diagnostic Regression Analysis*. Oxford statistical science series. Clarendon Press, 1985. URL https://books.google.pl/books?id=oFjgnQEACAAJ. [p]

J. Bi and K. P. Bennett. Regression error characteristic curves. In *ICML*, 2003. [p]

P. Biecek. DALEX: Explainers for Complex Predictive Models. *ArXiv e-prints*, 2018. [p]

G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society B*, pages 211–252, 1964. [p]

L. Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statist. Sci.*, 16(3):199–231, 2001. URL https://doi.org/10.1214/ss/1009213726. [p]

T. S. Breusch and A. R. Pagan. A simple test for heteroscedasticity and random coefficient variation. *Econometrica*, 47(5):1287–1294, 1979. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1911963. [p]

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016. URL http://arxiv.org/abs/1603.02754. [p]

H. Cramer. On the composition of elementary errors: Second paper: Statistical applications. *Scandinavian Actuarial Journal*, 1928(1):141–180, 1928. [p]

E. de Jonge and M. van der Loo. *Validatetools: Checking and Simplifying Validation Rule Sets*, 2018. URL https://CRAN.R-project.org/package=validatetools. R package version 0.4.3. [p]

J. J. Faraway. *Practical Regression and Anova Using R*. University of Bath, 2002. URL https://books.google.pl/books?id=UjhBnwEACAAJ. [p]

J. J. Faraway. *Linear Models with R*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2004. ISBN 9780203507278. URL https://books.google.pl/books?id=fvenzpofkagC. [p]

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, 2nd edition, 2011. URL http://socserv.socsci.mcmaster.ca/jfox/Books/Companion. [p]

M. Friendly. Corrgrams: Exploratory displays for correlation matrices. *The American Statistician*, 56(4): 316–324, 2002. [p]

K. P. F.R.S. *X. On the Criterion That a Given System of Deviations from the Probable in the Case of a Correlated System of Variables Is Such That It Can Be Reasonably Supposed to Have Arisen from Random Sampling*, volume 50. Taylor & Francis, 1900. URL https://doi.org/10.1080/14786440009463897. [p]

A. Gałecki and T. Burzykowski. *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*. Springer Texts in Statistics. Springer-Verlag, 2013. ISBN 9781461439004. URL https://books.google.pl/books?id=rbk_AAAAQBAJ. [p]

S. M. Goldfeld and R. E. Quandt. Some tests for homoscedasticity. *Journal of the American Statistical Association*, 60(310):539–547, 1965. URL https://doi.org/10.1080/01621459.1965.10480811. [p]

A. Gosiewska and P. Biecek. auditor: An R Package for Model-Agnostic Visual Validation and Diagnostic. *ArXiv e-prints*, 2018. [p]

J. Gross and U. Ligges. *Nortest: Tests for Normality*, 2015. URL https://CRAN.R-project.org/package=nortest. R package version 1.0-4. [p]

F. E. Harrell Jr. *Regression Modeling Strategies*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387952322. [p]

F. E. Harrell Jr. *Rms: Regression Modeling Strategies*, 2018. URL https://CRAN.R-project.org/package=rms. R package version 5.1-2. [p]

M. J. Harrison and B. P. M. McCabe. A test for heteroscedasticity based on ordinary least squares residuals. *Journal of the American Statistical Association*, 74(366):494–499, 1979. ISSN 01621459. URL http://www.jstor.org/stable/2286361. [p]

A. C. Harvey and P. Collier. Testing for functional misspecification in regression analysis. *Journal of Econometrics*, 6(1):103 – 119, 1977. ISSN 0304-4076. URL https://doi.org/10.1016/0304-4076(77)90057-4. [p]

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer-Verlag, New York, NY, USA, 2001. [p]

T. K. Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7128-9. URL http://dl.acm.org/citation.cfm?id=844379.844681. [p]

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL https://CRAN.R-project.org/doc/Rnews/. [p]

R. C. Littell, G. A. Milliken, W. W. Stroup, R. D. Wolfinger, and O. Schabenberger. *SAS for Mixed Models, Second Edition*. SAS Institute, 2007. ISBN 9781599940786. URL https://books.google.pl/books?id=z9qv32OyEu4C. [p]

S. Liu, X. Wang, M. Liu, and J. Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 1(1):48 – 56, 2017. ISSN 2468-502X. URL https://doi.org/10.1016/j.visinf.2017.01.006. [p]

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. [p]

R. Moral, J. Hinde, and C. Demétrio. Half-normal plots and overdispersed models in r: The hnp package. *Journal of Statistical Software, Articles*, 81(10):1–23, 2017. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v081.i10. [p]

C. O'Neil. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. Crown Publishing Group, New York, NY, USA, 2016. ISBN 0553418815, 9780553418811. [p]

A. H. Petersen and C. T. Ekstrom. *dataMaid: A Suite of Checks for Identification of Potential Errors in a Data Frame as Part of the Data Screening Process*, 2018. URL https://CRAN.R-project.org/package=dataMaid. R package version 1.1.2. [p]

E. A. Peña and E. H. Slate. Global validation of linear model assumptions. *Journal of the American Statistical Association*, 101(473):341–354, 2006. URL https://doi.org/10.1198/016214505000000637. PMID: 20157621. [p]

J. B. Ramsey. Tests for specification errors in classical linear least-squares regression analysis. *Journal of the Royal Statistical Society B*, 31(2):350–371, 1969. ISSN 00359246. URL http://www.jstor.org/stable/2984219. [p]

D. Robinson. *Broom: Convert Statistical Analysis Objects into Tidy Data Frames*, 2018. URL https://CRAN.R-project.org/package=broom. R package version 0.4.4. [p]

S. Sanford Shapiro and R. S. Francia. An approximate analysis of variance test for normality. *Journal of the American Statistical Association*, 67:215–216, 1972. [p]

S. Sheather. *A Modern Approach to Regression with R*. Springer Texts in Statistics. Springer-Verlag, 2009. ISBN 9780387096070. URL https://books.google.pl/books?id=zS3Jiyxqr98C. [p]

C. Sievert, C. Parmer, T. Hocking, S. Chamberlain, K. Ram, M. Corvellec, and P. Despouy. *Plotly: Create Interactive Web Graphics via 'plotly.js'*, 2017. URL https://CRAN.R-project.org/package=plotly. R package version 4.7.1. [p]

M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974. ISSN 01621459. URL http://www.jstor.org/stable/2286009. [p]

J. Swets. Measuring the accuracy of diagnostic systems. *Science*, 240(4857):1285–1293, 1988. ISSN 0036-8075. URL https://doi.org/10.1126/science.3287615. [p]

Y. Tang. Autoplotly - Automatic Generation of Interactive Visualizations for Popular Statistical Results. *ArXiv e-prints*, 2018. [p]

Y. Tang, M. Horikoshi, and W. Li. ggfortify: Unified Interface to Visualize Statistical Results of Popular R Packages. *The R Journal*, 8(2):474–485, 2016. URL https://journal.r-project.org/archive/2016/RJ-2016-060/index.html. [p]

J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley series in behavioral science. Addison-Wesley Publishing Company, 1977. ISBN 9780201076165. URL https://books.google.pl/books?id=UT9dAAAAIAAJ. [p]

J. M. Utts. The rainbow test for lack of fit in regression. *Communications in Statistics - Theory and Methods*, 11(24):2801–2815, 1982. URL https://doi.org/10.1080/03610928208828423. [p]

M. van der Loo. *Lumberjack: Track Changes in Data*, 2017. URL https://CRAN.R-project.org/package=lumberjack. R package version 0.2.0. [p]

A. W. van der Vaart. *Asymptotic Statistics*. Asymptotic Statistics. Cambridge University Press, 2000. ISBN 9780521784504. URL https://books.google.pl/books?id=UEuQEM5RjWgC. [p]

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. [p]

R. Von Mises. *Wahrscheinlichkeit, Statistik Und Wahrheit*. Number t. 3 in Schriften zur wissenschaftlichen Weltauffassung. Springer-Verlag, 1928. URL https://books.google.pl/books?id=W1IaAAAAIAAJ. [p]

H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. URL http://ggplot2.org. [p]

H. Wickham. Tidy data. *The Journal of Statistical Software*, 59, 2014. URL http://www.jstatsoft.org/v59/i10/. [p]

C. J. Willmott, S. G. Ackleson, R. E. Davis, J. J. Feddema, K. M. Klink, D. R. Legates, J. O'Donnell, and C. M. Rowe. Statistics for the evaluation and comparison of models. *Journal of Geophysical Research*, 90(C5):8995, 1985. URL https://doi.org/10.1029/jc090ic05p08995. [p]

K. Wright. *Corrgram: Plot a Correlogram*, 2018. URL https://CRAN.R-project.org/package=corrgram. R package version 1.13. [p]

M. N. Wright and A. Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017. URL https://doi.org/10.18637/jss.v077.i01. [p]

A. Zeileis and T. Hothorn. Diagnostic checking in regression relationships. *R News*, 2(3):7–10, 2002. URL https://CRAN.R-project.org/doc/Rnews/. [p]

A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber. Strucchange: An r package for testing for structural change in linear regression models. *Journal of Statistical Software, Articles*, 7(2):1–38, 2002. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v007.i02. [p]

Alicja Gosiewska
Faculty of Mathematics and Information Science
Warsaw University of Technology
Poland
ORCID: https://orcid.org/0000-0001-6563-5742
alicjagosiewska@gmail.com

Przemysław Biecek
Faculty of Mathematics and Information Science
Warsaw University of Technology
Faculty of Mathematics, Informatics and Mechanic
University of Warsaw
Poland
ORCID: https://orcid.org/0000-0001-8423-1823
przemyslaw.biecek@gmail.com

The R Package trafo for Transforming Linear Regression Models

by Lily Medina, Ann-Kristin Kreutzmann, Natalia Rojas-Perilla and Piedad Castro

Abstract Researchers and data-analysts often use the linear regression model for descriptive, predictive, and inferential purposes. This model relies on a set of assumptions that, when not satisfied, yields biased results and noisy estimates. A common problem that can be solved in many ways – use of less restrictive methods (e.g. generalized linear regression models or non-parametric methods), variance corrections or transformations of the response variable just to name a few. We focus on the latter option as it allows to keep using the simple and well-known linear regression model. The list of transformations proposed in the literature is long and varies according to the problem they aim to solve. Such diversity can leave analysts lost and confused. We provide a framework implemented as an R-package, **trafo**, to help select suitable transformations depending on the user requirements and data being analyzed. The package **trafo** contains a collection of selected transformations and estimation methods that complement and increase the breadth of methods that exist in R.

Introduction

To study the relation between two or more variables, the linear regression model is one of the most employed statistical methods. For an appropriate usage of this model, a set of assumptions needs to be fulfilled. These assumptions are, among others, related to the functional form and to the error terms, such as linearity and homoscedasticity. However, in practical applications, these assumptions are not always satisfied. This leads to the question of how to move on with the analysis in such cases. One way to proceed is to conduct the analysis ignoring the model assumption violations which is, of course, not recommended as it would likely yield misleading results. An alternative solution is to use more complex methods such as generalized linear regression models or non-parametric methods, as they might fit the data and problem better. A third method–and the focus of the present work– is the application of suitable transformations. Throughout the current manuscript, we use the term transformations to refer to the application of monotonic functions to the response variable of a linear regression model. For more flexible transformation functions, please refer to (e.g.) Hothorn et al. (2018).

Transformations have the potential to correct certain violations of model assumptions and by doing so, allow an analysis to continue with the linear regression model. Due to its convenience, transformations such as the logarithm or the Box-Cox (Box and Cox, 1964) are commonly applied in many branches of sciences; for example in economics (Hossain, 2011) and neuroscience (Morozova et al., 2016). In order to simplify the choice and the usage of transformations in the linear regression model, the R package **trafo** (Medina et al., 2018) is developed. The present work is inspired by the framework proposed in Rojas-Perilla (2018, pp. 9-45) and extends other existing R packages that provide transformations.

Many packages that contain transformations do not specifically focus on the use of transformations (Venables and Ripley, 2002; Fox and Weisberg, 2011; Molina and Marhuenda, 2015; Ribeiro Jr. and Diggle, 2016). They often only include widely used transformations like the logarithmic or the Box-Cox transformation family. The package **car** (Fox and Weisberg, 2011) expands the selection of transformations; it includes the Box-Cox, the Tukey (Tukey, 1957), and the Yeo-Johnson (Yeo and Johnson, 2000) transformation families, and uses the maximum likelihood approach for the estimation of the transformation parameter (Box and Cox, 1964). The package **rcompanio** (Mangiafico, 2019) focuses on the Tukey transformation with estimation via goodness of fit tests. In addition to the logarithm and Box-Cox, the package **bestNormalize** (Peterson, 2019) also includes the glog (see e.g. Durbin et al., 2002) and Yeo-Johnson transformations. An exponential transformation proposed by Manly (1976) is provided in the package **caret** (Kuhn, 2008) and the multiple parameter Johnson transformation (Johnson, 1949) in the packages **Johnson** (Fernandez, 2014) and **jtrans** (Wang, 2015). While the packages **MASS** (Venables and Ripley, 2002) and **car** (Fox and Weisberg, 2011) solely provide the maximum likelihood approach for the estimation of the transformation parameter for the Box-Cox family, the package **AID** (Dag et al., 2017) includes a wide range of methods, mostly based on goodness of fit tests like the Shapiro-Wilk or the Anderson-Darling test. Though the use of these methods is limited to the Box-Cox transformation. For a summary of the various transformations available in R packages, please see Table 1.

It is noticeable that most of the above-mentioned packages do not help the user in the process of deciding which transformation is actually suitable according to the users needs. Furthermore,

Table 1: Overview of available transformations and estimation methods in R packages

	AID	bestNormalize	car	caret	Johnson	jtrans	MASS	rcompanion	trafo
Transformation									
Log	✗	✗	✗	✗			✗		✗
Log (shift)	✗	✗	✗				✗		✗
Glog		✗							✗
Neglog									✗
Reciprocal		✗	✗						✗
Tukey			✗					✗	
Box-Cox	✗	✗	✗	✗			✗		✗
Box-Cox (shift)	✗		✗						✗
Log-shift opt									✗
Bickel-Docksum									✗
Yeo-Johnson		✗	✗	✗	✗	✗			✗
Square Root (shift)									✗
Manly				✗					✗
Modulus									✗
Dual									✗
Gpower									✗
Customized									✗
Estimation method									
Maximum likelihood theory	✗		✗	✗			✗		✗
Distribution moments optimization									✗
Divergence minimization									✗
Via goodness of fit tests	✗					✗		✗	
Rank-mapping		✗							
Via percentiles					✗				

most packages do not provide tools to "eyeball" whether the employed transformation improves the data with regard to fulfilling the model assumptions. Package **trafo** combines and extends the features provided by the packages mentioned above. Additionally to transformations that are already provided by existing packages, the **trafo** package includes, among others, the Bickel-Doksum (Bickel and Doksum, 1981), modulus (John and Draper, 1980), the neglog (Whittaker et al., 2005) and glog (see e.g. Durbin et al., 2002) transformations that are modifications of the Box-Cox and the logarithmic transformation in order to deal with negative values in the response variable. The selection of estimation methods for the transformation parameter is enlarged by methods based on moments and divergence measures (see e.g. Taylor, 1985; Yeo and Johnson, 2000; Royston et al., 2011). The main benefits of the package **trafo** can be summarized as follows:

- An initial check can be conducted that helps to decide if and which transformation is useful for the researchers' needs.
- The untransformed model and a model with a transformed dependent variable can be easily compared under the light of the model assumptions (more on this below). Alternatively, two transformed models can be run and compared simultaneously
- Extensive diagnostics are provided in order to check if the transformation helps to fulfill the model assumptions normality, homoscedasticity, and linearity.

Transformations and estimation methods

The equation describing and summarizing the relationship between a continuous outcome variable y and different covariates x (either categorical or continuous) is defined by $y_i = \mathbf{x}_i^T \boldsymbol{\beta} + e_i$, with $i = 1, \ldots, n$. This is also known as the linear regression model and is composed by a deterministic and a random component, which rely on different assumptions. Among others, these assumptions can be summarized as follows:

- Normality (N): The conditional distribution of y given x follows a normal distribution. This is an optional, but often desired assumption (e.g. Box and Cox (1964)).
- Homoscedasticity (H): The conditional variance of y given x is constant.
- Linearity (L): The conditional expectation of the outcome variable y given the continuous covariates x is a linear function in x.

As already mentioned, different approaches have been proposed to overcome the violations of these model assumptions. Some of them include alternative estimation methods of the regression terms or more complex regression models (see e.g. Nelder and Wedderburn, 1972; Berry, 1993). In the present manuscript, we focus on defining a parsimonious modification for the model, such as the usage of non-linear transformations of the outcome variable. The transformations implemented in the

package **trafo** particularly help to achieve normality. However, most of them simultaneously correct other assumptions (see also Table 2 and Table 3).
We classify transformations in two groups: non-parametric transformations and data-driven transformations with a transformation parameter that needs to be estimated. The first set of transformations presented in Table 2 comprises, among others, the logarithmic transformation, which is considered due to its popularity and straightforward application. The data-driven transformations presented in

Table 2: Non-parametric transformations

Transformation	Source	Formula	Support	N	H	L
Log (shift)	Box and Cox (1964)	$\log(y+s)$	$y \in \{-s; \infty\}$	✗	✗	✗
Glog	Rocke and Durbin (2001) Durbin et al. (2002) Huber et al. (2002, 2003)	$\log(y+\sqrt{y^2+1})$	$y \in \mathbb{R}$	✗	✗	✗
Neglog	Whittaker et al. (2005)	$\text{Sign}(y)\log(\lvert y\rvert+1)$	$y \in \mathbb{R}$	✗	✗	
Reciprocal	Tukey (1957)	$\frac{1}{y}$	$y \neq 0$	✗	✗	

Table 3 are dominated by the Box-Cox transformation and its modifications or alternatives, e.g. the modulus or Bickel-Doksum transformation. More flexible versions of the logarithmic transformation, as the log-shift opt, or the Manly transformation, an exponential transformation, are also included in the package **trafo**.

Table 3: Data-driven transformations.

Transformation	Source	Formula	Support	N	H	L
Box-Cox (shift)	Box and Cox (1964)	$\begin{cases} \frac{(y+s)^\lambda-1}{\lambda} & \text{if } \lambda \neq 0; \\ \log(y+s) & \text{if } \lambda = 0. \end{cases}$	$y \in \{-s; \infty\}$	✗	✗	✗
Log-shift opt	Feng et al. (2016)	$\log(y+\lambda)$	$y \in \{-s; \infty\}$	✗	✗	✗
Bickel-Docksum	Bickel and Doksum (1981)	$\frac{\lvert y\rvert^\lambda \text{Sign}(y)-1}{\lambda}$ if $\lambda > 0$	$y \in \mathbb{R}$	✗	✗	
Yeo-Johnson	Yeo and Johnson (2000)	$\begin{cases} \frac{(y+1)^\lambda-1}{\lambda} & \text{if } \lambda \neq 0, y \geq 0; \\ \log(y+1) & \text{if } \lambda = 0, y \geq 0; \\ \frac{(1-y)^{2-\lambda}-1}{\lambda-2} & \text{if } \lambda \neq 2, y < 0; \\ -\log(1-y) & \text{if } \lambda = 2, y < 0. \end{cases}$	$y \in \mathbb{R}$	✗	✗	
Square Root (shift)	Medina et al. (2018)	$\sqrt{y+\lambda}$	$y \in \mathbb{R}$	✗	✗	
Manly	Manly (1976)	$\begin{cases} \frac{e^{\lambda y}-1}{\lambda} & \text{if } \lambda \neq 0; \\ y & \text{if } \lambda = 0. \end{cases}$	$y \in \mathbb{R}$	✗	✗	
Modulus	John and Draper (1980)	$\begin{cases} \text{Sign}(y)\frac{(\lvert y\rvert+1)^\lambda-1}{\lambda} & \text{if } \lambda \neq 0; \\ \text{Sign}(y)\log(\lvert y\rvert+1) & \text{if } \lambda = 0. \end{cases}$	$y \in \mathbb{R}$	✗		
Dual	Yang (2006)	$\begin{cases} \frac{(y^\lambda-y^{-\lambda})}{2\lambda} & \text{if } \lambda > 0; \\ \log(y) & \text{if } \lambda = 0. \end{cases}$	$y > 0$	✗		
Gpower	Kelmansky et al. (2013)	$\begin{cases} \frac{(y+\sqrt{y^2+1})^\lambda-1}{\lambda} & \text{if } \lambda \neq 0; \\ \log(y+\sqrt{y^2+1}) & \text{if } \lambda = 0. \end{cases}$	$y \in \mathbb{R}$	✗		

Table 2 and 3 provide information about the range y that is supported by the transformation. Some transformations are only suitable for positive values of y. This is generally true for the logarithmic and Box-Cox transformations. However, in case that the dependent variable contains negative values, the values are shifted by a deterministic shift s such that $y + s > 0$ by default in package **trafo**. Furthermore, the tables highlights which assumptions the transformation helps to achieve. Kindly note that we are proposing general suggestions and the benefits of transformations depend on the data. For specific properties of each transformation we refer to the further references. The square root shift transformation with a data-driven shift in analogy to the log-shift opt transformation is, to the best of our knowledge, firstly implemented in this work. In contrast, a square root transformation with deterministic shift, for example, is suggested in Bartlett (1947).

Below, we summarize the collection of methods included in **trafo** to estimate the parameters of the transformations presented in Table 3. The benefit of each estimation method depends on the research analysis and the underlying data.

- Maximum likelihood theory (Box and Cox, 1964)
- Distribution moments optimization: Skewness or kurtosis (Carroll and Ruppert, 1987; Royston et al., 2011; Rojas-Perilla, 2018)
- Divergence minimization: Following Kolmogorov-Smirnov (KS), Cramér-von-Mises (KM) or Kullback-Leibler (KL) measurements (Cramér, 1928; Kolmogorov, 1933; Smirnov, 1939; Kullback and Leibler, 1951; Hernandez and Johnson, 1980; Rojas-Perilla, 2018)

Table 4: Diagnostic checks provided in the package **trafo**.

Assumption	Diagnostic check	Fast check
Normality	Skewness and kurtosis	✗
	Shapiro-Wilk/Anderson-Darling test	✗
	Quantile-quantile plot	
	Histograms	
Homoscedasticity	Breusch-Pagan test	✗
	Residuals vs. fitted plot	
	Scale-location	
Linearity	Scatter plots between y and x	✗
	Observed vs. fitted plot	

The maximum likelihood estimation method finds the set of values for the transformation parameter that maximizes the likelihood function of the dataset under the selected transformation (Box and Cox, 1964). This is a standard approach that is also implemented in several of the mentioned R packages (Venables and Ripley, 2002; Fox and Weisberg, 2011). However, since the maximum likelihood estimation is rather sensitive to outliers, the skewness or kurtosis optimization might be preferable for the estimation of the transformation parameter in the presence of outliers (see e.g. Royston et al., 2011). The use of kurtosis over skewness optimization depends entirely on the shape of the distribution of the data and the goal of the analyst – skewness optimizations corrects for asymmetry and kurtosis for light or heavy tails. Additionally, if the focus lies on comparing the whole distribution of the transformed data with a normal distribution, and not only on some moments, different divergence measures as the KS, KM or KL can be used (see e.g. Yeo and Johnson, 2000). For all estimation methods, a range on which the functions are evaluated needs to be proposed. Therefore, default values are set for the predefined transformations. For more information about different estimation methods we refer to Rojas-Perilla (2018, pp. 9-45).

Since the user can only decide if the transformation is helpful by checking the above mentioned assumptions, the package **trafo** contains a wide range of diagnostic checks (e.g. Anderson and Darling, 1954; Shapiro and Wilk, 1965; Breusch and Pagan, 1979). A smaller selection is used in the fast check that helps to decide if a transformation might be useful. Table 4 summarizes the implemented diagnostic checks that are simultaneously returned for the untransformed and a transformed model or two differently transformed models and indicates which diagnostics are conducted in the fast check. Additionally, plots are provided that help to detect outliers such as the Cook's distance plot and influential observations by the residuals vs leverage plot.

Another feature of the package **trafo** is the possibility of defining a customized transformation. Thus, a user can also use the infrastructure of the package for a transformation that suits the individuals needs better than the predefined transformations. However, in this version of the package **trafo** the user needs to define the transformation and the standardized transformation in order to use this feature. For the derivation of the standardized transformation of all predefined transformations, see the Appendix.

Applications

The usage of transformations in practice may help to meet model assumptions but it can also lead to complexities as the interpretation of parameters and standard errors in inference or back-transformation biases in prediction (Rojas-Perilla, 2018). For instance, it is questionable how to address the estimation of the transformation parameter in inference. Box and Cox (1964) point out that after applying the Box-Cox transformation to the outcome variable, the transformation parameter should be treated as fixed and known and the subsequent analysis could be done in the transformed scale. However, Bickel and Doksum (1981) emphasize that estimating a transformation parameter in a model could overestimate the parameters' variance yielding conservative confidence intervals. In prediction, on the other hand, lost interpretability of parameters and standard errors may be less important but the back-transformation could lead to a bias neglecting the non-linearity of the transformation (see e.g. Mosimann et al., 2018).

Nevertheless, several studies show how transformations can be useful in applications. Pek et al. (2017) demonstrate how the log transformation can be used for describing the relation between earnings and years of experience and the reciprocal transformation for the effect of intelligence quotient (IQ) on performance on mental sum problems. The logarithm and the Box-Cox transformation are often applied in econometric research, e.g. to describe monetary policies (Zarembka, 1968, 1974). Transformations have also been used to improve the functional form in studies of demand functions for meat (shyong Chang, 1977), travel costs (Vaughan et al., 1982), and recreation (Ziemer et al., 1980) in the U.S and for import equations in the Republic of Ireland (Boylan et al., 1982). Another research field for the application of transformations is genetics (Huber et al., 2003). The data sets often exhibit a

Table 5: Core functions of package **trafo**.

Function	Description
`assumptions()`	Enables a fast check whether A transformation is suitable.
`trafo_lm()`	Compares the untransformed model with a transformed model.
`trafo_compare()`	Compares two differently transformed models.
`diagnostics()`	Returns information about the transformation and different diagnostics checks in form of tests.
`plot()`	Returns graphical diagnostics checks.

high variability and non-normality problems. To address this, the glog and gpower can be useful in practice (Durbin et al., 2002; Kelmansky et al., 2013).

When using package **trafo** for applications, it should be noted that the package focuses on finding a suitable transformation with regards to fulfilling specific model assumptions, the user still has to decide if the transformation is reasonable in a specific application. The following section shows which functionalities the package provides for the user.

Case study

In order to show the functionality of the package **trafo**, we present – in form of a case study – the steps a user faces when checking the assumptions of the linear model. For this illustration, we use the data set called `University` from the R package **Ecdat** (Croissant, 2016). This data set contains variables measuring the equipment and costs of university teaching and research. These data can be made available as follows:

```
R> library(Ecdat)
R> data(University)
```

A practical question for the head of a university could be how study fees (`stfees`) raise the universities net assets (`nassets`). Both variables are metric. Thus, a linear regression could help to explain the relation between these two variables. A linear regression model can be conducted in R using the `lm` function.

```
R> linMod <- lm(nassets ~ stfees, data = University)
```

The features in the package **trafo** that help to find a suitable transformation for this model and to compare different models are summarized in Table 5 and illustrated in the next subsections.

Finding a suitable transformation

It is well known that the reliability of the linear regression model depends on the assumptions presented above. In this section, we focus on presenting how the user can decide and assess which (and whether) transformations help to fulfill these model assumptions. A first fast check of these model assumptions can be used in the package **trafo** in order to find out if the untransformed model meets these assumptions or if using a transformation seems suitable. The fast check can be conducted by the function `assumptions`. This function returns the skewness, the kurtosis and the Shapiro-Wilk/Anderson-Darling test for normality, the Breusch-Pagan test for homoscedasticity and scatter plots between the dependent and the explanatory variables for checking the linear relation. All possible arguments of the function `assumptions` are summarized in Table 6. In the following, we only show the returned normality and homoscedasticity tests. The results are ordered by the p value of the Shapiro-Wilk and Breusch-Pagan test.

```
R> assumptions(linMod)

The default lambdarange for the log shift opt transformation is calculated
dependent on the data range. The lower value is set to -2035.751 and the upper
value to 404527.249

The default lambdarange for the square root shift transformation is calculated
dependent on the data range. The lower value is set to -2035.751 and the upper
value to 404527.249
```

Table 6: Arguments of function `assumptions`.

Argument	Description	Default
`object`	Object of class `lm`.	
`method`	Estimation method for the transformation parameter.	Maximum likelihood
`std`	Normal or standardized transformation.	Normal
`...`	Addtional arguments can be added, especially for changing the lambda range for the estimation of the parameter, e.g. `manly_lr = c(0.000005,0.00005)`.	Default values of lambda range of each transformation

```
Test normality assumption
              Skewness Kurtosis Shapiro_W Shapiro_p
logshiftopt    -0.4201   4.0576    0.9741    0.2132
boxcox         -0.4892   4.2171    0.9621    0.0527
bickeldoksum   -0.4892   4.2171    0.9621    0.0527
gpower         -0.4892   4.2171    0.9621    0.0527
modulus        -0.4892   4.2171    0.9621    0.0527
yeojohnson     -0.4892   4.2171    0.9621    0.0527
dual           -0.4837   4.2180    0.9619    0.0519
sqrtshift       0.6454   5.2752    0.9504    0.0139
log            -1.1653   5.1156    0.9140    0.0004
neglog         -1.1651   5.1150    0.9140    0.0004
glog           -1.1653   5.1156    0.9140    0.0004
untransformed   2.4503  12.7087    0.7922    0.0000
reciprocal     -3.7260  19.0487    0.5676    0.0000

Test homoscedasticity assumption
              BreuschPagan_V BreuschPagan_p
modulus               0.1035         0.7477
yeojohnson            0.1035         0.7477
boxcox                0.1035         0.7476
bickeldoksum          0.1036         0.7476
gpower                0.1035         0.7476
dual                  0.1128         0.7369
logshiftopt           0.1154         0.7341
neglog                0.7155         0.3976
log                   0.7158         0.3975
glog                  0.7158         0.3975
reciprocal            1.6109         0.2044
sqrtshift             5.4624         0.0194
untransformed         9.8244         0.0017
```

Following the Shapiro-Wilk test, the log-shift opt transformation yields a transformed outcome variable that is (statistically) normally distributed ($p = 0.2132$). The same applies for the Box-Cox, Bickel-Doksum, gpower, modulus and Yeo-Johnson transformations though at lower significance level ($\alpha = 0.05$). For improving the homoscedasticity assumption, all transformations help except the square root (shift) transformation. As mentioned before, default values for the range of lambda for all transformations are predefined and these are used in this fast check. Since the default values for the log-shift opt and square root (shift) transformation depend on the range of the response variable, the chosen range is reported in the return. The Manly transformation is not in the list since the default lambda range for the estimation of the transformation parameter is not suitable for this data set. For such a case, the user can change the lambda range for the transformations manually. Similarly, the user can change the estimation methods for the transformation parameter. For instance, if symmetry is of special interest for the user the skewness minimization might be a better choice than the default maximum likelihood method. In this case study all assumptions are assumed to be equally important. Thus, we choose the Box-Cox transformation for the further illustrations even though some other transformations would be suitable as well.

Comparing the untransformed model with a transformed model

For a more detailed comparison of the transformed model with the untransformed model, a function called `trafo_lm` (for the arguments see Table 7) can be used as follows:

```
R> linMod_trafo <- trafo_lm(linMod)
```

The Box-Cox transformation is the default option such that only the `lm` object needs to be given to the function. The object `linMod_trafo` is of class `trafo_lm` and the user can conduct the methods `print`, `summary` and `plot` in the same way as for an object of class `lm`. The difference is that the new methods simultaneously return the results for both models, the untransformed model and the transformed model. Furthermore, a method called `diagnostics` helps to compare results of normality and homoscedasticity tests. In the following, we will show the return of the `diagnostics` method and some selected plots in order to check the normality, homoscedasticity, and the linearity assumption of the linear regression model.

```
R> diagnostics(linMod_trafo)

Diagnostics: Untransformed vs transformed model

Transformation:  boxcox
Estimation method:  ml
Optimal Parameter:  0.1894257

Residual diagnostics:

Normality:
Pearson residuals:
                      Skewness  Kurtosis Shapiro_W    Shapiro_p
Untransformed model  2.4503325 12.708681 0.7921672 6.024297e-08
Transformed model   -0.4892222  4.217105 0.9620688 5.267566e-02

Heteroscedasticity:
                   BreuschPagan_V BreuschPagan_p
Untransformed model      9.8243555     0.00172216
Transformed model        0.1035373     0.74762531
```

The first part of the output shows information of the applied transformation. As chosen, the Box-Cox transformation is used with the optimal transformation parameter around 0.19 which is estimated using the maximum likelihood approach that is also set as default. The optimal transformation parameter differs from 0, which would be equal to the logarithmic transformation, and 1, which means that no transformation is optimal. The Shapiro-Wilk test rejects normality of the residuals of the untransformed model but it does not reject normality for the residuals of the transformed model on a 5% level of significance. Furthermore, the skewness shows that the residuals in the transformed model are more symmetric and the kurtosis is closer to 3, the value of the kurtosis of the normal distribution. The results of the Breusch-Pagan test clearly show that homoscedasticity is rejected in the untransformed model but not in the transformed model. These two findings can be supported by diagnostic plots shown in Figure 1.

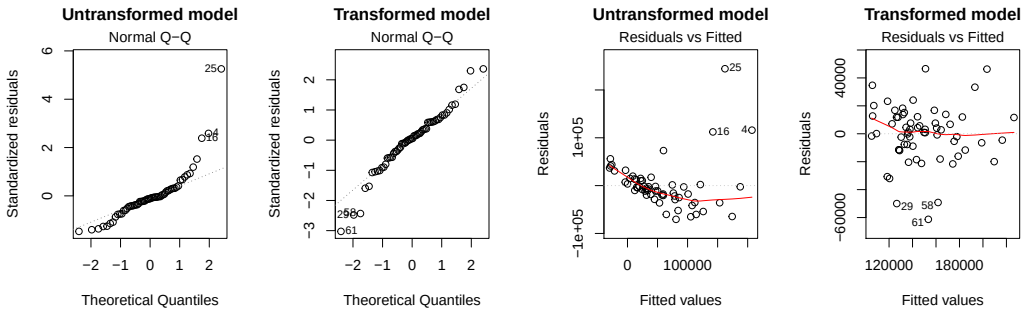
```
R> plot(linMod_trafo)
```

In order to evaluate the linearity assumption, scatter plots of the dependent variable against the explanatory variable can help. Figure 2 shows that the assumption of linearity is violated in the untransformed model. The upper panel shows the Pearson correlation coefficient. In contrast, the relation between the transformed net assets and the study fees seems to be linear. As shown above, the user can obtain diagnostics for an untransformed and a transformed model with only a little more effort in comparison to fitting the standard linear regression model without transformation. While we only show the example with the default transformation, the user can also easily change the transformation and the estimation method. For instance, the user could choose the log-shift opt transformation with the skewness minimization as estimation method.

```
R> linMod_trafo2 <- trafo_lm(object = linMod, trafo = ''logshiftopt'',
+   method = "skew")
```

Compare two transformed models

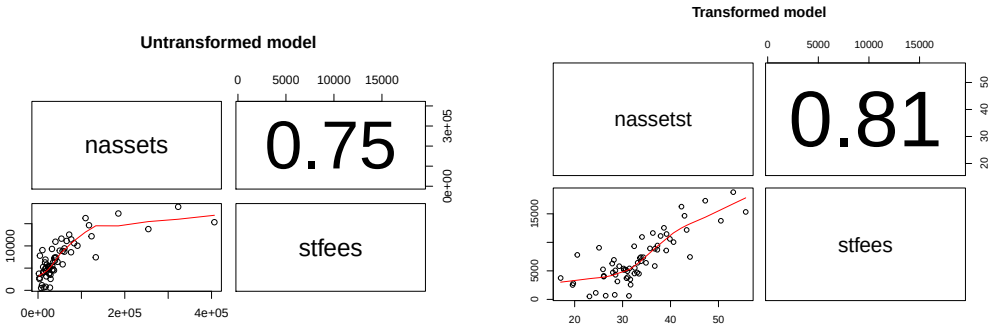
The user can also compare different transformations within the frame of the model assumptions. Oftentimes the logarithm is blindly used without much consideration about its usefulness. In order to



(a) Q-Q plots of the error terms.

(b) Residuals versus fitted.

Figure 1: Selection of diagnostic plots obtained by using `plot(linMod_trafo)`. (a) shows Q-Q plots error terms of the untransformed and the transformed model. (b) shows the residuals against the fitted values of the untransformed and the transformed model.



(a)

(b)

Figure 2: Selection of obtained diagnostic plots by using `plot(linMod_trafo)`. (a) shows the scatter plot of the untransformed net assets and the study fees (b) shows scatter plot of the transformed net assets and the study fees. The numbers specify the correlation coefficient between the dependent and independent variable.

Table 7: Arguments of function `trafo_lm`.

Argument	Description	Default
`object`	Object of class `lm`.	
`trafo`	Selected transformation.	Box-Cox
`lambda`	Estimation or a self-selected numeric value.	Estimation
`method`	Estimation method for the transformation parameter.	Maximum likelihood
`lambdarange`	Determines range for the estimation of the transformation parameter.	Default lambdarange for each transformation.
`std`	Normal or standardized transformation.	Normal
`custom_trafo`	Add customized transformation.	None

Table 8: Arguments of function `trafo_compare`.

Argument	Description	Default
`object`	Object of class `lm`.	
`trafos`	List of objects of class `trafo`.	
`std`	Normal or standardized transformation.	Normal

compare the logarithm with (e.g.) the selected Box-Cox transformation, the user needs to specify two objects of class `trafo` as follows:

```
R> boxcox_uni <- boxcox(linMod)
R> log_uni <- logtrafo(linMod)
```

The utility of `trafo` objects is twofold. First, the user can use the functions for each transformation in order to simply receive the transformed vector. The `print` method gives first information about the vector and the method `as.data.frame` returns the whole data frame with the transformed variable in the last column. The variable is named as the dependent variable with an added t.

```
R> head(as.data.frame(boxcox_uni))

    nassets stfees nassetst
1   3669.71   2821 19.71248
2  12156.00   4037 26.07723
3 185203.00  17296 47.24867
4 323100.00  18800 53.08840
5  32154.00   9314 32.42140
6  41669.00   7388 34.31882
```

Second, the objects can be used to compare linear models with differently transformed dependent variable using function `trafo_compare`. The arguments of this functions are shown in Table 8. The user creates an object of class `trafo_compare` by:

```
R> linMod_comp <- trafo_compare(object = linMod,
+   trafos = list(boxcox_uni, log_uni))
```

For this object, the user can use the same methods as for an object of class `trafo_lm`. In this work, we only want to show the return of method `diagnostics`.

```
R> diagnostics(linMod_comp)

Diagnostics of two transformed models

Transformations:  Box-Cox and Log
Estimation methods:  ml  and no estimation
Optimal Parameters:  0.1894257  and no parameter


Residual diagnostics:

Normality:
Pearson residuals:
          Skewness Kurtosis Shapiro_W    Shapiro_p
Box-Cox -0.4892222 4.217105 0.9620688 0.0526756632
Log     -1.1653028 5.115615 0.9140135 0.0003534879

Heteroscedasticity:
        BreuschPagan_V BreuschPagan_p
Box-Cox      0.1035373      0.7476253
Log          0.7158162      0.3975197
```

The first part of the return points out that the Box-Cox transformation is a data-driven transformation with a transformation parameter while the logarithmic transformation does not adapt to the data. Furthermore, we can see that normality is rejected for the model with a logarithmic transformed dependent variable, while it is not rejected when the Box-Cox transformation is used. The violation of the homoscedasticity assumption can be fixed by both transformations.

Additional features

Extract the transformed model and vector

The **trafo** package provides focused but limited methods to analyze the model. However, the transformed model can be easily extracted from the `trafo_lm` object.

```
R> class(linMod_trafo$trafo_mod)
[1] "lm"
```

The extracted object is of class `lm` such that all available methods for "lm" objects can also be used for the extracted object.

Similarly, it is possible to get the transformed vector.

```
R> head(linMod_custom$trafo_mod$model)
      nassetst stfees
1     13466771   2821
2    147768336   4037
3  34300151209  17296
4 104393610000  18800
5   1033879716   9314
6   1736305561   7388
```

Customized transformation

As summarized in the introduction, many R packages, including package **trafo**, provide a large number of transformations. Naturally, we do not include the comprehensive list of available transformations as this would be a too ambitious task, though we do acknowledge that depending on the needs of the user, a non-implemented transformation might be of interest (for the wide range of possible transformations, see e.g. Rojas-Perilla, 2018). Motivated by this, we include the option to employ our framework– e.g. the estimation of the transformation parameter – with transformations not provided in our package. In the following lines, we show the application of this future using the Tukey transformation (Tukey, 1957).

In a first step, the transformation and the standardized or scaled transformation need to be defined.

```
R> tukey <- function(y, lambda = lambda) {
+   lambda_cases <- function(y, lambda = lambda) {
+   lambda_absolute <- abs(lambda)
+   if (lambda_absolute <= 1e-12) {
+       y <- log(y)
+   } else {
+       y <- y^2
+   }
+   return(y)}
+   y <- lambda_cases(y = y, lambda = lambda)
+   return(y = y)}

R> tukey_std <- function(y, lambda) {
+   gm <- exp(mean(log(y)))
+   if (abs(lambda) > 1e-12) {
+     y <- (y^lambda) / (lambda * ((gm)^(lambda - 1)))
+   } else {
+     y <- gm * log(y)
+   }
+   return(y)}
```

Second, the user inserts the two functions as a list argument to the `trafo_lm` function. Furthermore, the user needs to specify for the `trafo` argument if the transformation is without a parameter (`''custom_wo''`) or with one parameter (`''custom_one''`). The Tukey transformation relies on a transformation parameter. Thus, a `lambdarange` argument will be speified.

```
R> linMod_custom <- trafo_lm(linMod, trafo = "custom_one",
+   lambdarange = c(0, 2), custom_trafo = list(tukey, tukey_std))
```

One limitation of this feature is the necessity to insert both the transformation and the scaled transformation since the latter is often not known. Furthermore, the framework is only suitable for transformations without and with one transformation parameter.

Conclusions and future developments

Although transformations were developed in the absence of efficient machines as an alternative to high memory-consumming methods, they are still a parsimonious way to meet model assumptions for linear regression model. We showed how the package **trafo** helps the user to easily decide whether and which transformations are suitable to fulfill normality, homoscedasticity, and linearity. To the best of our knowledge **trafo** is the only R package that supports this decision process. Furthermore, the package **trafo** provides an extensive collection of transformations usable in linear regression models and a wide range of estimation methods for the transformation parameter. In future versions, we plan to enlarge this collection as well as providing similar functionality for other types of data, e.g. count data. Additionally, more methods that are available for the class `lm` could be developed for objects of class `trafo_lm`. We would also like to expand the infrastructure for linear mixed regression models.

Acknowledgment

We thank Prof. Dr. Timo Schmid for fruitful discussions.

Appendix: Likelihood derivation of the transformations

Log (shift) transformation

Let $J(y)$ denote the Jacobian of a transformation from y_i to y_i^*. In order to obtain z_i^*, the scaled log (shift) transformation, given by $\frac{y_i^*}{J(y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{LS}}$. Therefore, the Jacobian, the scaled, and the inverse of the log (shift) transformation are given below.

The log (shift) transformation presented in Table 2 is defined as:

$$y_i^* = \log(y_i + s).$$

In case, the fixed shift parameter s would not be necessary, the standard logarithm function (logarithmic transformation with $s = 0$) is applied.

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{LS} = \left[\prod_{i=1}^{n} y_i + s\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian is defined as:

$$\begin{aligned} J(\boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*}{dy} \\ &= \prod_{i=1}^{n} \frac{1}{y_i + s} \\ &= \bar{y}_{LS}^{-n}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^* = \log(y_i + s)\, \bar{y}_{LS}.$$

The inverse function of the log (shift) transformation is denoted as:

$$\begin{aligned} f(y_i) &= \log(y_i + s) \\ y_i^* &= \log(y_i + s) \\ y_i &= e^{y_i^*} - s \\ \Rightarrow f^{-1}(y_i^*) &= e^{y_i^*} - s. \end{aligned}$$

Glog transformation

Let $J(y)$ denote the Jacobian of a transformation from y_i to y_i^*. In order to obtain z_i^*, the scaled glog transformation, given by $\frac{y_i^*}{J(y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{GL}}$. Therefore, the Jacobian, the scaled, and the inverse of the glog transformation are given below.

The glog transformation presented in Table 2 is defined as:

$$y_i^* = \log\left(y_i + \sqrt{y_i^2+1}\right) \text{ if } \lambda = 0.$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{GL} = \left[\prod_{i=1}^{n} 1 + y_i^2\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian is defined as:

$$\begin{aligned}
J(\boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*}{dy} \\
&= \prod_{i=1}^{n} \frac{1}{y_i + \sqrt{y_i^2+1}} \left(1 + \frac{2y_i}{2\sqrt{y_i^2+1}}\right) \\
&= \prod_{i=1}^{n} \frac{1}{y_i + \sqrt{y_i^2+1}} \left(\frac{y_i + \sqrt{y_i^2+1}}{\sqrt{y_i^2+1}}\right) \\
&= \prod_{i=1}^{n} \frac{1}{\sqrt{y_i^2+1}} \\
&= \bar{y}_{GL}^{\frac{-n}{2}}.
\end{aligned}$$

The scaled transformation is given by:

$$z_i^* = \log\left(y_i + \sqrt{y_i^2+1}\right) \bar{y}_{GL}^{\frac{1}{2}}.$$

The inverse function of the glog transformation is denoted as:

$$\begin{aligned}
f(y_i) &= \log\left(y_i + \sqrt{y_i^2+1}\right) \\
y_i^* &= \log\left(y_i + \sqrt{y_i^2+1}\right) \\
e^{y_i^*} - y_i &= \sqrt{y_i^2+1} \\
\left(e^{y_i^*} - y_i\right)^2 &= y_i^2 + 1 \\
e^{y_i^{*2}} - 2e^{y_i^*} y_i &= 1 \\
y_i &= -\frac{\left(1 - e^{y_i^{*2}}\right)}{2e^{y_i^*}} \\
\Rightarrow f^{-1}(y_i^*) &= -\frac{\left(1 - e^{y_i^{*2}}\right)}{2e^{y_i^*}}.
\end{aligned}$$

Neglog transformation

Let $J(y)$ denote the Jacobian of a transformation from y_i to y_i^*. In order to obtain z_i^*, the scaled neglog transformation, given by $\frac{y_i^*}{J(y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{NL}}$. Therefore, the Jacobian, the scaled, and the inverse of the neglog transformation are given below.

The neglog transformation presented in Table 2 is defined as:

$$y_i^* = \text{sign}(y_i) \log(|y_i| + 1).$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{NL} = \left[\prod_{i=1}^{n} (|y_i| + 1) \right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian comes to:

$$\begin{aligned} J(\boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*}{dy} \\ &= \prod_{i=1}^{n} \text{sign}(y_i) \frac{1}{|y_i| + 1} \\ &= \text{sign}\left(\prod_{i=1}^{n} y_i \right) \left(\prod_{i=1}^{n} |y_i| + 1 \right)^{-1} \\ &= \text{sign}\left(\prod_{i=1}^{n} y_i \right) \bar{y}_{NL}^{-n}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^* = \text{sign}(y_i) \log(|y_i| + 1) \, \text{sign}\left(\prod_{i=1}^{n} y_i \right) \bar{y}_{NL}.$$

The inverse function of the neglog transformation is denoted as:

$$\begin{aligned} f(y_i) &= \text{sign}(y_i) \log(|y_i| + 1) \\ y_i^* &= \text{sign}(y_i) \log(|y_i| + 1) \\ |y_i| &= e^{\text{sign}(y_i^*) y_i^*} - 1 \\ \Rightarrow f^{-1}(y_i^*) &= \pm \left[e^{\text{sign}(y_i^*) y_i^*} - 1 \right]. \end{aligned}$$

Reciprocal transformation

Let $J(y)$ denote the Jacobian of a transformation from y_i to y_i^*. In order to obtain z_i^*, the scaled reciprocal transformation, given by $\frac{y_i^*}{J(y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{R}}$. Therefore, the Jacobian, the scaled, and the inverse of the reciprocal transformation are given below.

The reciprocal transformation presented in Table 2 is defined as:

$$y_i^* = \frac{1}{y_i}.$$

The definition of the geometric mean is:

$$\bar{y}_R = \left[\prod_{i=1}^{n} y_i \right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian is defined as:

$$\begin{aligned} J(\boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*}{dy} \\ &= \prod_{i=1}^{n} -\frac{1}{y_i^2} \\ &= -\bar{y}_R^{-2n}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^* = -\frac{1}{y_i}\bar{y}_R^2.$$

The inverse function of the reciprocal transformation is denoted as:

$$\begin{aligned} f(y_i) &= \frac{1}{y_i} \\ y_i^* &= \frac{1}{y_i} \\ y_i &= \frac{1}{y_i^*} \\ \Rightarrow f^{-1}(y_i^*) &= \frac{1}{y_i^*}. \end{aligned}$$

Box-Cox (shift) transformation

$$y_i^*(\lambda) = \begin{cases} \frac{(y_i+s)^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \quad (A); \\ \log(y_i + s) & \text{if } \lambda = 0 \quad (B). \end{cases}$$

Box-Cox (shift) transformation case (A)

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain $z_i^*(\lambda)$, the scaled Box-Cox (shift)(A) transformation, given by $\frac{y_i^*(\lambda)}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{BC}}$. Therefore, the Jacobian, the scaled, and the inverse of the Box-Cox (shift)(A) transformation are given below.

The Box-Cox (shift)(A) transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = \frac{(y_i + s)^\lambda - 1}{\lambda} \text{ if } \lambda \neq 0.$$

In case, the fixed shift parameter s is not necessary for making the dataset positive, the standard Box-Cox transformation (with $s = 0$) is applied.

The definition of the geometric mean is:

$$\bar{y}_{BC} = \left[\prod_{i=1}^{n} y_i + s\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian comes to:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*(\lambda)}{dy} \\ &= \prod_{i=1}^{n} \frac{\lambda(y_i + s)^{\lambda-1}}{\lambda} \\ &= \prod_{i=1}^{n} (y_i + s)^{\lambda-1} \\ &= \bar{y}_{BC}^{n(\lambda-1)}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^*(\lambda) = \frac{(y_i + s)^\lambda - 1}{\lambda} \frac{1}{\bar{y}_{BC}^{\lambda-1}}.$$

The inverse function of the Box-Cox (shift)(A) transformation is denoted as:

$$\begin{aligned} f(y_i) &= \frac{(y_i+s)^{\lambda}-1}{\lambda} \\ y_i^* &= \frac{(y_i+s)^{\lambda}-1}{\lambda} \\ y_i &= (\lambda y_i^*+1)^{\frac{1}{\lambda}}-s \\ \Rightarrow f^{-1}(y_i^*) &= (\lambda y_i^*+1)^{\frac{1}{\lambda}}-s. \end{aligned}$$

Box-Cox (shift) transformation case (B)

This case is exactly equal to the log (shift) case.

Log-shift opt transformation

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain $z_i^*(\lambda)$, the scaled log-shift opt transformation, given by $\frac{y_i^*(\lambda)}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{LSO}}$. Therefore, the Jacobian, the scaled, and the inverse of the log-shift opt transformation are given below.

The log-shift opt transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = \log(y_i+\lambda).$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{LSO} = \left[\prod_{i=1}^{n} y_i+\lambda\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian is defined as:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*(\lambda)}{dy} \\ &= \prod_{i=1}^{n} \frac{1}{y_i+\lambda} \\ &= \bar{y}_{LSO}^{-n}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^*(\lambda) = \log(y_i+\lambda)\bar{y}_{LSO}.$$

The inverse function of the log-shift opt transformation is denoted as:

$$\begin{aligned} f(y_i) &= \log(y_i+\lambda) \\ y_i^* &= \log(y_i+\lambda) \\ y_i &= e^{y_i^*}-\lambda \\ \Rightarrow f^{-1}(y_i^*) &= e^{y_i^*}-\lambda. \end{aligned}$$

Bickel-Docksum transformation

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain $z_i^*(\lambda)$, the scaled Bickel-Docksum transformation, given by $\frac{y_i^*(\lambda)}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{BD}}$. Therefore, the Jacobian, the scaled, and the inverse of the Bickel-Docksum transformation are given below.

The Bickel-Docksum transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = \frac{|y_i|^{\lambda}\text{sign}(y_i)-1}{\lambda} \text{ if } \lambda > 0.$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{BD} = \left[\prod_{i=1}^{n} |y_i|\right]^{\frac{1}{n}}.$$

Therefore, the expression of the jacobian comes to:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*(\lambda)}{dy} \\ &= \prod_{i=1}^{n} \frac{\text{sign}(y_i)\lambda|y_i|^{\lambda-1}}{\lambda} \\ &= \text{sign}\left(\prod_{i=1}^{n} y_i\right)\left(\prod_{i=1}^{n} |y_i|\right)^{\lambda-1} \\ &= \text{sign}\left(\prod_{i=1}^{n} y_i\right)\bar{y}_{BD}^{n(\lambda-1)}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^*(\lambda) = \frac{|y_i|^{\lambda}\text{sign}(y_i) - 1}{\lambda} \frac{1}{\text{sign}\left(\prod_{i=1}^{n} y_i\right)\bar{y}_{BD}^{(\lambda-1)}}.$$

The inverse function of the Bickel-Docksum transformation is denoted as:

$$\begin{aligned} f(y_i) &= \frac{|y_i|^{\lambda}\text{sign}\,(y_i) - 1}{\lambda} \\ y_i^* &= \frac{|y_i|^{\lambda}\text{sign}\,(y_i) - 1}{\lambda} \\ |y_i| &= \left[\text{sign}(y_i^*)\,(y_i^*\lambda + 1)\right]^{\frac{1}{\lambda}} \\ \Rightarrow f^{-1}(y_i^*) &= \pm\left[\text{sign}(y_i^*)\,(y_i^*\lambda + 1)\right]^{\frac{1}{\lambda}}. \end{aligned}$$

Yeo-Johnson transformation

$$y_{ij}^*(\lambda) = \begin{cases} \frac{(y_i+1)^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0, y_i \geq 0 \quad (A); \\ \log(y_i + 1) & \text{if } \lambda = 0, y_i \geq 0 \quad (B); \\ -\frac{(1-y_i)^{2-\lambda}-1}{2-\lambda} & \text{if } \lambda \neq 2, y_i < 0 \quad (C); \\ -\log(1 - y_i) & \text{if } \lambda = 0, y_i < 0 \quad (D). \end{cases}$$

Yeo-Johnson transformation case (A)

This case is exactly equal to the Box-Cox (shift) case (A), with $s = 1$.

Yeo-Johnson transformation case (B)

This case is exactly equal to the log (shift) case, with $s = 1$.

Yeo-Johnson transformation case (C)

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain $z_i^*(\lambda)$, the scaled Yeo-Johnson(C) transformation, given by $\frac{y_i^*(\lambda)}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{YC}}$. Therefore, the Jacobian, the scaled, and the inverse of the Yeo-Johnson(C) transformation are given below.

The Yeo-Johnson(C) transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = -\frac{(1 - y_i)^{2-\lambda} - 1}{2 - \lambda} \text{ if } \lambda \neq 2 \text{ and } y_i < 0.$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{YC} = \left[\prod_{i=1}^{n} 1 - y_i\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian comes to:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*(\lambda)}{dy} \\ &= \prod_{i=1}^{n} \frac{(2-\lambda)(1-y_i)^{1-\lambda}}{2-\lambda} \\ &= \prod_{i=1}^{n} (1-y_i)^{1-\lambda} \\ &= \bar{y}_{YC}^{n(1-\lambda)}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^*(\lambda) = -\frac{\left(1-y_{ij}\right)^{2-\lambda} - 1}{2-\lambda} \bar{y}_{YC}^{n(1-\lambda)}.$$

The inverse function of the Yeo-Johnson(C) transformation is denoted as:

$$\begin{aligned} f(y_i) &= -\frac{(1-y_i)^{2-\lambda} - 1}{2-\lambda} \\ y_i^* &= -\frac{(1-y_i)^{2-\lambda} - 1}{2-\lambda} \\ -y_i^*(2-\lambda) &= (1-y_i)^{2-\lambda} - 1 \\ y_i &= 1 - [-y_i^*(2-\lambda) + 1]^{\frac{1}{2-\lambda}} \\ \Rightarrow f^{-1}(y_i^*) &= 1 - [-y_i^*(2-\lambda) + 1]^{\frac{1}{2-\lambda}}. \end{aligned}$$

Yeo-Johnson transformation case (D)

Let $J(y)$ denote the Jacobian of a transformation from y_i to y_i^*. In order to obtain z_i^*, the scaled Yeo-Johnson(D) transformation, given by $\frac{y_i^*}{J(y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\mathrm{YD}}$. Therefore, the Jacobian, the scaled, and the inverse of the Yeo-Johnson(D) transformation are given below.

The Yeo-Johnson(D) transformation presented in Table 3 is defined as:

$$y_i^* = -\log(1-y_i).$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{YD} = \left[\prod_{i=1}^{n} 1 - y_i\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian is defined as:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*}{dy} \\ &= \prod_{i=1}^{n} \frac{1}{1-y_i} \\ &= \bar{y}_{YD}^{-n}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^* = -\log(1-y_i)\,\bar{y}_{YD}.$$

The inverse function of the Yeo-Johnson(D) transformation is denoted as:

$$\begin{aligned} f(y_i) &= -\log(1-y_i) \\ y_i^* &= -\log(1-y_i) \\ y_i &= -e^{-y_i^*}+1 \\ \Rightarrow f^{-1}(y_i^*) &= -e^{-y_i^*}+1. \end{aligned}$$

Square root-shift opt transformation

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain z_i^*, the scaled square root-shift opt transformation, given by $\frac{y_i^*(\lambda)}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\mathrm{SR}}$. Therefore, the Jacobian, the scaled, and the inverse of the square root-shift opt transformation are given below.

The square root-shift opt transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = \sqrt{y_i+\lambda}.$$

The definition of the geometric mean is:

$$\bar{y}_{SR} = \left[\prod_{i=1}^{n} y_i+\lambda\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian is defined as:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*}{dy} \\ &= \prod_{i=1}^{n} -\frac{1}{2\sqrt{y_i+\lambda}} \\ &= \frac{1}{2}\bar{y}_{SR}^{\frac{-n}{2}}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^* = -\frac{1}{y_i}\bar{y}_{SR}^2.$$

The inverse function of the square root-shift opt transformation is denoted as:

$$\begin{aligned} f(y_i) &= \sqrt{y_i+\lambda} \\ y_i^* &= \sqrt{y_i+\lambda} \\ y_i &= y_i^{*2}-\lambda \\ \Rightarrow f^{-1}(y_i^*) &= y_i^{*2}-\lambda. \end{aligned}$$

Manly transformation

$$y_i^*(\lambda) = \begin{cases} \frac{e^{\lambda y_i}-1}{\lambda} & \text{if } \lambda \neq 0 \quad (A); \\ y_i & \text{if } \lambda = 0 \quad (B). \end{cases}$$

Manly transformation case (A)

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain $z_i^*(\lambda)$, the scaled Manly(A) transformation, given by $\frac{y_i^*(\lambda))}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\mathrm{M}}$. Therefore, the Jacobian, the scaled, and the inverse of the Manly(A) transformation are given below.

The Manly(A) transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = \frac{e^{\lambda y_i} - 1}{\lambda} \text{ if } \lambda \neq 0.$$

The modification of the definition of the geometric mean for this transformation is:

$$\begin{aligned} \bar{y}_M &= \left[\prod_{i=1}^{n} e^{y_i}\right]^{\frac{1}{n}} \\ &= \left[e^{\sum_{i=1}^{n} y_i}\right]^{\frac{1}{n}} \\ &= e^{\bar{y}}. \end{aligned}$$

Therefore, the expression of the Jacobian comes to:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*(\lambda)}{dy} \\ &= \prod_{i=1}^{n} \frac{\lambda e^{\lambda y_i}}{\lambda} \\ &= \left(\prod_{i=1}^{n} e^{y_i}\right)^{\lambda} \\ &= \bar{y}_M^{\lambda n} \\ &= e^{\lambda n \bar{y}}. \end{aligned}$$

The scaled transformation is given by:

$$\begin{aligned} z_i^*(\lambda) &= \frac{e^{\lambda y_i} - 1}{\lambda} \frac{1}{\bar{y}_M^{\lambda}} \\ &= \frac{e^{\lambda y_i} - 1}{\lambda} \frac{1}{e^{\lambda \bar{y}}}. \end{aligned}$$

The inverse function of the Manly(A) transformation is denoted as:

$$\begin{aligned} f(y_i) &= \frac{e^{\lambda y_i} - 1}{\lambda} \\ y_i^* &= \frac{e^{\lambda y_i} - 1}{\lambda} \\ \lambda y_i^* + 1 &= e^{\lambda y_i} \\ y_i &= \frac{\log\left(\lambda y_i^* + 1\right)}{\lambda} \\ \Rightarrow f^{-1}(y_i^*) &= \frac{\log\left(\lambda y_i^* + 1\right)}{\lambda}. \end{aligned}$$

Manly transformation case (B)

The variable remains equal, $y_i^* = y_i$.

Modulus transformation

$$y_i^*(\lambda) = \begin{cases} \text{sign}(y_i) \frac{(|y_i|+1)^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0 \quad (A); \\ \text{sign}(y_i) \log(|y_i| + 1) & \text{if } \lambda = 0 \quad (B). \end{cases}$$

Modulus transformation case (A)

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain $z_i^*(\lambda)$, the scaled modulos(A) transformation, given by $\frac{y_i^*(\lambda)}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the

definition of the geometric mean, denoted by $\bar{y}_{\text{MA}}$. Therefore, the Jacobian, the scaled, and the inverse of the modulus(A) transformation are given below.

The modulus(A) transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = \text{sign}\,(y_i)\,\frac{(|y_i|+1)^{\lambda}-1}{\lambda} \text{ if } \lambda \neq 0.$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{MA} = \left[\prod_{i=1}^{n} |y_i|+1\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian comes to:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*(\lambda)}{dy} \\ &= \prod_{i=1}^{n} \frac{\text{sign}\,(y_i)\lambda(|y_i|+1)^{\lambda-1}}{\lambda} \\ &= \text{sign}\left(\prod_{i=1}^{n} y_i\right)\left(\prod_{i=1}^{n}|y_i|+1\right)^{\lambda-1} \\ &= \text{sign}\left(\prod_{i=1}^{n} y_i\right)\bar{y}_{MA}^{n(\lambda-1)}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^*(\lambda) = \text{sign}\,(y_i)\,\frac{(|y_i|+1)^{\lambda}-1}{\lambda}\,\frac{1}{\text{sign}\left(\prod_{i=1}^{n} y_i\right)\bar{y}_{MA}^{(\lambda-1)}}.$$

The inverse function of the modulus(A) transformation is denoted as:

$$\begin{aligned} f(y_i) &= \text{sign}\,(y_i)\,\frac{(|y_i|+1)^{\lambda}-1}{\lambda} \\ y_i^* &= \text{sign}(y_i)\,\frac{(|y_i|+1)^{\lambda}-1}{\lambda} \\ |y_i| &= [\text{sign}\,(y_i^*)\,\lambda+1]^{\frac{1}{\lambda}}-1 \\ \Rightarrow f^{-1}(y_i^*) &= \pm\left[(\text{sign}(y_i^*)\lambda+1)^{\frac{1}{\lambda}}-1\right]. \end{aligned}$$

Modulus transformation case (B)

This case is exactly equal to the neglog transformation case.

Dual power transformation

$$y_i^*(\lambda) = \begin{cases} \frac{y_i^{\lambda}-y_i^{-\lambda}}{2\lambda} & \text{if } \lambda > 0 \quad (A); \\ \log(y_i) & \text{if } \lambda = 0 \quad (B). \end{cases}$$

Dual power transformation case (A)

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain $z_i^*(\lambda)$, the scaled dual power(A) transformation, given by $\frac{y_i^*(\lambda)}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the definition of the geometric mean, denoted by $\bar{y}_{\text{DA}}$. Therefore, the Jacobian, the scaled, and the inverse of the dual power(A) transformation are given below. The dual power(A) transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = \frac{y_i^{\lambda}-y_i^{-\lambda}}{2\lambda} \text{ if } \lambda > 0.$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{DA} = \left[\prod_{i=1}^{n}\left(y_i^{\lambda-1} + y_i^{-\lambda-1}\right)\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian comes to:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*(\lambda)}{dy} \\ &= \prod_{i=1}^{n} \frac{\lambda y_i^{\lambda-1} + \lambda y_i^{-\lambda-1}}{2\lambda} \\ &= \frac{1}{2}\bar{y}_{DA}^{n}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^*(\lambda) = \frac{y_i^{\lambda} - y_i^{-\lambda}}{2\lambda}\frac{2}{\bar{y}_{DA}}.$$

The inverse function of the dual power(A) transformation is found by solving the quadratic by completing the square as:

$$\begin{aligned} f(y_i) &= \frac{y_i^{\lambda} - y_i^{-\lambda}}{2\lambda} \\ y_i^* &= \frac{y_i^{\lambda} - y_i^{-\lambda}}{2\lambda} \\ 2\lambda y_i^* &= y_i^{\lambda} - y_i^{-\lambda} \\ 2\lambda y_i^* &= y_i^{\lambda} - \frac{1}{y_i^{\lambda}} \\ 2\lambda y_i^* &= \frac{y_i^{2\lambda} - 1}{y_i^{\lambda}} \\ 2\lambda y_i^* y_i^{\lambda} &= y_i^{2\lambda} - 1 \\ 1 + \lambda^2 y_i^{*2} &= y_i^{2\lambda} - 2\lambda y_i^* y_i^{\lambda} + \lambda^2 y_i^{*2} \\ 1 + \lambda^2 y_i^{*2} &= (y_i^{\lambda} - \lambda y_i^*)^2 \\ \sqrt{1 + \lambda^2 y_i^{*2}} + \lambda y_i^* &= y_i^{\lambda} \\ y_i &= \left[\sqrt{1 + \lambda^2 y_i^{*2}} + \lambda y_i^*\right]^{\frac{1}{\lambda}} \\ \Rightarrow f^{-1}(y_i^*) &= \left[\sqrt{1 + \lambda^2 y_i^{*2}} + \lambda y_i^*\right]^{\frac{1}{\lambda}}. \end{aligned}$$

Dual power transformation case (B)

This case is exactly equal to the Box-Cox (shift) transformation, case (B).

Gpower transformation

$$y_i^*(\lambda) = \begin{cases} \frac{\left(y_i + \sqrt{y_i^2+1}\right)^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0 \quad (A); \\ \log\left(y_i + \sqrt{y_i^2+1}\right) & \text{if } \lambda = 0 \quad (B). \end{cases}$$

Gpower transformation case (A)

Let $J(\lambda, y)$ denote the Jacobian of a transformation from y_i to $y_i^*(\lambda)$. In order to obtain $z_i^*(\lambda)$, the scaled gpower(A) transformation, given by $\frac{y_i^*(\lambda)}{J(\lambda,y)^{1/n}}$, and for simplicity, we use a modification of the

definition of the geometric mean, denoted by $\bar{y}_{\text{GA}}$. Therefore, the Jacobian, the scaled, and the inverse of the gpower(A) transformation are given below.

The gpower(A) transformation presented in Table 3 is defined as:

$$y_i^*(\lambda) = \frac{\left[y_i + \sqrt{y_i^2+1}\right]^{\lambda} - 1}{\lambda} \text{ if } \lambda \neq 0.$$

The modification of the definition of the geometric mean for this transformation is:

$$\bar{y}_{GA} = \left[\prod_{i=1}^{n}\left(y_i + \sqrt{y_i^2+1}\right)^{\lambda-1}\left(1 + \frac{y_i}{\sqrt{y_i^2+1}}\right)\right]^{\frac{1}{n}}.$$

Therefore, the expression of the Jacobian comes to:

$$\begin{aligned} J(\lambda, \boldsymbol{y}) &= \prod_{i=1}^{n} \frac{dy_i^*(\lambda)}{dy} \\ &= \prod_{i=1}^{n} \frac{\lambda\left(y_i + \sqrt{y_i^2+1}\right)^{\lambda-1}\left(1 + \frac{2y_i}{2\sqrt{y_i^2+1}}\right)}{\lambda} \\ &= \bar{y}_{GA}^{n}. \end{aligned}$$

The scaled transformation is given by:

$$z_i^*(\lambda) = \frac{\left[y_i + \sqrt{y_i^2+1}\right]^{\lambda} - 1}{\lambda} \frac{1}{\bar{y}_{GA}}.$$

The inverse function of the gpower(A) transformation is denoted as:

$$\begin{aligned} f(y_i) &= \frac{\left[y_i + \sqrt{y_i^2+1}\right]^{\lambda} - 1}{\lambda} \\ y_i^* &= \frac{\left[y_i + \sqrt{y_i^2+1}\right]^{\lambda} - 1}{\lambda} \\ \lambda y_i^* + 1 &= \left[y_i + \sqrt{y_i^2+1}\right]^{\lambda} \\ (\lambda y_i^* + 1)^{\frac{1}{\lambda}} &= y_i + \sqrt{y_i^2+1} \\ \left[(\lambda y_i^* + 1)^{\frac{1}{\lambda}} - y_i\right]^2 &= \left[\sqrt{y_i^2+1}\right]^2 \\ (\lambda y_i^* + 1)^{\frac{2}{\lambda}} - 2y_i(\lambda y_i^* + 1)^{\frac{1}{\lambda}} + y_i^2 &= y_i^2 + 1 \\ -y_i(\lambda y_i^* + 1)^{\frac{1}{\lambda}} &= \frac{1 - (\lambda y_i^* + 1)^{\frac{2}{\lambda}}}{2} \\ y_i &= -\left[\frac{1 - (\lambda y_i^* + 1)^{\frac{2}{\lambda}}}{2(\lambda y_i^* + 1)^{\frac{1}{\lambda}}}\right] \\ \Rightarrow f^{-1}(y_i) &= -\left[\frac{1 - (\lambda y_i + 1)^{\frac{2}{\lambda}}}{2(\lambda y_i + 1)^{\frac{1}{\lambda}}}\right]. \end{aligned}$$

Gpower transformation case (B)

This case is exactly equal to the glog transformation case.

Bibliography

T. W. Anderson and D. A. Darling. A test of goodness of fit. *Journal of the American Statistical Association*, 49(268):765–769, 1954. URL https://www.jstor.org/stable/2281537. [p102]

M. S. Bartlett. The use of transformations. *Biometrics*, 3(1):39–52, 1947. URL http://dx.doi.org/10.2307/3001536. [p101]

W. D. Berry. *Understanding Regression Assumptions*. SAGE Publications, Thousand Oaks, 1993. URL https://dx.doi.org/10.4135/9781412986427. [p100]

P. J. Bickel and K. A. Doksum. An analysis of transformations revisited. *Journal of the American Statistical Association*, 76(374):296–311, 1981. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1981.10477649. [p100, 101, 102]

G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society B*, 26 (2):211–252, 1964. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.321.3819. [p99, 100, 101, 102]

T. Boylan, M. Cuddy, and I. O'Muircheartaigh. Import demand equations: An application of a generalized Box-Cox methodology. *International Statistical Review*, 50(1):103–112, 1982. URL https://www.jstor.org/stable/1402461. [p102]

T. S. Breusch and A. R. Pagan. A simple test for heteroscedasticity and random coefficient variation. *Econometrica*, 47(5):1287–1294, 1979. URL https://www.jstor.org/stable/1911963. [p102]

R. J. Carroll and D. Ruppert. Diagnostics and robust estimation when transforming the regression model and the response. *Technometrics*, 29(3):287–299, 1987. URL https://www.tandfonline.com/doi/abs/10.1080/00401706.1987.10488239. [p101]

H. Cramér. On the composition of elementary errors. *Scandinavian Actuarial Journal*, 1928(1):13–74, 1928. URL https://doi.org/10.1080/03461238.1928.10416862. [p101]

Y. Croissant. *Ecdat: Data Sets for Econometrics*, 2016. URL https://CRAN.R-project.org/package=Ecdat. R package version 0.3-1. [p103]

O. Dag, O. Asar, and O. Ilk. *AID: Box-Cox Power Transformation*, 2017. URL https://CRAN.R-project.org/package=AID. R package version 2.3. [p99]

B. P. Durbin, J. S. Hardin, D. M. Hawkins, and D. M. Rocke. A variance-stabilizing transformation for gene-expression microarray data. *Bioinformatics*, 18(1):105–110, 2002. URL https://doi.org/10.1093/bioinformatics/18.suppl_1.S105. [p99, 100, 101, 103]

Q. Feng, J. Hannig, and J. S. Marron. A note on automatic data transformation. *Stat*, 5(1):82–87, 2016. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sta4.104. [p101]

E. S. Fernandez. *Johnson: Johnson Transformation*, 2014. URL https://CRAN.R-project.org/package=Johnson. R package version 1.4. [p99]

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. SAGE Publications, Thousand Oaks, 2011. [p99, 102]

F. Hernandez and R. A. Johnson. The large-sample behavior of transformations to normality. *Journal of the American Statistical Association*, 75(372):855–861, 1980. URL http://www.jstor.org/stable/2287172. [p101]

M. Z. Hossain. The use of Box-Cox transformation technique in economic and statistical analyses. *Journal of Emerging Trends in Economics and Management Sciences*, 2(1):32–39, 2011. URL https://journals.co.za/content/sl_jetems/2/1/EJC133850. [p99]

T. Hothorn, L. Möst, and P. Bühlmann. Most likely transformations. *Scandinavian Journal of Statistics*, 45(1):110–134, 2018. URL https://doi.org/10.1111/sjos.12291. [p99]

W. Huber, A. Von Heydebreck, H. Sültmann, A. Poustka, and M. Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18(1):96–104, 2002. URL https://doi.org/10.1093/bioinformatics/18.suppl_1.S96. [p101]

W. Huber, A. von Heydebreck, H. Sültmann, A. Poustka, and M. Vingron. Parameter estimation for the calibration and variance stabilization of microarray data. *Statistical Applications in Genetics and Molecular Biology*, 2(1):1–24, 2003. URL https://doi.org/10.2202/1544-6115.1008. [p101, 102]

J. A. John and N. R. Draper. An alternative family of transformations. *Journal of the Royal Statistical Society C*, 29(2):190–197, 1980. URL https://www.jstor.org/stable/2986305. [p100, 101]

N. L. Johnson. Systems of frequency curves generated by methods of translation. *Biometrika*, 36(1/2): 149–176, 1949. URL http://www.jstor.org/stable/2332539. [p99]

D. M. Kelmansky, E. J. Martínez, and V. Leiva. A new variance stabilizing transformation for gene expression data analysis. *Statistical Applications in Genetics and Molecular Biology*, 12(6):653–666, 2013. URL https://doi.org/10.1515/sagmb-2012-0030. [p101, 103]

A. Kolmogorov. Sulla determinazione empirica di una legge di distributione. *Giornale dell'Istituto Italiano degli Attuar*, 4(1):1–11, 1933. [p101]

M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5): 1–26, 2008. URL https://www.jstatsoft.org/article/view/v028i05. [p99]

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22 (1):79–86, 1951. URL https://www.jstor.org/stable/2236703. [p101]

S. Mangiafico. *Rcompanion: Functions to Support Extension Education Program Evaluation*, 2019. URL https://CRAN.R-project.org/package=rcompanion. R package version 3.3. [p99]

B. F. J. Manly. Exponential data transformations. *Journal of the Royal Statistical Society D*, 25(1):37–42, 1976. URL https://www.jstor.org/stable/2988129. [p99, 101]

L. Medina, P. Castro, A.-K. Kreutzmann, and N. Rojas-Perilla. *Trafo: Estimation, Comparison and Selection of Transformations*, 2018. URL https://CRAN.R-project.org/package=trafo. R package version 1.0.0. [p99, 101]

I. Molina and Y. Marhuenda. Sae: An R package for small area estimation. *The R Journal*, 7(1):81–98, 2015. URL https://journal.r-project.org/archive/2015/RJ-2015-007. [p99]

M. Morozova, K. Koschutnig, E. Klein, and G. Wood. Monotonic non-linear transformations as a tool to investigate age-related effects on brain white matter integrity: A Box-Cox investigation. *NeuroImage*, 125:1119–1130, 2016. URL https://doi.org/10.1016/j.neuroimage.2015.08.003. [p99]

M. Mosimann, L. Frossard, M. Keiler, R. Weingartner, and A. Zischg. A robust and transferable model for the prediction of flood losses on household contents. *Multidisciplinary Digital Publishing Institute*, 10(11):1596, 2018. URL https://doi.org/10.3390/w10111596. [p102]

J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society A*, 135(3):370–384, 1972. URL https://www.jstor.org/stable/2344614. [p100]

J. Pek, O. Wong, and C. M. Wong. Data transformations for inference with linear regression: Clarifications and recommendations. *Practical Assessment, Research and Evaluation*, 22(9):1–11, 2017. [p102]

R. A. Peterson. *bestNormalize: Normalizing Transformation Functions*, 2019. URL https://CRAN.R-project.org/package=bestNormalize. R package version 3.1. [p99]

P. J. Ribeiro Jr. and P. J. Diggle. geoR: Analysis of geostatistical data. *R News*, 1(2):15–18, 2016. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.3971&rep=rep1&type=pdf. [p99]

D. M. Rocke and B. Durbin. A model for measurement error for gene expression arrays. *Journal of Computational Biology*, 8(6):557–569, 2001. URL https://doi.org/10.1089/106652701753307485. [p101]

N. Rojas-Perilla. *The Use of Data-Driven Transformations and Their Application in Small Area Estimation*. PhD thesis, Freie Universität Berlin, 2018. URL http://dx.doi.org/10.17169/refubium-1006. [p99, 101, 102, 108]

P. Royston, P. C. Lambert, and others. *Flexible Parametric Survival Analysis Using Stata: Beyond the Cox Model*. Stata Press, College Station, 2011. [p100, 101, 102]

S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality. *Biometrika*, 52(3/4):591–611, 1965. URL https://www.jstor.org/stable/2333709. [p102]

H. shyong Chang. Functional forms and the demand for meat in the United States. *The Review of Economics and Statistics*, 59(3):355–359, 1977. URL https://www.jstor.org/stable/1925054. [p102]

N. Smirnov. Sur les écarts de la courbe de distribution empirique. *Recueil Mathematique (Matematiceskii Sbornik)*, 6(1):3–26, 1939. [p101]

J. M. G. Taylor. Power transformations to symmetry. *Biometrika*, 72(1):145–152, 1985. URL https://doi.org/10.1093/biomet/72.1.1450. [p100]

J. W. Tukey. On the comparative anatomy of transformations. *Ann. Math. Statist.*, 28(3), 1957. URL https://www.jstor.org/stable/2237223. [p99, 101, 108]

W. J. Vaughan, C. S. Russell, and M. Hazilla. A note on the use of travel cost models with unequal zonal populations: Comment. *Land Economics*, 58(3):400–407, 1982. URL https://www.jstor.org/stable/3145948. [p102]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, New York, 2002. URL 10.1007/978-0-387-21706-2. [p99, 102]

Y. Wang. *Jtrans: Johnson Transformation for Normality*, 2015. URL https://CRAN.R-project.org/package=jtrans. R package version 0.2.1. [p99]

J. Whittaker, C. Whitehead, and M. Somers. The neglog transformation and quantile regression for the analysis of a large credit scoring database. *Journal of the Royal Statistical Society C*, 54(4):863–878, 2005. URL doi:10.1111/j.1467-9876.2005.00520.x. [p100, 101]

Z. Yang. A modified family of power transformations. *Economics Letters*, 92(1):14–19, 2006. URL doi:10.1016/j.econlet.2006.01.011. [p101]

I.-K. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000. URL https://www.jstor.org/stable/2673623. [p99, 100, 101, 102]

P. Zarembka. Functional form in the demand for money. *Journal of the American Statistical Association*, 63(322):502–511, 1968. URL https://www.jstor.org/stable/2284021. [p102]

P. Zarembka. Transformation of variables in econometrics. In P. Macmillan, editor, *The New Palgrave Dictionary of Economics*, pages 1–3. Palgrave Macmillan, London, 1974. URL https://doi.org/10.1057/978-1-349-95121-5_1882-1. [p102]

R. F. Ziemer, W. N. Musser, and R. C. Hill. Recreation demand equations: Functional form and consumer surplus. *American Journal of Agricultural Economics*, 62(1):136–141, 1980. URL https://www.jstor.org/stable/1239482. [p102]

Lily Medina
Freie Universität Berlin
Garystraße 21, 14195 Berlin
Germany
lilymiru@gmail.com

Ann-Kristin Kreutzmann
Freie Universität Berlin
Garystraße 21, 14195 Berlin
Germany
Ann-Kristin.Kreutzmann@fu-berlin.de

Natalia Rojas-Perilla
Freie Universität Berlin
Garystraße 21, 14195 Berlin
Germany
natalia.rojas@fu-berlin.de

Piedad Castro
Freie Universität Berlin
Garystraße 21, 14195 Berlin
Germany
piedad@madmath.co

BondValuation: An R Package for Fixed Coupon Bond Analysis

by Wadim Djatschenko

Abstract The purpose of this paper is to introduce the R package **BondValuation** for the analysis of large datasets of fixed coupon bonds. The conceptual heterogeneity of fixed coupon bonds traded in the global markets imposes a high degree of complexity on their comparative analysis. Contrary to baseline fixed income theory, in practice, most bonds feature coupon period irregularities. In addition, there are a multitude of day count methods that determine the interest accrual, the cash flows and the discount factors used in bond valuation. Several R packages, *e.g.*, **fBonds**, **RQuantLib**, and **YieldCurve**, provide tools for fixed income analysis. Nevertheless, none of them is capable of evaluating bonds featuring irregular first and/or final coupon periods, and neither provides adequate coverage of day count conventions currently used in the global bond markets. The R package **BondValuation** closes this gap using the generalized valuation methodology presented in Djatschenko (2019).

Introduction

Although bond valuation using the traditional present value approach is fundamental in financial theory and practice, the R community lacks applications that comprehensively handle the peculiarities of real-world fixed coupon bonds. A possible reason for the slow development of adequate computation tools concerns the matter's theoretical intricacy, characterized by a complex interaction of day count conventions (`DCC`) and irregularities in the temporal structure of the fixed income instruments.

A day count convention is an instrument-specific set of rules that prescribes the way in which calendar dates are converted to numerical values. Thus, given a schedule of a bond's anniversary dates (*i.e.*, issue date, coupon payment dates, maturity date), a day count convention is used, *e.g.*, to determine the fraction of regular coupon periods between two calendar dates within the bond's life. Irregular first and final coupon periods occur irrespective of the stipulated day count convention. The lengths of the first and final coupon periods are measured in fractions of regular coupon periods and calculated according to the rules of the specified day count convention. A fist or final coupon period is irregular, if its length differs from 1, which is the length of a regular coupon period.[1]

The R package **RQuantLib** (Eddelbuettel et al., 2018) provides access to parts of *QuantLib* (QuantLib Team, 2018), which is the leading open-source software library for quantitative finance. Currently, *QuantLib* incorporates methods for the analysis and valuation of a wide variety of financial instruments, such as options, swaps, various financial derivatives, and several types of bonds, including fixed rate bonds. Nevertheless, *QuantLib* does not implement methods for handling irregular coupon periods, and the coverage of `DCC`s is not exhaustive with nine different conventions.

A closer examination of bond market data reveals the importance of this problem. According to the Thomson Reuters EIKON database, 99.66% of the plain vanilla fixed coupon bonds that were issued worldwide in 2017 are spread over 15 different `DCC`s, and 67% of them feature irregular first and/or final coupon periods. Given the enormous size of the global bond market, neglecting irregular coupon periods potentially leads to cash flow miscalculations in the tens of billions of US dollars, as Djatschenko (2019) points out.

Essentially, `DCC`s influence bond valuation in three places. First, the amounts of interest payable at the end of any irregular coupon period are computed according to the respective convention. Second, the powers of the discount factors used in present value calculations depend on the stipulated `DCC`. Finally, in contrast to stocks, the full prices of bonds are usually not directly observable but need to be calculated as the sum of the quoted clean price and accrued interest, which is paid by the buyer to the seller if the transaction is conducted between two coupon payment dates. Accrued interest is computed conformal to the stipulated bond- and market-specific `DCC`s.

Djatschenko (2019) addresses these three aspects and proposes a generalized valuation methodology for fixed coupon bonds that allows for irregular first and final coupon periods and is compatible with any conceivable `DCC`. In summary, the methodology can be described as follows. In a first step, Djatschenko (2019) introduces a standardized bond-specific temporal structure, which is determined by the stipulated `DCC`. Based on this time structure, a valuation formula is derived that allows for first and final coupon periods of any lengths. The novelty of this proposed evaluation formula lies

[1] Djatschenko (2019) provides a comprehensive overview of most day count conventions currently used in the global bond markets and demonstrates their interactions with irregular coupon periods.

in the isolation of each DCC-dependent parameter, resulting in a modular structure that can easily integrate any conceivable DCC. In addition, Djatschenko (2019) presents closed-form solutions for the valuation formula's first and second derivatives, which are useful in the Newton-Raphson based determination of the bond's yield as well as in calculation of duration and convexity. The approach outlined in Djatschenko (2019) relies exclusively on information that is typically provided by financial data vendors and is seamlessly implemented in the R package **BondValuation**.

The remainder of this paper consists of the two main sections, "The **BondValuation** package" and "Application of the package **BondValuation**". The section entitled "The **BondValuation** package" provides an overview of the functions implemented in the R package **BondValuation** and briefly illustrates the underlying theoretical concepts. The subsection entitled "Day count conventions" introduces the DCCs covered by **BondValuation** and demonstrates their impact on interest accrual using the function `AccrInt()`. Subsequently, the Bond-specific temporal structure and its implementation within the function `AnnivDates()` are illustrated. In the following subsection, the calculation of Cash flows, accrued interest, and dirty price is demonstrated using the functions `AnnivDates()` and `DP()`. Next, the functions `BondVal.Yield()` and `BondVal.Price()` are used to compute Yield to maturity, duration, and convexity. The section entitled "Application of the package **BondValuation**" demonstrates how the R package **BondValuation** can be used for the analysis of large data frames of fixed coupon bonds. The paper ends with a short "Conclusion".

The BondValuation package

The R package **BondValuation** consists of five functions, `AccrInt()`, `AnnivDates()`, `BondVal.Price()`, `BondVal.Yield()`, and `DP()`, and four data frames, `List.DCC`, `NonBusDays.Brazil`, `PanelSomeBonds2016`, `SomeBonds2016`.

The workhorse function of the package, `AnnivDates()`, performs a variety of sanity checks on the input data and, if possible, automatically corrects corrupted entries. It determines the bond-specific temporal structure and cash flows. The output of `AnnivDates()` is used in the downstream processes of the functions `BondVal.Price()`, `BondVal.Yield()`, and `DP()`. While `AnnivDates()`, `BondVal.Price()`, `BondVal.Yield()`, and `DP()` require bond data as input, the function `AccrInt()` simply computes the amount of interst accruing from some start date to some end date.

The data frames `PanelSomeBonds2016` and `SomeBonds2016` provide simulated data of 100 plain vanilla fixed coupon corporate bonds issued in 2016. `List.DCC` provides an overview of the DCCs implemented in the R package **BondValuation**. `NonBusDays.Brazil` is used with the *BusDay/252 (Brazilian)* convention and contains all non-business days in Brazil from 1946-01-01 to 2299-12-31 based on the Brazilian national holiday calendar.

Day count conventions

All DCCs that are identified by Thomson Reuters EIKON for plain vanilla fixed coupon bonds in 2017, and, additionally, the *30E/360 (ISDA)* method, are covered by the R package **BondValuation**[2]:

```
> # example 1
> library(BondValuation)
> print(List.DCC, row.names = FALSE)
DCC                DCC.Name                                                DCC.Reference
  1          ACT/ACT (ISDA)                ISDA (1998); ISDA (2006) section 4.16 (b)
  2          ACT/ACT (ICMA)              ICMA Rule 251; ISDA (2006) section 4.16 (c)
  3           ACT/ACT (AFB)                       ISDA (1998); EBF (2004); SWX (2003)
  4                ACT/365L                                  ICMA Rule 251; SWX (2003)
  5                  30/360     ISDA (2006) section 4.16 (f); MSRB (2017) Rule G-33
  6                 30E/360 ICMA Rule 251; ISDA (2006) section 4.16 (g); SWX (2003)
  7          30E/360 (ISDA)                               ISDA (2006) section 4.16 (h)
  8         30/360 (German)                                     EBF (2004); SWX (2003)
  9              30/360 US                                   Mayle (1993); SWX (2003)
 10         ACT/365 (Fixed)                ISDA (2006) section 4.16 (d); SWX (2003)
 11             ACT(NL)/365                     Krgin (2002); Thomson Reuters EIKON
 12                 ACT/360                ISDA (2006) section 4.16 (e); SWX (2003)
 13                  30/365                     Krgin (2002); Thomson Reuters EIKON
 14 ACT/365 (Canadian Bond)                      IIAC (2018); Thomson Reuters EIKON
 15                 ACT/364                                   Thomson Reuters EIKON
 16  BusDay/252 (Brazilian)   Caputo Silva et al. (2010), Itau Unibanco S.A. (2017)
```

[2] Djatschenko (2019) provides a comprehensive overview of these DCCs.

The function `AccrInt()` can be used to compare the differences in interest accrual between the day count methods. As an example, the code below returns the number of days and the amount of interest (in percent of the bond's par value) accrued from `Start = 2011-08-31` to `End = 2012-02-29` with different `DCC`s, ceteris paribus. In this example, we assume that `CpY = 2` coupons are paid per year, the nominal interest rate p.a. is `Coup = 5.25%`, the bond is redeemed at `RV = 100%` of its par value, and payments follow the End-of-Month rule[3], *i.e.*, `EOM = 1`. In addition, some of the `DCC`s require specification of the next coupon payment's year figure, `YearNCP = 2012`, and the maturity date, `Mat = 2021-08-31`.

```
> # example 2
> library(BondValuation)
> DCC_Comparison<-data.frame(Start = rep(as.Date("2011-08-31"), 16),
+                            End = rep(as.Date("2012-02-29"), 16),
+                            Coup = rep(5.25, 16),
+                            DCC = seq(1, 16),
+                            DCC.Name = List.DCC[, 2],
+                            RV = rep(100, 16),
+                            CpY = rep(2, 16),
+                            Mat = rep(as.Date("2021-08-31"), 16),
+                            YearNCP = rep(2012, 16),
+                            EOM = rep(1, 16))
> AccrIntOutput <- suppressWarnings(
+   apply(
+     DCC_Comparison[, c('Start', 'End', 'Coup', 'DCC', 'RV', 'CpY', 'Mat',
+                        'YearNCP', 'EOM')], 1,
+     function(y) AccrInt(y[1], y[2], y[3], y[4], y[5], y[6], y[7], y[8], y[9])
+   )
+ )
> Accrued_Interest <- do.call(rbind, lapply(AccrIntOutput, function(x) x[[1]]))
> Days_Accrued <- do.call(rbind, lapply(AccrIntOutput, function(x) x[[2]]))
> DCC_Comparison <- cbind(DCC_Comparison, Accrued_Interest, Days_Accrued)
> print(DCC_Comparison[, c('DCC.Name', 'Start', 'End', 'Days_Accrued',
+                          'Accrued_Interest')], row.names = FALSE)
              DCC.Name      Start        End  Days_Accrued  Accrued_Interest
         ACT/ACT (ISDA) 2011-08-31 2012-02-29           182          2.615490
         ACT/ACT (ICMA) 2011-08-31 2012-02-29           182          2.625000
          ACT/ACT (AFB) 2011-08-31 2012-02-29           182          2.617808
               ACT/365L 2011-08-31 2012-02-29           182          2.610656
                 30/360 2011-08-31 2012-02-29           179          2.610417
                30E/360 2011-08-31 2012-02-29           179          2.610417
         30E/360 (ISDA) 2011-08-31 2012-02-29           180          2.625000
        30/360 (German) 2011-08-31 2012-02-29           180          2.625000
              30/360 US 2011-08-31 2012-02-29           179          2.610417
        ACT/365 (Fixed) 2011-08-31 2012-02-29           182          2.617808
            ACT(NL)/365 2011-08-31 2012-02-29           182          2.617808
                ACT/360 2011-08-31 2012-02-29           182          2.654167
                 30/365 2011-08-31 2012-02-29           179          2.574658
ACT/365 (Canadian Bond) 2011-08-31 2012-02-29           182          2.625000
                ACT/364 2011-08-31 2012-02-29           182          2.625000
 BusDay/252 (Brazilian) 2011-08-31 2012-02-29           124          2.549769
```

Bond-specific temporal structure

The function `AnnivDates()` evaluates bond-specific information and returns the bond's time-invariant characteristics in the data frame `Traits`, the bond's temporal structure in the data frame `DateVectors` and, if the nominal interest rate is passed, the bond's cash flows in the data frame `PaySched`.[4] The classes and formats of input data are checked and adjusted, if possible. Moreover, `AnnivDates()` performs several plausibility tests, *e.g.*, whether the provided calendar dates are in a correct chronological order and whether there are inconsistencies among the provided parameters. The results of these sanity checks are reported in the data frame `Warnings`.

[3] See manual to the R package **BondValuation** for details on implementation and Krgin (2002) for the theoretical background of the End-of-Month rule.

[4] See Djatschenko (2019) for theoretical background.

The minimum accepted input for `AnnivDates()` are two calendar dates, of which the first is interpreted as the bond's issue date and the second as its maturity date. If, as illustrated below, only two dates are passed to `AnnivDates()`, several parameters take on default values, which are reported in warning messages.

```
> # example 3
> library(BondValuation)
> AnnivDates(as.Date("2019-05-31"), "2021-07-31")
$`Warnings`
Em_FIAD_differ    EmMatMissing    CpYOverride   RV_set100percent     NegLifeFlag
             0               0              1                  1               0
      ZeroFlag   Em_Mat_SameMY ChronErrorFlag    FIPD_LIPD_equal IPD_CpY_Corrupt
             0               0              0                  0               0
 EOM_Deviation     EOMOverride    DCCOverride            NoCoups
             0               1              1                  0
$Traits
DateOrigin           CpY           FIAD           Em      Em_Orig          FIPD
1970-01-01             2           <NA>   2019-05-31   2019-05-31          <NA>
 FIPD_Orig      est_FIPD           LIPD    LIPD_Orig     est_LIPD           Mat
      <NA>    2019-07-31           <NA>         <NA>   2021-01-31    2021-07-31
     Refer       FCPType      FCPLength      LCPType    LCPLength           Par
2021-07-31         short      0.3370166      regular            1           100
CouponInPercent.p.a  DayCountConvention     EOM_Orig      est_EOM      EOM_used
                 NA                   2           NA            1             1

$DateVectors
 RealDates   RD_indexes    CoupDates   CD_indexes   AnnivDates   AD_indexes
2019-05-31    0.6629834   2019-07-31            1   2019-01-31            0
2019-07-31    1.0000000   2020-01-31            2   2019-07-31            1
2020-01-31    2.0000000   2020-07-31            3   2020-01-31            2
2020-07-31    3.0000000   2021-01-31            4   2020-07-31            3
2021-01-31    4.0000000   2021-07-31            5   2021-01-31            4
2021-07-31    5.0000000         <NA>           NA   2021-07-31            5

Warning messages:
1: In InputFormatCheck(Em = Em, Mat = Mat, CpY = CpY, FIPD = FIPD,  :
   The maturity date (Mat) is supplied as a string of class "character" in the
   format "yyyy-mm-dd". It is converted to class "Date" using the command
   "as.Date(Mat,"%Y-%m-%d")" and processed as Mat = 2021-07-31 .
2: In AnnivDates(as.Date("2019-05-31"), "2021-07-31") :
   Number of interest payments per year (CpY) is missing or NA. CpY is set 2!
3: In AnnivDates(as.Date("2019-05-31"), "2021-07-31") :
   Redemption value (RV) is missing or NA. RV is set 100!
4: In AnnivDates(as.Date("2019-05-31"), "2021-07-31") :
   EOM was not provided or NA! EOM is set 1 .
   Note: The available calandar dates suggest that EOM = 1 .
5: In AnnivDates(as.Date("2019-05-31"), "2021-07-31") :
   The day count indentifier (DCC) is missing or NA. DCC is set 2 (Act/Act (ICMA))!
```

Since neither the first nor the penultimate coupon payment date is passed to `AnnivDates()`, the calendar dates in the data frame `DateVectors` are constructed backwards starting from the maturity date 2021-07-31. This results in a bond with a short first coupon period having a length of `$Traits$FCPLength = 0.3370166` regular coupon periods.

The data frame `DateVectors` contains three vectors of calendar dates, `RealDates`, `CoupDates`, and `AnnivDates`, and their corresponding indexes, `RD_indexes`, `CD_indexes`, and `AD_indexes`. The vector `RealDates` comprises the bond's issue date, maturity date, and all coupon payment dates in between, while `CoupDates` contains only the coupon payment dates. The vector `AnnivDates` consists of the bond's so-called anniversary dates, *i.e.*, scheduled coupon dates and notional coupon dates located before the first and after the penultimate coupon payment dates. The lengths of the first (`$Traits$FCPLength`) and final coupon periods (`$Traits$LCPLength`) are calculated as differences between the coresponding values of the vectors `AD_indexes` and `RD_indexes`. `RD_indexes` are used in the functions `BondVal.Price()`, `BondVal.Yield()`, and `DP()` to determine the powers of discount factors in pricing formulas.

As warning message 4 in the example above reports, the function `AnnivDates()` analyzes the

provided calendar dates to ascertain whether the bond follows the End-of-Month rule (EOM). An automated replacement of the provided value of EOM by the value determined by the function AnnivDates() can be activated by setting option FindEOM = TRUE, which defaults to FALSE.

The following example illustrates the effect of EOM on DateVectors. In addition to issue date (Em) and maturity date (Mat), the first coupon payment date (FIPD), the penultimate coupon payment date (LIPD), the number of coupon payments p.a. (CpY), and EOM are passed to the function AnnivDates():

```
> # example 4
> library(BondValuation)
> # example 4a: computing DateVectors for EOM = 1
> EOM.input <- 1
> AnnivDates(Em = as.Date("2019-05-31"),
+            Mat = as.Date("2021-07-31"),
+            CpY = 2,
+            FIPD = as.Date("2020-02-29"),
+            LIPD = as.Date("2021-02-28"),
+            EOM = EOM.input)$DateVectors
   RealDates   RD_indexes    CoupDates   CD_indexes   AnnivDates   AD_indexes
1 2019-05-31    -0.500000   2020-02-29     1.000000   2019-02-28           -1
2 2020-02-29     1.000000   2020-08-31     2.000000   2019-08-31            0
3 2020-08-31     2.000000   2021-02-28     3.000000   2020-02-29            1
4 2021-02-28     3.000000   2021-07-31     3.831522   2020-08-31            2
5 2021-07-31     3.831522         <NA>           NA   2021-02-28            3
6       <NA>           NA         <NA>           NA   2021-08-31            4
Warning messages:
1: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"),  :
   Redemption value (RV) is missing or NA. RV is set 100!
2: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"),  :
   The day count indentifier (DCC) is missing or NA. DCC is set 2 (Act/Act (ICMA))!
>
> # example 4b: computing DateVectors for EOM = 0
> EOM.input <- 0
> AnnivDates(Em = as.Date("2019-05-31"),
+            Mat = as.Date("2021-07-31"),
+            CpY = 2,
+            FIPD = as.Date("2020-02-29"),
+            LIPD = as.Date("2021-02-28"),
+            EOM = EOM.input)$DateVectors
   RealDates   RD_indexes    CoupDates   CD_indexes   AnnivDates   AD_indexes
1 2019-05-31   -0.4945055   2020-02-29     1.000000   2019-02-28           -1
2 2020-02-29    1.0000000   2020-08-29     2.000000   2019-08-29            0
3 2020-08-29    2.0000000   2021-02-28     3.000000   2020-02-29            1
4 2021-02-28    3.0000000   2021-07-31     3.840659   2020-08-29            2
5 2021-07-31    3.8406593         <NA>           NA   2021-02-28            3
6       <NA>           NA         <NA>           NA   2021-08-29            4
Warning messages:
1: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"),  :
   Redemption value (RV) is missing or NA. RV is set 100!
2: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"),  :
   The available calandar dates suggest that EOM = 1 .
   Option FindEOM = FALSE is active. Provided EOM is not overridden and remains
   EOM = 0 .
3: In AnnivDates(Em = as.Date("2019-05-31"), Mat = as.Date("2021-07-31"),  :
   The day count indentifier (DCC) is missing or NA. DCC is set 2 (Act/Act (ICMA))!
```

In contrast to *example 3*, the bond in *example 4* features a long first and a short final coupon period. The function AnnivDates() has checked whether the provided dates FIPD and LIPD are on each other's annivesary dates and constructed the calendar dates in DateVectors backwards and forwards from LIPD. The values of RD_indexes and AD_indexes are illustrated in Figure 1, where E corresponds to the first element of RD_indexes and M is the final element of RD_indexes.

As shown in *example 4*, all else equal, the value of EOM affects the DCC-conformal temporal locations of the issue date E and the maturity date M and, hence, the lengths of the first and final coupon periods, which, in turn, determine the amounts of interest paid on the first and final coupon payment dates. So far, no value of DCC was passed to the function AnnivDates(). As reported in the warning

messages in *example 4*, the parameter `DCC` defaults to the *Act/Act (ICMA)* convention. The following, *example 5*, illustrates how `RD_indexes`, `FCPLength` and `LCPLength` vary across `DCC`s for `EOM = 0`.

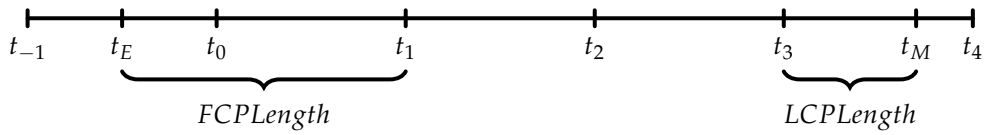


Figure 1: Timeline illustration of the bonds in examples 4 and 5.

```
> # example 5
> library(BondValuation)
> TempStruct.by.DCC <- data.frame(Em = rep(as.Date("2019-05-31"), 16),
+                                 Mat = rep(as.Date("2021-07-31"), 16),
+                                 CpY = rep(2, 16),
+                                 FIPD = rep(as.Date("2020-02-29"), 16),
+                                 LIPD = rep(as.Date("2021-02-28"), 16),
+                                 FIAD = rep(as.Date("2019-05-31"), 16),
+                                 DCC = seq(1, 16),
+                                 EOM = rep(0, 16),
+                                 DCC.Name = List.DCC[, 2])
> # Applying AnnivDates() to the data frame TempStruct.by.DCC for EOM = 0
> suppressWarnings(
+   FullAnalysis.EOM0 <- apply(
+     TempStruct.by.DCC[, c('Em','Mat','CpY','FIPD','LIPD','FIAD','DCC','EOM')],
+     1, function(y) AnnivDates(
+       y[1], y[2], y[3], y[4], y[5], y[6], , , y[7], y[8])
+   )
+ )
> FCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[[`, 2), `[[`, 15)
+                          , na.omit)
> FCPLength.EOM0 <- as.data.frame(do.call(rbind, lapply(FCPLength.EOM0, round, 4)))
> LCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[[`, 2), `[[`, 17)
+                          , na.omit)
> LCPLength.EOM0 <- as.data.frame(do.call(rbind, lapply(LCPLength.EOM0, round, 4)))
> TempStruct.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[[`, 3), `[[`, 2)
+                           , na.omit)
> TempStruct.EOM0 <- lapply(TempStruct.EOM0, `length<-`,
+                           max(lengths(TempStruct.EOM0)))
> TempStruct.EOM0 <- as.data.frame(do.call(rbind, lapply(TempStruct.EOM0, round, 4)))
> TempStruct.by.DCC.EOM0 <- cbind(TempStruct.by.DCC, TempStruct.EOM0,
+                                 FCPLength.EOM0, LCPLength.EOM0)
> names(TempStruct.by.DCC.EOM0)[c(10:16)] <- c("E", "01", "02", "03", "M",
+                                              "FCPLength", "LCPLength")
> print(TempStruct.by.DCC.EOM0[, c(9:ncol(TempStruct.by.DCC.EOM0))],
+       row.names = FALSE)
               DCC.Name         E   01   02   03        M  FCPLength  LCPLength
         ACT/ACT (ISDA)   -0.4945    1    2    3   3.8407     1.4945     0.8407
         ACT/ACT (ICMA)   -0.4945    1    2    3   3.8407     1.4945     0.8407
          ACT/ACT (AFB)   -0.4945    1    2    3   3.8407     1.4945     0.8407
               ACT/365L   -0.4945    1    2    3   3.8407     1.4945     0.8407
                 30/360   -0.4862    1    2    3   3.8453     1.4862     0.8453
                30E/360   -0.4917    1    2    3   3.8398     1.4917     0.8398
         30E/360 (ISDA)   -0.4972    1    2    3   3.8380     1.4972     0.8380
        30/360 (German)   -0.4972    1    2    3   3.8380     1.4972     0.8380
              30/360 US   -0.4862    1    2    3   3.8453     1.4862     0.8453
        ACT/365 (Fixed)   -0.4945    1    2    3   3.8407     1.4945     0.8407
            ACT(NL)/365   -0.4945    1    2    3   3.8407     1.4945     0.8407
                ACT/360   -0.4945    1    2    3   3.8407     1.4945     0.8407
                 30/365   -0.4862    1    2    3   3.8453     1.4862     0.8453
ACT/365 (Canadian Bond)   -0.4945    1    2    3   3.8407     1.4945     0.8407
                ACT/364   -0.4945    1    2    3   3.8407     1.4945     0.8407
 BusDay/252 (Brazilian)   -0.5040    1    2    3   3.8425     1.5040     0.8425
```

The output of *example 5* shows that for the specified bond, there is little variation in temporal structure across the DCCs. Specifically, with DCC $\in \{1, 2, 3, 4, 10, 11, 12, 14, 15\}$, the values of RD_indexes are $\{-0.4945, 1, 2, 3, 3.8407\}$; with DCC $\in \{5, 9, 13\}$, it holds RD_indexes $= \{-0.4862, 1, 2, 3, 3.8453\}$, and with DCC $\in \{7, 8\}$, we get RD_indexes $= \{-0.4972, 1, 2, 3, 3.8380\}$. Only the DCCs 6 and 16 produce a unique temporal structure for this bond. Nevertheless, it would be wrong to infer from this that the temporal structure always has so little variation across the DCCs, as *example 6* illustrates.

```
> # example 6
> library(BondValuation)
> TempStruct.by.DCC <- data.frame(Em = rep(as.Date("2019-10-31"), 16),
+                                 Mat = rep(as.Date("2024-02-29"), 16),
+                                 CpY = rep(2, 16),
+                                 FIPD = rep(as.Date("2020-03-30"), 16),
+                                 LIPD = rep(as.Date("2023-03-30"), 16),
+                                 FIAD = rep(as.Date("2019-10-31"), 16),
+                                 DCC = seq(1, 16),
+                                 EOM = rep(0, 16),
+                                 DCC.Name = List.DCC[, 2])
>
> # Applying AnnivDates() to the data frame TempStruct.by.DCC for EOM = 0
> suppressWarnings(
+   FullAnalysis.EOM0 <- apply(
+     TempStruct.by.DCC[, c('Em','Mat','CpY','FIPD','LIPD','FIAD','DCC','EOM')],
+     1, function(y) AnnivDates(
+       y[1], y[2], y[3], y[4], y[5], y[6], , , y[7], y[8])
+   )
+ )
> FCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[[`, 2), `[[`, 15)
+                          , na.omit)
> FCPLength.EOM0 <- as.data.frame(do.call(rbind, lapply(FCPLength.EOM0, round, 4)))
> LCPLength.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[[`, 2), `[[`, 17)
+                          , na.omit)
> LCPLength.EOM0 <- as.data.frame(do.call(rbind, lapply(LCPLength.EOM0, round, 4)))
> TempStruct.EOM0 <- lapply(lapply(lapply(FullAnalysis.EOM0, `[[`, 3), `[[`, 2)
+                           , na.omit)
> TempStruct.EOM0 <- lapply(TempStruct.EOM0, `length<-`,
+                           max(lengths(TempStruct.EOM0)))
> TempStruct.EOM0 <- as.data.frame(do.call(rbind, lapply(TempStruct.EOM0, round, 4)))
> TempStruct.by.DCC.EOM0 <- cbind(TempStruct.by.DCC, TempStruct.EOM0,
+                                 FCPLength.EOM0, LCPLength.EOM0)
> names(TempStruct.by.DCC.EOM0)[c(10:20)] <- c("E","01","02","03","04","05","06",
+                                              "07","M","FCPLength","LCPLength")
> print(TempStruct.by.DCC.EOM0[, c(9:ncol(TempStruct.by.DCC.EOM0))],
+       row.names = FALSE)
                DCC.Name       E 01 02 03 04 05 06 07      M FCPLength LCPLength
          ACT/ACT (ISDA)  0.1706  1  2  3  4  5  6  7 8.8354    0.8294    1.8354
          ACT/ACT (ICMA)  0.1703  1  2  3  4  5  6  7 8.8352    0.8297    1.8352
           ACT/ACT (AFB)  0.1708  1  2  3  4  5  6  7 8.8375    0.8292    1.8375
                ACT/365L  0.1703  1  2  3  4  5  6  7 8.8352    0.8297    1.8352
                  30/360  0.1667  1  2  3  4  5  6  7 8.8278    0.8333    1.8278
                 30E/360  0.1667  1  2  3  4  5  6  7 8.8278    0.8333    1.8278
          30E/360 (ISDA)  0.1667  1  2  3  4  5  6  7 8.8278    0.8333    1.8278
         30/360 (German)  0.1667  1  2  3  4  5  6  7 8.8333    0.8333    1.8333
               30/360 US  0.1667  1  2  3  4  5  6  7 8.8278    0.8333    1.8278
         ACT/365 (Fixed)  0.1703  1  2  3  4  5  6  7 8.8352    0.8297    1.8352
             ACT(NL)/365  0.1713  1  2  3  4  5  6  7 8.8398    0.8287    1.8398
                 ACT/360  0.1703  1  2  3  4  5  6  7 8.8352    0.8297    1.8352
                  30/365  0.1667  1  2  3  4  5  6  7 8.8278    0.8333    1.8278
ACT/365 (Canadian Bond)  0.1703  1  2  3  4  5  6  7 8.8352    0.8297    1.8352
                 ACT/364  0.1703  1  2  3  4  5  6  7 8.8352    0.8297    1.8352
 BusDay/252 (Brazilian)  0.1840  1  2  3  4  5  6  7 8.8279    0.8160    1.8279
```

The output of *example 6* reveals that, all else equal, the specified bond can feature 7 different temporal structures, depending on the stipulated DCC. While in *example 5* the "ACT/ACT" family of DCCs produced the same temporal structure, in *example 6* most of the "30/360" DCCs result in the same day count.

Cash flows, accrued interest, and dirty price

In the preceding examples of `AnnivDates()`, no information on nominal interest rate (`Coup`) and redemption value (`RV`) was passed to the function. If this information, however, is provided, then `AnnivDates()` generates the data frame `PaySched`, consisting of the scheduled coupon dates and the corresponding cash flows. As Djatschenko (2019) points out, precise application of the respective `DCC`'s mathematical rule results in varying interest payments. While this is intended for calculation of the cash flows paid at the ends of irregular first and final coupon periods, most issuers design their bonds to pay the same cash flow at the end of each regular period. With default `RegCF.equal = 0` the function `AnnivDates()` calculates all cash flows according to the mathematical rule of the respective `DCC`. Passing any other value to `RegCF.equal` forces all regular cash flows to be equal sized. The following, *example 7*, uses the same input as *example 6* supplemented by information on nominal interest rate p.a. (`Coup` $= 10\%$) and redemption value (`RV` $= 100\%$) and illustrates the differences in cash flows (in percent of the bond's par value) by `DCC` between the two modes of `RegCF.equal`.

```
> # example 7
> library(BondValuation)
> CashFlows.by.DCC <- data.frame(Em = rep(as.Date("2019-10-31"), 16),
+                                Mat = rep(as.Date("2024-02-29"), 16),
+                                CpY = rep(2, 16),
+                                FIPD = rep(as.Date("2020-03-30"), 16),
+                                LIPD = rep(as.Date("2023-03-30"), 16),
+                                FIAD = rep(as.Date("2019-10-31"), 16),
+                                RV = rep(100, 16),
+                                Coup = rep(10, 16),
+                                DCC = seq(1, 16),
+                                EOM = rep(0, 16),
+                                DCC.Name = List.DCC[, 2])
>
> # Applying AnnivDates() to the data frame CashFlows.by.DCC for EOM = 0
> # with option RegCF.equal = 0 and RegCF.equal = 1
> Suffix <- c("RegCFvary","RegCFequal")
> for (i in c(0,1)) {
+   suppressWarnings(
+     FullAnalysis <- apply(
+       CashFlows.by.DCC[, c('Em','Mat','CpY','FIPD','LIPD','FIAD','RV',
+                            'Coup','DCC','EOM')],
+       1, function(y) AnnivDates(
+         y[1],y[2],y[3],y[4],y[5],y[6],y[7],y[8],y[9],y[10], RegCF.equal = i)
+     )
+   )
+   CashFlows <- lapply(lapply(lapply(FullAnalysis, `[[`, 4), `[[`, 2)
+                       , na.omit)
+   CashFlows <- as.data.frame(do.call(rbind, lapply(CashFlows, round, 4)))
+   CashFlows <- cbind(CashFlows.by.DCC, CashFlows)
+   names(CashFlows)[c(12:19)] <- c(
+     "CF.1","CF.2","CF.3","CF.4","CF.5","CF.6","CF.7","CF.M")
+   assign(paste0("CashFlows.by.DCC.",Suffix[i+1]),CashFlows)
+   rm(FullAnalysis,CashFlows)
+ }
>
> # RegCF.equal = 0, \textit{i.e.}, regular cash flows may vary
> print(CashFlows.by.DCC.RegCFvary[, c(11:ncol(CashFlows.by.DCC.RegCFvary))],
+       row.names = FALSE)
               DCC.Name   CF.1   CF.2   CF.3   CF.4   CF.5   CF.6   CF.7   CF.M
         ACT/ACT (ISDA) 4.1303 5.0273 4.9519 5.0411 4.9589 5.0411 4.9589 9.2011
         ACT/ACT (ICMA) 4.1484 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1758
          ACT/ACT (AFB) 4.1257 5.0411 4.9589 5.0411 4.9589 5.0411 4.9589 9.2055
               ACT/365L 4.1257 5.0273 4.9589 5.0411 4.9589 5.0411 4.9589 9.1803
                 30/360 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
                30E/360 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
         30E/360 (ISDA) 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
        30/360 (German) 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1667
              30/360 US 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
        ACT/365 (Fixed) 4.1370 5.0411 4.9589 5.0411 4.9589 5.0411 4.9589 9.2055
```

```
           ACT(NL)/365 4.1096 5.0411 4.9589 5.0411 4.9589 5.0411 4.9589 9.2055
               ACT/360 4.1944 5.1111 5.0278 5.1111 5.0278 5.1111 5.0278 9.3333
                30/365 4.1096 4.9315 4.9315 4.9315 4.9315 4.9315 4.9315 9.0137
ACT/365 (Canadian Bond) 4.1370 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1644
               ACT/364 4.1484 5.0549 4.9725 5.0549 4.9725 5.0549 4.9725 9.2308
 BusDay/252 (Brazilian) 3.9332 4.8809 4.8809 4.8809 4.8809 4.8809 4.8809 9.0060
>
> # RegCF.equal = 1, \textit{i.e.}, regular cash flows forced to be equal
> print(CashFlows.by.DCC.RegCFequal[, c(11:ncol(CashFlows.by.DCC.RegCFequal))],
+       row.names = FALSE)
              DCC.Name   CF.1   CF.2   CF.3   CF.4   CF.5   CF.6   CF.7   CF.M
         ACT/ACT (ISDA) 4.1303 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2011
         ACT/ACT (ICMA) 4.1484 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1758
          ACT/ACT (AFB) 4.1257 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2055
               ACT/365L 4.1257 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1803
                 30/360 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
                30E/360 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
         30E/360 (ISDA) 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
        30/360 (German) 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1667
              30/360 US 4.1667 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1389
        ACT/365 (Fixed) 4.1370 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2055
            ACT(NL)/365 4.1096 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2055
                ACT/360 4.1944 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.3333
                 30/365 4.1096 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.0137
ACT/365 (Canadian Bond) 4.1370 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.1644
               ACT/364 4.1484 5.0000 5.0000 5.0000 5.0000 5.0000 5.0000 9.2308
 BusDay/252 (Brazilian) 3.9332 4.8809 4.8809 4.8809 4.8809 4.8809 4.8809 9.0060
```

Please note that, irrespective of the value of `RegCF.equal` passed to `AnnivDates()`, the cash flows at the ends of all regular coupon periods are equal sized with the conventions *ACT/ACT (ICMA)*, *ACT/365 (Canadian Bond)*, and *BusDay/252 (Brazilian)*. This is due to the `DCC`-specific rules described in Djatschenko (2019). While with the majority of `DCC`s, the cash flows are computed based upon the ratio of the nominal interest rate p.a. and the number of interest payments per year, which yields regular cash flows of 5%, *BusDay/252 (Brazilian)* determines them exponentially, resulting in regular cash flows of 4.8809%.

The vast majority of bonds are quoted clean, *i.e.*, their observable prices do not contain accrued interest. The actual price that a bond buyer pays to the seller is called full or dirty price and computed as the sum of the quoted clean price and accrued interest, which is calculated according to the respective `DCC`. Accrued interest and the dirty price of a specific bond can be calculated using the function `DP()`. In addition to the input parameters required by `AnnivDates()`, the clean price (`CP`) and the settlement date (`SETT`) need to be passed to the function `DP()`. The following, *example 8*, returns the accrued interest and dirty price by `DCC` for the same bond as used in *example 7*, assuming that on the settlement dates `SETT1 = 2020-09-28`, `SETT2 = 2023-03-30`, and `SETT3 = 2024-01-15`, the quoted clean price is 105% of the bond's par value.

```
> # example 8
> library(BondValuation)
> AccrIntDP.by.DCC <- data.frame(CP = 105,
+                                SETT1 = rep(as.Date("2020-09-28"), 16),
+                                SETT2 = rep(as.Date("2023-03-30"), 16),
+                                SETT3 = rep(as.Date("2024-01-15"), 16),
+                                Em = rep(as.Date("2019-10-31"), 16),
+                                Mat = rep(as.Date("2024-02-29"), 16),
+                                CpY = rep(2, 16),
+                                FIPD = rep(as.Date("2020-03-30"), 16),
+                                LIPD = rep(as.Date("2023-03-30"), 16),
+                                FIAD = rep(as.Date("2019-10-31"), 16),
+                                RV = rep(100, 16),
+                                Coup = rep(10, 16),
+                                DCC = seq(1, 16),
+                                EOM = rep(0, 16),
+                                DCC.Name = List.DCC[, 2])
>
> Suffix <- c("SETT1","SETT2","SETT3")
> for (i in c(1:3)) {
```

```
+   DP.Output<-suppressWarnings(
+     apply(AccrIntDP.by.DCC[,c('CP',paste0('SETT',i),'Em','Mat','CpY','FIPD',
+                               'LIPD','FIAD','RV','Coup','DCC')],
+           1,function(y) DP(y[1],y[2],y[3],y[4],y[5],y[6],y[7],
+                            y[8],y[9],y[10],y[11])))
+   AI<-do.call(rbind,lapply(lapply(lapply(DP.Output, `[[`, 2), `[[`, 3), round, 4))
+   DP<-do.call(rbind,lapply(lapply(lapply(DP.Output, `[[`, 2), `[[`, 1), round, 4))
+   AccrIntDP.by.DCC<-cbind(AccrIntDP.by.DCC,AI,DP)
+   names(AccrIntDP.by.DCC)[
+     c((ncol(AccrIntDP.by.DCC) - 1) : ncol(AccrIntDP.by.DCC))] <- c(
+     paste0("AI.", Suffix[i]), paste0("DP.", Suffix[i]))
+   rm(DP.Output,AI,DP)
+ }
> print(AccrIntDP.by.DCC[,c(15:ncol(AccrIntDP.by.DCC))], row.names = FALSE)
               DCC.Name AI.SETT1 DP.SETT1 AI.SETT2 DP.SETT2 AI.SETT3 DP.SETT3
         ACT/ACT (ISDA)   4.9727 109.9727        0      105   7.9716 112.9716
         ACT/ACT (ICMA)   4.9457 109.9457        0      105   7.9396 112.9396
          ACT/ACT (AFB)   4.9863 109.9863        0      105   7.9726 112.9726
               ACT/365L   4.9727 109.9727        0      105   7.9508 112.9508
                 30/360   4.9444 109.9444        0      105   7.9167 112.9167
                 30E/360   4.9444 109.9444        0      105   7.9167 112.9167
         30E/360 (ISDA)   4.9444 109.9444        0      105   7.9167 112.9167
        30/360 (German)   4.9444 109.9444        0      105   7.9167 112.9167
              30/360 US   4.9444 109.9444        0      105   7.9167 112.9167
        ACT/365 (Fixed)   4.9863 109.9863        0      105   7.9726 112.9726
            ACT(NL)/365   4.9863 109.9863        0      105   7.9726 112.9726
                ACT/360   5.0556 110.0556        0      105   8.0833 113.0833
                 30/365   4.8767 109.8767        0      105   7.8082 112.8082
ACT/365 (Canadian Bond)   4.9863 109.9863        0      105   7.9315 112.9315
                ACT/364   5.0000 110.0000        0      105   7.9945 112.9945
 BusDay/252 (Brazilian)   4.8412 109.8412        0      105   7.7354 112.7354
```

Yield to maturity, duration, and convexity

The yield to maturity p.a. is determined as the value y that fulfills equation (1).

$$DP_\tau = CP_\tau + AC(t_\tau) = \frac{CN(t_\tau)}{\left(1+\frac{y}{h}\right)^w} + \sum_{i=1}^{\eta} \frac{CF_{i+k}}{\left(1+\frac{y}{h}\right)^{w+i}} + \frac{CF_M + RV}{\left(1+\frac{y}{h}\right)^{w+\eta+z}}. \tag{1}$$

In equation (1), DP_τ denotes the dirty price, consisting of the quoted clean price CP_τ and accrued interest $AC(t_\tau)$. Conformal with the notation in Djatschenko (2019), t_τ is the settlement date and τ its index in the temporal structure established by the function `AnnivDates()`. On the right side of equation (1), $CN(t_\tau)$ denotes the next coupon payment after the settlement date t_τ, and w is the fraction of a regular coupon period left until this payment. The set CF_{i+k} with $i \in \{x \in \mathbb{N} \mid x \in [1, \eta]\}$ contains all interest payments after t_k, excluding the final coupon payment, CF_M, where k is the index of the next coupon date after t_τ, η is the number of interest payment dates between t_τ and the penultimate coupon date, and M is the index corresponding to the bond's maturity date. RV denotes the redemption payment, z represents the length of the final coupon period, and h represents the number of regular interest payments per year.

The dirty price DP_τ and the accrued interest $AC(\tau)$ are computed as illustrated in *example 8*. The cash flows $CN(t_\tau)$, CF_{i+k}, and CF_M are calculated as demonstrated in *example 7*. The powers in the denominators in equation (1) are found based on the temporal structure established by the function `AnnivDates()`, as shown in *example 6*.

Essentially, the same `DCC` is used for computation of cash flows, accrued interest and the indexes of the temporal structure. Nevertheless, the option `Calc.Method` in the functions `BondVal.Price()` and `BondVal.Yield()` allows for switching the calculation method for the temporal structure to `DCC = 2`, *i.e.*, *ACT/ACT (ICMA)*, while keeping the `DCC` passed to the function for determination of cash flows and accrued interest.

The function `BondVal.Price()` can be used to compute a bond's clean price, (CP_τ), given its yield

to maturity p.a. (y), while the function `BondVal.Yield()` returns y given CP_τ. Besides accrued interest ($AC(t_\tau)$) and dirty price (DP_τ), both functions return τ, MacAulay duration, modified duration, and convexity of the specified bond.[5] The following, *example 9*, demonstrates the use of the function `BondVal.Yield()` for the bond analyzed in *example 8*. In the output of *example 9*, `YtM` denotes the bond's yield to maturity p.a. in percent, `DUR` is the bond's modified duration in years, and `Conv` is the bond's convexity in years. The suffixes `.S1`, `.S2`, and `.S3` correspond to the three analyzed settlement dates, `SETT1 = 2020-09-28`, `SETT2 = 2023-03-30`, and `SETT3 = 2024-01-15`. For space reasons, the first column of the displayed data frame contains the `DCC`-codes instead of their names.

```
> # example 9
> library(BondValuation)
> YtM.by.DCC <- data.frame(CP = 105,
+                          SETT1 = rep(as.Date("2020-09-28"), 16),
+                          SETT2 = rep(as.Date("2023-03-30"), 16),
+                          SETT3 = rep(as.Date("2024-01-15"), 16),
+                          Em = rep(as.Date("2019-10-31"), 16),
+                          Mat = rep(as.Date("2024-02-29"), 16),
+                          CpY = rep(2, 16),
+                          FIPD = rep(as.Date("2020-03-30"), 16),
+                          LIPD = rep(as.Date("2023-03-30"), 16),
+                          FIAD = rep(as.Date("2019-10-31"), 16),
+                          RV = rep(100, 16),
+                          Coup = rep(10, 16),
+                          DCC = seq(1, 16),
+                          EOM = rep(0, 16))
>
> Suffix <- c("S1","S2","S3")
> i<-1
> for (i in c(1:3)) {
+   BondValYield.Output<-suppressWarnings(
+     apply(YtM.by.DCC[,c('CP',paste0('SETT',i),'Em','Mat','CpY','FIPD',
+                         'LIPD','FIAD','RV','Coup','DCC')],
+           1,function(y) BondVal.Yield(y[1],y[2],y[3],y[4],y[5],y[6],y[7],
+                                       y[8],y[9],y[10],y[11])))
+
+   YtM<-do.call(rbind,lapply(lapply(BondValYield.Output, `[[`, 4), round, 3))
+   ModDUR<-do.call(rbind,lapply(lapply(BondValYield.Output, `[[`, 5), round, 4))
+   Conv<-do.call(rbind,lapply(lapply(BondValYield.Output, `[[`, 7), round, 4))
+   YtM.by.DCC<-cbind(YtM.by.DCC,YtM,ModDUR,Conv)
+   names(YtM.by.DCC)[
+     c((ncol(YtM.by.DCC) - 2) : ncol(YtM.by.DCC))] <- c(
+       paste0("YtM.", Suffix[i]), paste0("DUR.", Suffix[i]),
+       paste0("Conv.", Suffix[i]))
+   rm(BondValYield.Output,YtM,ModDUR,Conv)
+ }
> print(YtM.by.DCC[,c(13,15:ncol(YtM.by.DCC))], row.names = FALSE)
 DCC  YtM.S1  DUR.S1  Conv.S1  YtM.S2  DUR.S2  Conv.S2   YtM.S3  DUR.S3  Conv.S3
   1   8.244  2.7593   4.9777   4.360  0.8824   0.7786  -27.035  0.1277   0.0163
   2   8.252  2.7590   4.9766   4.334  0.8825   0.7788  -26.956  0.1279   0.0164
   3   8.245  2.7595   4.9793   4.360  0.8833   0.7803  -26.899  0.1282   0.0164
   4   8.238  2.7604   4.9813   4.339  0.8824   0.7787  -27.002  0.1279   0.0164
   5   8.251  2.7564   4.9676   4.313  0.8792   0.7730  -27.373  0.1265   0.0160
   6   8.251  2.7564   4.9676   4.313  0.8792   0.7730  -27.373  0.1265   0.0160
   7   8.251  2.7564   4.9676   4.313  0.8792   0.7730  -27.373  0.1265   0.0160
   8   8.252  2.7584   4.9746   4.329  0.8817   0.7774  -26.568  0.1293   0.0167
   9   8.251  2.7564   4.9676   4.313  0.8792   0.7730  -27.373  0.1265   0.0160
  10   8.248  2.7587   4.9763   4.365  0.8822   0.7784  -26.973  0.1279   0.0164
  11   8.243  2.7604   4.9824   4.354  0.8845   0.7823  -26.825  0.1286   0.0165
  12   8.381  2.7505   4.9554   4.498  0.8812   0.7765  -26.824  0.1279   0.0163
  13   8.120  2.7645   4.9882   4.183  0.8802   0.7748  -27.521  0.1265   0.0160
  14   8.236  2.7593   4.9776   4.322  0.8826   0.7789  -26.983  0.1279   0.0164
  15   8.274  2.7570   4.9722   4.391  0.8820   0.7780  -26.943  0.1279   0.0164
  16   8.032  2.7729   5.0104   4.175  0.8803   0.7750  -26.038  0.1314   0.0173
```

[5] Djatschenko (2019) provides the theoretical background on the implemented key figures.

Application of the package BondValuation

This section demonstrates how the R package **BondValuation** can be applied for the analysis of large data frames. For this purpose, the two sample data frames, `SomeBonds2016` and `PanelSomeBonds2016`, are used. `SomeBonds2016` contains time-invariant information of 100 hypothetical bonds. `PanelSomeBonds2016` provides daily clean prices and yields of the same bonds in long format.

Checking the data with AnnivDates()

Since erroneous entries in the data are often an issue, the function `AnnivDates()` performs several plausibility checks. `Example 10` provides a summary of `SomeBonds2016` and illustrates a strategy for error identification in this data frame.

```
> # example 10
> library(BondValuation)
> summary(SomeBonds2016)
    ID.No              Coup.Type            Issue.Date          FIAD.Input
Min.   :  1.00       Length:100           Min.   :2016-01-01   Min.   :2016-01-01
1st Qu.: 25.75       Class :character     1st Qu.:2016-04-25   1st Qu.:2016-04-25
Median : 50.50       Mode  :character     Median :2016-06-12   Median :2016-06-12
Mean   : 50.50                            Mean   :2016-06-19   Mean   :2016-06-19
3rd Qu.: 75.25                            3rd Qu.:2016-08-23   3rd Qu.:2016-08-23
Max.   :100.00                            Max.   :2016-10-14   Max.   :2016-10-28
  FIPD.Input           LIPD.Input           Mat.Date            CpY.Input
Min.   :2016-04-24   Min.   :2016-08-23   Min.   :2017-01-24   Min.   : 1.0
1st Qu.:2016-09-30   1st Qu.:2019-02-14   1st Qu.:2019-07-24   1st Qu.: 2.0
Median :2016-12-15   Median :2020-06-08   Median :2020-11-20   Median : 2.5
Mean   :2016-12-15   Mean   :2021-11-11   Mean   :2022-05-18   Mean   : 4.3
3rd Qu.:2017-03-02   3rd Qu.:2022-11-04   3rd Qu.:2023-05-05   3rd Qu.: 6.0
Max.   :2017-08-23   Max.   :2056-05-20   Max.   :2056-08-31   Max.   :12.0
  Coup.Input           RV.Input             DCC.Input           EOM.Input
Min.   : 0.010       Min.   :100          Min.   : 1.00        Min.   :0.00
1st Qu.: 0.800       1st Qu.:100          1st Qu.: 5.00        1st Qu.:1.00
Median : 1.410       Median :100          Median :10.00        Median :1.00
Mean   : 2.270       Mean   :100          Mean   : 8.95        Mean   :0.79
3rd Qu.: 2.869       3rd Qu.:100          3rd Qu.:13.00        3rd Qu.:1.00
Max.   :24.020       Max.   :100          Max.   :16.00        Max.   :1.00
```

The summary information above reveals that all bonds in the data frame were issued (`Issue.Date`) and started to accrue interest (`FIAD.Input`) in 2016. The terms to maturity (`Mat.Date`) span from about 1 to approximately 40 years. The summary of variable `CpY.Input` shows that there are no zero coupon bonds in the dataset and the number of interest payments per year varies from 1 to 12. Nominal interest rates (`Coup.Input`) average 2.27%, varying from 0.01% to 24.02%. All bonds are redeemed (`RV.Input`) at 100% of their respective par values and 79% of them follow the End-of-Month rule (`EOM.Input`). Now `AnnivDates()` is used to analyze the data for plausibility.

```
> # example 10: continued (I)
>
> # Applying AnnivDates() to the data frame SomeBonds2016.
> FullAnalysis<-suppressWarnings(
+   apply(
+     SomeBonds2016[,c('Issue.Date','Mat.Date','CpY.Input','FIPD.Input',
+                      'LIPD.Input','FIAD.Input','RV.Input','Coup.Input',
+                      'DCC.Input','EOM.Input')], 1,
+     function(y) AnnivDates(y[1], y[2], y[3], y[4], y[5], y[6], y[7],
+                            y[8], y[9], y[10])
+   )
+ )
> # Extracting the data frame Warnings and binding the Warnings to the bonds
> BondsWithWarnings<-cbind(
+   SomeBonds2016, do.call(
+     rbind, lapply(FullAnalysis, `[[`, 1)
+   )
+ )
```

```
> summary(BondsWithWarnings[,c((ncol(SomeBonds2016)+1):ncol(BondsWithWarnings))])
Em_FIAD_differ    EmMatMissing    CpYOverride  RV_set100percent    NegLifeFlag
  Min.   :0.00       Min.   :0      Min.   :0         Min.   :0      Min.   :0
  1st Qu.:0.00       1st Qu.:0      1st Qu.:0         1st Qu.:0      1st Qu.:0
  Median :0.00       Median :0      Median :0         Median :0      Median :0
  Mean   :0.04       Mean   :0      Mean   :0         Mean   :0      Mean   :0
  3rd Qu.:0.00       3rd Qu.:0      3rd Qu.:0         3rd Qu.:0      3rd Qu.:0
  Max.   :1.00       Max.   :0      Max.   :0         Max.   :0      Max.   :0
      ZeroFlag   Em_Mat_SameMY ChronErrorFlag   FIPD_LIPD_equal IPD_CpY_Corrupt
     Min.   :0       Min.   :0   Min.   :0.00      Min.   :0.00    Min.   :0.00
     1st Qu.:0       1st Qu.:0   1st Qu.:0.00      1st Qu.:0.00    1st Qu.:0.00
     Median :0       Median :0   Median :0.00      Median :0.00    Median :0.00
     Mean   :0       Mean   :0   Mean   :0.01      Mean   :0.02    Mean   :0.09
     3rd Qu.:0       3rd Qu.:0   3rd Qu.:0.00      3rd Qu.:0.00    3rd Qu.:0.00
     Max.   :0       Max.   :0   Max.   :1.00      Max.   :1.00    Max.   :1.00
 EOM_Deviation     EOMOverride    DCCOverride          NoCoups
  Min.   :0.00    Min.   :0.00      Min.   :0      Min.   :0.00
  1st Qu.:0.00    1st Qu.:0.00      1st Qu.:0      1st Qu.:0.00
  Median :1.00    Median :1.00      Median :0      Median :0.00
  Mean   :0.69    Mean   :0.68      Mean   :0      Mean   :0.01
  3rd Qu.:1.00    3rd Qu.:1.00      3rd Qu.:0      3rd Qu.:0.00
  Max.   :1.00    Max.   :1.00      Max.   :0      Max.   :1.00
```

The summary information in *example 10: continued (I)* reveals that 1% of the bonds suffer from a chronological error (`ChronErrorFlag`) and 9% feature inconsistencies between the coupon payment dates and the number of interest payment dates per year `CpY` (`IPD_CpY_Corrupt`). To illustrate the rationale behind the plausibility analysis, a manual inspection of the affected bonds is performed below.[6]

```
> # example 10: continued (II)
>
> # manual examination of the rows where ChronErrorFlag = 1
> print(BondsWithWarnings[
+   which(BondsWithWarnings$ChronErrorFlag == 1),
+   c('ID.No', 'Issue.Date', 'FIAD.Input', 'FIPD.Input', 'LIPD.Input', 'Mat.Date')],
+   row.names = FALSE)
ID.No   Issue.Date   FIAD.Input   FIPD.Input   LIPD.Input     Mat.Date
   17   2016-08-23   2016-08-23   2017-08-23   2016-08-23   2017-08-23
>
> # manual examination of the rows where IPD_CpY_Corrupt = 1
> print(BondsWithWarnings[
+   which(BondsWithWarnings$IPD_CpY_Corrupt == 1),
+   c('ID.No', 'Issue.Date', 'FIAD.Input', 'FIPD.Input', 'LIPD.Input', 'Mat.Date',
+     'CpY.Input')], row.names = FALSE)
ID.No   Issue.Date   FIAD.Input   FIPD.Input   LIPD.Input     Mat.Date   CpY.Input
    2   2016-06-23   2016-06-23   2016-07-15   2019-05-15   2019-06-15           4
    4   2016-05-24   2016-05-24   2016-05-31   2017-04-30   2017-05-31           2
   19   2016-09-28   2016-09-28   2017-02-28   2021-08-31   2021-09-28           1
   56   2016-07-26   2016-07-26   2017-01-26   2020-07-26   2020-10-26           1
   64   2016-04-13   2016-04-13   2016-04-24   2017-03-24   2017-04-24           6
   65   2016-09-30   2016-09-30   2016-10-31   2018-02-28   2018-03-29           1
   70   2016-08-26   2016-08-26   2016-11-20   2056-05-20   2056-08-31           1
   82   2016-06-30   2016-06-30   2016-07-15   2028-09-15   2028-12-15           2
   84   2016-07-20   2016-07-20   2016-07-24   2016-09-24   2017-01-24           2
```

The chronological error occurred because the provided penultimate coupon date (`LIPD.Input`) is located prior to the supplied first interest payment date (`FIPD.Input`). Since the authenticity of `FIPD.Input` and `LIPD.Input` is unclear in this case, both are automatically dropped by `AnnivDates()`, and the calculation continues based upon the provided values of `Issue.Date` and `Mat.Date`.

As can be seen in the manual examination of the rows where `IPD_CpY_Corrupt = 1`, for all of them, there are inconsistencies between the value of `CpY.Input` and the interval between `FIPD.Input` and `LIPD.Input`. In the first row, for example, `CpY.Input` indicates that coupons are paid quarterly. If the

[6] Please refer to the package manual of **BondValuation** for detailed descriptions of the other warning flags.

value of FIPD.Input is correct, coupon payments should occur on July 15th, October 15th, January 15th, and April 15th. If the value of LIPD.Input is genuine, however, interest should be paid on May 15th, August 15th, November 15th, and February 15th. Finally, in this case, FIPD.Input and LIPD.Input can both be correct, lying on each other's anniversary dates for a value of CpY.Input = 6. Since it is not clear which of the three values is correct, AnnivDates() cannot automatically revise the input but only helps the user to identify the inconsistency. If the data are passed to AnnivDates() as they are, CpY.Input is assumed to be genuine, FIPD.Input and LIPD.Input are dropped, and the execution continues as if they were not provided in the first place, resulting in the temporal structure and cash flows provided below in *example 10: continued (III)*.

```
> # example 10: continued (III)
> # Printing data frame DateVectors for bond with ID.No = 2
> print(
+   as.data.frame(do.call(rbind, lapply(FullAnalysis, `[[`, 3)[2])),
+   row.names = FALSE)
 RealDates    RD_indexes    CoupDates   CD_indexes   AnnivDates   AD_indexes
2016-06-23    0.08888889   2016-09-15            1   2016-06-15            0
2016-09-15    1.00000000   2016-12-15            2   2016-09-15            1
2016-12-15    2.00000000   2017-03-15            3   2016-12-15            2
2017-03-15    3.00000000   2017-06-15            4   2017-03-15            3
2017-06-15    4.00000000   2017-09-15            5   2017-06-15            4
2017-09-15    5.00000000   2017-12-15            6   2017-09-15            5
2017-12-15    6.00000000   2018-03-15            7   2017-12-15            6
2018-03-15    7.00000000   2018-06-15            8   2018-03-15            7
2018-06-15    8.00000000   2018-09-15            9   2018-06-15            8
2018-09-15    9.00000000   2018-12-15           10   2018-09-15            9
2018-12-15   10.00000000   2019-03-15           11   2018-12-15           10
2019-03-15   11.00000000   2019-06-15           12   2019-03-15           11
2019-06-15   12.00000000         <NA>           NA   2019-06-15           12
>
> # Printing data frame PaySched for bond with ID.No = 2
> print(
+   as.data.frame(do.call(rbind, lapply(FullAnalysis, `[[`, 4)[2])),
+   row.names = FALSE)
 CoupDates   CoupPayments
2016-09-15      0.7368611
2016-12-15      0.8087500
2017-03-15      0.8087500
2017-06-15      0.8087500
2017-09-15      0.8087500
2017-12-15      0.8087500
2018-03-15      0.8087500
2018-06-15      0.8087500
2018-09-15      0.8087500
2018-12-15      0.8087500
2019-03-15      0.8087500
2019-06-15      0.8087500
```

The consequences of the plausibility-check-induced automated data revision by AnnivDates() are stored in the data frame Traits. Alongside the values that were initially provided and that are actually used in the subsequent calculations, Traits contains information on the types and lengths of the first and final coupon periods. *Example 10: continued (IV)* demonstrates how the data frame Traits can be extracted from the output of AnnivDates() and provides summary information on the lengths and types of the first and final coupon periods in the data frame SomeBonds2016. Of the 100 bonds in SomeBonds2016, only 20 have regular first coupon periods and 28 feature final coupon periods of regular length. The lengths of the first coupon periods vary from 1.37% to 1,200% of the bond-specific regular coupon period length, while the final coupon periods average 238% and span from 2.78% to 1,200% of the respective bond's regular coupon period length.

```
> # example 10: continued (IV)
> # Extracting the data frame Warnings and binding the Warnings to the bonds
> BondsWithTraits<-cbind(
+   SomeBonds2016, do.call(
+     rbind, lapply(FullAnalysis, `[[`, 2)
+   )
```

```
+ )
> summary(BondsWithTraits[, c('FCPType', 'LCPType', 'FCPLength', 'LCPLength')])
   FCPType       LCPType         FCPLength           LCPLength
long   :49    long   :51     Min.   : 0.01366     Min.   : 0.02778
short  :31    regular:28     1st Qu.: 0.92150     1st Qu.: 1.00000
regular:20    short  :21     Median : 1.00000     Median : 1.25140
                             Mean   : 2.19291     Mean   : 2.38417
                             3rd Qu.: 3.00000     3rd Qu.: 3.00000
                             Max.   :12.00000     Max.   :12.00000
```

Applying BondVal.Yield() to long format data

In addition to the time-invariant information in `SomeBonds2016`, the data frame `PanelSomeBonds2016` provides daily clean prices (`CP.Input`) and yields to maturity (`YtM.Input`) that correspond to the trade dates, `TradeDate`, and settlement dates, `SETT`. `TradeDate` is the calendar date on which the transaction is initiated and the quoted clean price is observed; `SETT` is the actual calendar date on which the transfer of cash and assets is completed. The settlement date is used for the following computation. `Example 11` below shows that `PanelSomeBonds2016` has 12,718 rows and 16 columns and provides summary information regarding the time-variant variables. The clean prices span from 90.38% to 224.16%, while the yields to maturity average −0.01593%, varying from −1.725% to 2%.

```
> # example 11
> library(BondValuation)
> dim(PanelSomeBonds2016)
[1] 12718    16
> summary(PanelSomeBonds2016[, c(13:16)])
        TradeDate                 SETT          CP.Input          YtM.Input
Min.   :2016-01-29    Min.   :2016-02-02    Min.   : 90.38    Min.   :-1.72500
1st Qu.:2016-08-03    1st Qu.:2016-08-05    1st Qu.:102.73    1st Qu.:-0.35000
Median :2016-10-03    Median :2016-10-05    Median :105.79    Median :-0.02500
Mean   :2016-09-20    Mean   :2016-09-23    Mean   :112.53    Mean   :-0.01593
3rd Qu.:2016-11-17    3rd Qu.:2016-11-21    3rd Qu.:113.21    3rd Qu.: 0.27500
Max.   :2016-12-30    Max.   :2017-01-03    Max.   :224.16    Max.   : 2.00000
```

In the following, *example 12*, the function `BondVal.Yield()` is used to determine τ, accrued interest, dirty price, yield to maturity, modified duration, MacAulay duration, and convexity for each bond and settlement date in `PanelSomeBonds2016`. In alternative 1, the function `BondVal.Yield()` is applied to every row of the data frame `PanelSomeBonds`. Alternative 2 demonstrates a significantly faster approach, where `AnnivDates()` is applied to every bond's time-invariant characteristics before its output is passed to `BondVal.Yield()` for every settlement date. Alternative 2 takes less than half the time of alternative 1.[7]

```
> # example 12
> # analysis of PanelSomeBonds2016 with BondValuation
> library(BondValuation)
> Panel <- PanelSomeBonds2016
> Vars <- c("tau","AccrInt","DP","YtM","ModDUR","MacDUR","Conv")
> Panel[, c((ncol(Panel) + 1) : (ncol(Panel) + length(Vars)))] <- as.numeric(NA)
> names(Panel)[(ncol(Panel) - length(Vars) + 1) : ncol(Panel)] <- Vars
>
> # Alternative 1: loop through the data frame
> #                applying BondVal.Yield to each row
> Time.Alt01 <- system.time(
+   for (i in c(1:nrow(Panel))) {
+     BondVal.Out <- suppressWarnings(
+       BondVal.Yield(CP = Panel$CP.Input[i],
+                     SETT = Panel$SETT[i],
+                     Em = Panel$Issue.Date[i],
+                     Mat = Panel$Mat.Date[i],
+                     CpY = Panel$CpY.Input[i],
+                     FIPD = Panel$FIPD.Input[i],
+                     LIPD = Panel$LIPD.Input[i],
```

[7] On an "Intel(R) Core(TM) i7-3687U CPU @ 2.10GHz" machine, alternative 1 takes about 350 seconds, while alternative 2 takes ca. 170 seconds.

```
+                     FIAD = Panel$FIAD.Input[i],
+                     RV = Panel$RV.Input[i],
+                     Coup = Panel$Coup.Input[i],
+                     DCC = Panel$DCC.Input[i],
+                     EOM = Panel$EOM.Input[i],
+                     Precision = .Machine$double.eps^0.5
+       )
+     )
+     Panel[i, c((ncol(Panel) - length(Vars) + 1) : ncol(Panel))] <-
+       round(as.numeric(BondVal.Out[c(11, 2 : 7)]), 4)
+   }, gcFirst = TRUE
+ )
>
>
> # Alternative 2: Run AnnivDates() once per Bond-ID and pass its output
> #                to BondVal.Yield() for every row with the same Bond-ID
> NonDuplID <- c(which(!duplicated(Panel$ID.No)), (nrow(Panel)+1))
> Time.Alt02 <- system.time(
+   for (i in c(1 : (length(NonDuplID) - 1))) {
+     BondCount <- NonDuplID[i]
+     AnnivDates.Out <- suppressWarnings(
+       AnnivDates(Em = Panel$Issue.Date[BondCount],
+                  Mat = Panel$Mat.Date[BondCount],
+                  CpY = Panel$CpY.Input[BondCount],
+                  FIPD = Panel$FIPD.Input[BondCount],
+                  LIPD = Panel$LIPD.Input[BondCount],
+                  FIAD = Panel$FIAD.Input[BondCount],
+                  RV = Panel$RV.Input[BondCount],
+                  Coup = Panel$Coup.Input[BondCount],
+                  DCC = Panel$DCC.Input[BondCount],
+                  EOM = Panel$EOM.Input[BondCount]
+       )
+     )
+     for (j in c(NonDuplID[i] : (NonDuplID[i + 1] - 1))) {
+       BondVal.Out <- suppressWarnings(
+         BondVal.Yield(CP = Panel$CP.Input[j],
+                       SETT = Panel$SETT[j],
+                       Em = Panel$Issue.Date[j],
+                       Mat = Panel$Mat.Date[j],
+                       CpY = Panel$CpY.Input[j],
+                       FIPD = Panel$FIPD.Input[j],
+                       LIPD = Panel$LIPD.Input[j],
+                       FIAD = Panel$FIAD.Input[j],
+                       RV = Panel$RV.Input[j],
+                       Coup = Panel$Coup.Input[j],
+                       DCC = Panel$DCC.Input[j],
+                       EOM = Panel$EOM.Input[j],
+                       InputCheck = 0,
+                       Precision = .Machine$double.eps^0.5,
+                       AnnivDatesOutput = AnnivDates.Out
+         )
+       )
+       Panel[j, c((ncol(Panel) - length(Vars) + 1) : ncol(Panel))] <-
+         round(as.numeric(BondVal.Out[c(11, 2 : 7)]), 4)
+     }
+   }, gcFirst = TRUE
+ )
>
> round(Time.Alt02[[3]] / Time.Alt01[[3]], 2)
[1] 0.48
```

Conclusion

This article introduces the R package **BondValuation** and provides guidance on its application for analysis of large data frames of fixed coupon bonds. The theoretical foundation of the package is the generalized valuation methodology developed by Djatschenko (2019). Its seamless implementation in **BondValuation** is framed by a set of routines that assist the user in data quality evaluation and automatically correct corrupted entries.

BondValuation is the first R package that properly handles irregular first and final coupon periods of fixed coupon bonds and provides a comprehensive coverage of the day count conventions (`DCC`) used in the global bond markets. Currently, 16 different `DCC`s are implemented, which account for the vast majority of the methods used in the global fixed income markets. Within its scope, the R package **BondValuation** performs correctly and efficiently. Nevertheless, the current version of the software remains open for further development and refinement. Essentially, the calculations are performed under the assumption that interest accrual and temporal structure follow the same `DCC`. The option `CalcMethod` in the functions `BondVal.Price()` and `BondVal.Yield()` can be used to force the temporal structure to follow the *ACT/ACT (ICMA)* method, while the `DCC` passed to the respective function is used to compute accrued interest. In future versions of the package, I intend to implement an explicit assignment of `DCC` to both interest accrual and temporal structure, which will increase the flexibility of the package.

A further limitation is that the calendar dates of the temporal structure are currently returned, regardless of whether or not they are business days. Although this is the common approach in theoretical bond valuation, including the possibility of business day adjustments for cash flows would be particularly appealing to practitioners. Along with business day adjustments, future versions of **BondValuation** can be extended by methods for bond portfolio analysis.

The current version of **BondValuation** is designed for processing non-callable, option-free, non-sinkable fixed coupon bonds and zero bonds. With the implemented methods, callable bonds can be analyzed through appropriate adjustment of the maturity date to the next call date, returning the so-called yield-to-worst and the corresponding duration and convexity measures. Based on the implemented functions, the R package **BondValuation** can be extended to incorporate methods for explicit treatment of callable, sinkable and convertible fixed and floating rate bonds.

The R package **BondValuation** provides the computational foundation for the exploration of a variety of interesting research questions related to the analysis of fixed income securities across markets. Even considering the limitations described above, the software is also useful to practitioners. I intend to continuously extend and improve the package, and I highly appreciate feedback from the users.

Acknowledgments

I would like to thank Ingo Geishecker for our frequent discussions and his profound advice. I am also grateful to Karl Ludwig Keiber and Philipp Otto for their helpful comments and suggestions, and to Inna Keil for her excellent research assistance. All remaining errors are my own responsibility.

Bibliography

Banking Federation of the European Union. Master Agreement for Financial Transactions - Supplement to the Derivatives Annex - Interest Rate Transactions, 2004. URL http://www.ebf.eu/wp-content/uploads/2017/07/10InterestRateTransactions-2004-02699-01-E.pdf. [p125]

A. Caputo Silva, L. Oliveira de Carvalho, and O. Ladeira de Medeiros. *PUBLIC DEBT: the Brazilian experience*. National Treasury Secretariat and World Bank, Brasilia, BR, 2010. ISBN 978-85-87841-44-5. URL http://documents.worldbank.org/curated/en/967171469672182286/pdf/700810ESW0P1160Brazilian0Experience.pdf. [p125]

D. Christie and SWX Swiss Exchange. Accrued Interest & Yield Calculations and Determination of Holiday Calendars, 2003. URL http://janroman.dhis.org/finance/General/accrued_interest_en.pdf. [p125]

W. Djatschenko. The Nitty Gritty of Bond Valuation: A Generalized Methodology for Straight Bond Analysis, July 2019. URL http://dx.doi.org/10.2139/ssrn.3205167. Discussion Paper. [p124, 125, 126, 131, 132, 133, 134, 140]

D. Eddelbuettel, K. Nguyen, and T. Leitch. *RQuantLib: R Interface to the 'QuantLib' Library*, 2018. URL https://CRAN.R-project.org/package=RQuantLib. R package version 0.4.5. [p124]

International Capital Market Association. Rule 251 Accrued Interest Calculation - Excerpt from ICMA's Rules and Recommendations, 2010. URL https://www.isda.org/a/NIJEE/ICMA-Rule-Book-Rule-251-reproduced-by-permission-of-ICMA.pdf. [p125]

International Swaps and Derivatives Association, Inc. EMU and Market Conventions: Recent Developments, 1998. [p125]

International Swaps and Derivatives Association, Inc. *2006 ISDA Definitions*. International Swaps and Derivatives Association, Inc., New York, 2006. [p125]

Investment Industry Association of Canada (IIAC). Canadian Conventions in Fixed Income Markets - A Reference Document of Fixed Income Securities Formulas and Practices; Release: 1.3, 2018. URL https://iiac.ca/wp-content/uploads/Canadian-Conventions-in-FI-Markets-Release-1.3.pdf. [p125]

Itaú Unibanco. *Brazilian Sovereign Fixed Income and Foreign Exchange Markets*. Itaú Unibanco, 1st edition, 2017. [p125]

D. Krgin. *The Handbook of Global Fixed Income Calculations*. Wiley, New York, 1st edition, 2002. ISBN 978-0-471-21835-7. [p125, 126]

J. Mayle. *Standard Securities Calculation Methods: Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, volume 1. Securities Industry Association, New York, 3rd edition, 1993. ISBN 1-882936-01-9. [p125]

Municipal Securities Rulemaking Board. *MSRB Rule Book*. Municipal Securities Rulemaking Board, Washington, DC, 2017. URL http://www.msrb.org/msrb1/pdfs/MSRB-Rule-Book-October-2017.pdf. [p125]

QuantLib Team. Quantlib: A free/open-source library for quantitative finance, 2018. URL http://quantlib.org/. [p124]

Wadim Djatschenko
European University Viadrina
Frankfurt (Oder)
Germany
ORCiD: 0000-0003-0653-8779
wadim.djatschenko@gmx.de

ConvergenceClubs: A Package for Performing the Phillips and Sul's Club Convergence Clustering Procedure

by Roberto Sichera and Pietro Pizzuto

Abstract This paper introduces package ConvergenceClubs, which implements functions to perform the Phillips and Sul (2007, 2009) club convergence clustering procedure in a simple and reproducible manner. The approach proposed by Phillips and Sul to analyse the convergence patterns of groups of economies is formulated as a nonlinear time varying factor model that allows for different time paths as well as individual heterogeneity. Unlike other approaches in which economies are grouped a priori, it also allows the endogenous determination of convergence clubs. The algorithm, usage, and implementation details are discussed.

Introduction

Economic convergence refers to the idea that per–capita incomes of poorer economies will tend to grow at faster rates than those of richer economies. The issue has been widely investigated in economic literature since the classical contributions on economic growth and development (Solow, 1956; Myrdal, 1957). In addition to the traditional concepts of beta and sigma convergence, an increasing amount of literature has recently emerged on the concept of *club convergence*. This notion was originally introduced by Baumol (1986) to describe convergence among a subset of national economies and it has quickly spread also at the regional level. Several contributions have tried to empirically investigate the topic proposing different methodologies. For example, Quah (1996) developed a Markov chain model with probability transitions to estimate the evolution of income distribution. Le Gallo and Dall'Erba (2005) proposed a spatial approach to detect convergence clubs using the Getis–Ord statistic. Corrado et al. (2005) introduced a multivariate stationarity test in order to endogenously identify regional club clustering.

More recently, Phillips and Sul (2007, 2009) proposed a time-varying factor model that allows for individual and transitional heterogeneity to identify convergence clubs. Due to its positive attributes, this methodology has become predominant in the analysis of the convergence patterns of economies. In fact, it has several advantages. First, it allows for different time paths as well as individual heterogeneity, therefore, different transitional paths are possible[1]. Second, unlike other approaches in which economies are grouped a priori, this methodology enables the endogenous (data–driven) determination of convergence clubs. Third, the test does not impose any particular assumption concerning trend stationarity or stochastic non–stationarity since it is robust to heterogeneity and to the stationarity properties of the series.

As for existing routines, Phillips and Sul (2007, 2009) provided Gauss (Aptech Systems, 2016) code used in their empirical studies. Schnurbus et al. (2017) provided a set of R functions to replicate the key results of Phillips and Sul (2009), while Du (2018) developed a full Stata (StataCorp, 2017) package to perform the club convergence algorithm. A dedicated R package for this methodology has been missing. The **ConvergenceClubs** (Sichera and Pizzuto, 2019) package fills this gap, since it allows to carry out the Phillips and Sul's methodology in a simple and reproducible fashion, allowing for easy definition of the parameters. Moreover, our package also implements the alternative club merging algorithm developed by von Lyncker and Thoennessen (2017).

The remainder of the paper is organised as follows. First, the club convergence methodology is presented. Then, the main features of the package are listed and described. Finally, an example based on Phillips and Sul (2009) data is provided.

Methodology

The log–t test

The approach proposed by Phillips and Sul is based on a modification of the conventional panel data decomposition of the variable of interest. In fact, panel data X_{it} are usually decomposed in the

[1]For example, in the context of income convergence, the approach proposed by Phillips and Sul allows to account for heterogeneity in technology growth rates and in the speed of convergence, unlike the traditional neoclassical model *à la* Solow that assumes homogeneous technological progress.

following way:

$$X_{it} = g_{it} + a_{it}\,, \tag{1}$$

where g_{it} is the systematic factor (including the permanent common component) and a_{it} is the transitory component. In order to account for temporal transitional heterogeneity they modify eq. (1) as follows:

$$X_{it} = \left(\frac{g_{it} + a_{it}}{\mu_t}\mu_t\right) = b_{it}\mu_t\,, \tag{2}$$

where b_{it} is the systematic idiosyncratic element that is allowed to evolve over time and to include a random component that absorbs a_{it}, and μ_t is the common factor. In this dynamic factor formulation, that allows to separate common from idiosyncratic components, b_{it} becomes the transition path of the economy to the common steady–state growth path determined by μ_t. Particularly, the common growth component μ_t, may follow either a trend–stationary process or a non–stationary stochastic trend with drift, since a specific assumption regarding the behaviour of μ_t is not necessary.

In order to test if different economies converge, a key role is played by the estimation of b_{it}. According to the authors, the estimation of this parameter is not possible without imposing additional structural restrictions and assumptions. However, as a viable way to model this element, they propose the construction of the following relative transition component:

$$h_{it} = \frac{X_{it}}{N^{-1}\sum_{i=1}^{N} X_{it}} = \frac{b_{it}}{N^{-1}\sum_{i=1}^{N} b_{it}}\,, \tag{3}$$

which is called *relative transition path* and can be directly computed from the data. In such a way it is possible to remove the common steady–state trend μ_t, tracing an individual trajectory for each economy i in relation to the panel average. In other words, the relative transition path describes the relative individual behaviour as well as the relative departures of the i–th economy from the common growth path μ_t.

In presence of convergence, there should be a common limit in the transition path of each economy and the coefficient h_{it} should converge towards unity ($h_{it} \to 1$) for all $i = 1, \ldots, N$, as $t \to \infty$. At the same time, the cross–sectional variation H_{it} (computed as the quadratic distance measure for the panel from the common limit) should converge to zero:

$$H_t = N^{-1}\sum_{i=1}^{N}(h_{it} - 1)^2 \to 0 \qquad \text{as } \; t \to \infty\,. \tag{4}$$

In order to construct a formal statistical test for convergence, Phillips and Sul (2007, 2009) assume the following semi–parametric specification of b_{it}:

$$b_{it} = b_i + \frac{\sigma_i\,\xi_{it}}{L(t)\,t^{\alpha}}\,, \tag{5}$$

where b_i is fixed (time invariant), the ξ_{it} are i.i.d. $N(0,1)$ random variables across i, but weakly dependent over t, $L(t)$ is a slowly varying increasing function (with $L(t) \to \infty$ as $t \to \infty$), and α is the decay rate, or more specifically in this case, the convergence rate. The null hypothesis of convergence can be written as $H_0 : b_i = b$ and $\alpha \geq 0$ versus the alternative $H_1 : b_i \neq b$ for all i, or $\alpha < 0$. Under H_0, different transitional paths are possible, including temporary divergence (a stylized way in which economies may converge is shown in fig. 1).

More formally, to test the presence of convergence among different economies, Phillips and Sul (2007, 2009) suggest to estimate the following equation model through the ordinary least squares method:

$$\log\frac{H_1}{H_t} = -2\log(\log t) = \alpha + \beta\log t + u_t\,, \quad \text{for } t = [rT], [rT]+1, \ldots, T\,, \tag{6}$$

where $H_t = N^{-1}\sum_{i=1}^{N}(h_{it} - 1)^2$, and H_1/H_t is the cross–sectional variance ratio; β is the speed of convergence parameter of b_{it}; $-2\log(\log t)$ is a penalization function that improves the performance of the test mainly under the alternative; r assumes a positive value in the interval $(0, 1]$ in order to discard the first block of observation from the estimation, and $[rT]$ is the integer part of rT. To this regard, Phillips and Sul suggest to use $r \in [0.2,\ 0.3]$ for small sample size ($T < 50$) as a result of Monte Carlo simulations. The null hypothesis of convergence is tested through a one–sided t–test robust to heteroskedasticity and autocorrelation (*HAC*) of the inequality $\alpha > 0$ (using the estimated $\hat{\beta} = 2\,\alpha$) and specifically it is rejected at the 5% level if $t_{\hat{\beta}} < -1.65$. This procedure, generally called *log–t test*,

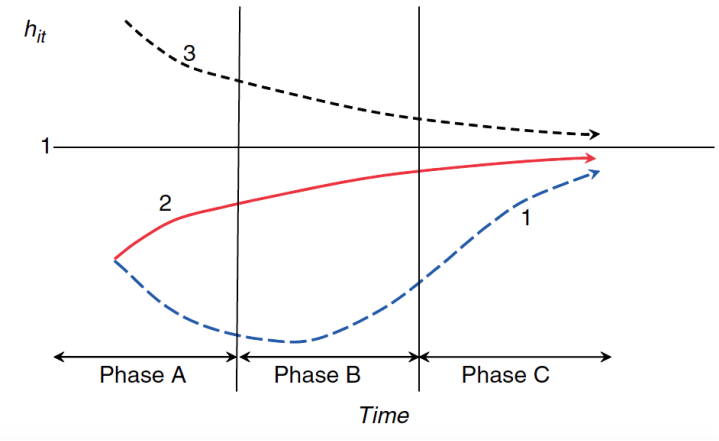


Figure 1: Different transition paths and phases of transition. Source: Phillips and Sul (2009).

has power against cases of club convergence. Hence, if the log–t test is rejected for the whole sample, the authors suggest to repeat the test procedures according to a clustering mechanism consisting of four steps, described below (see next section).

The clustering algorithm

When the log t–test is rejected for the whole sample, the test procedure should be repeated according to the following clustering mechanism:

1. (Cross–section last observation ordering): Sort units in descending order according to the last panel observation of the period;
2. (Core group formation): Run the log–t regression for the first k units ($2 < k < N$) maximizing k under the condition that t-value > -1.65. In other words, choose the core group size k^* as follows:

$$k^* = \arg\max_k \{t_k\} \quad \text{subject to } \min\{t_k\} > -1.65\,. \tag{7}$$

   If the condition $t_k > -1.65$ does not hold for $k = 2$ (the first two units), drop the first unit and repeat the same procedure. If $t_k > -1.65$ does not hold for any units chosen, the whole panel diverges;
3. (Sieve the data for club membership): After the core group k^* is detected, run the log–t regression for the core group adding (one by one) each unit that does not belong to the latter. If t_k is greater than a critical value c^* add the new unit in the convergence club. All these units (those included in the core group k^* plus those added) form the first convergence club;
4. (Recursion and stopping rule): If there are units for which the previous condition fails, gather all these units in one group and run the log–t test to see if the condition $t_k > -1.65$ holds. If the condition is satisfied, conclude that there are two convergence clubs. Otherwise, step 1 to 3 should be repeated on the same group to determine whether there are other subgroups that constitute convergence clubs. If no further convergence clubs are found (hence, no k in step 2 satisfies the condition $t_k > -1.65$), the remaining units diverge.

Phillips and Sul (2007) suggest to make sure $t_k > -1.65$ for the clubs. Otherwise, repeat the procedure by increasing the value of the c^* parameter until the condition $t_k > -1.65$ is satisfied for the clubs.

The merging algorithms

Due to the fact that the number of identified clubs strongly depends on the core group formation, a key role is played by the critical value c^*. The choice of this parameter is related to the desired degree of conservativeness, where a higher level of c^* corresponds to a more conservative selection. In other words, the higher is c^* the less likely we add wrong members to the convergence clubs. Related to this, for small samples ($T < 50$) Phillips and Sul suggest to set $c^* = 0$.

However, as the same authors suggest, a high value of c^* can lead to more groups than those really existing. For these reasons Phillips and Sul (2009) suggest a *club merging algorithm* to avoid this

over–determination. This algorithm suggests to merge for adjacent groups. In particular, it works as follows:

1. Take the first two groups detected in the basic clustering mechanism and run the log–t test. If the t statistic is larger than -1.65, these groups together form a new convergence club;
2. Repeat the test adding the next group and continue until the basic condition (t statistic > -1.65) holds;
3. If the convergence hypothesis is rejected, conclude that all previous groups converge, except the last one. Hence, start again the merging algorithm beginning from the group for which the hypothesis of convergence did not hold.

In our package we also provide the implementation in R of an alternative club merging algorithm developed by von Lyncker and Thoennessen (2017). They introduce two innovations in the club merging algorithm by Phillips and Sul. First, they add a further condition to the club clustering algorithm to avoid mistakes in merging procedures in the case of transition across clubs. Second, they propose an algorithm for diverging units. The first algorithm works as follows:

1. Take all the P groups detected in the basic clustering mechanism and run the t–test for adjacent groups, obtaining a $(M \times 1)$ vector of convergence test statistics t (where $M = P - 1$ and $m = 1, \ldots, M$);
2. Merge for adjacent groups starting from the first, under the conditions $t(m) > -1.65$ and $t(m) > t(m+1)$. In particular, if both conditions hold, the two clubs determining $t(m)$ are merged and the algorithm starts again from step 1, otherwise it continues for all following pairs;
3. For the last element of vector M (the value of the last two clubs) the only condition required for merging is $t(m = M) > -1.65$.

For the second algorithm, von Lyncker and Thoennessen (2017) claim that units identified as divergent by the original clustering procedure by Phillips and Sul might not necessarily still diverge in the case of new convergence clubs detected with the club merging algorithm. To test if divergent units may be included in one of the new convergence clubs, they propose the following algorithm:

1. Run a log–t test for all diverging units; if $t_k > -1.65$, all these units form a convergence club (this step is implicitly included in Phillips and Sul basic algorithm);
2. Run a log–t test for each diverging units and each club, creating a matrix of t–statistic values with dimension $d \times p$, where each row d represents a divergent region and each column p represents a convergence club;
3. Take the highest t–value greater than a critical parameter e^* and add the respective region to the corresponding club, then start again from step 1. The authors suggest to use $e^* = t = -1.65$;
4. The algorithm stops when no t–value $> e^*$ is found in step 3, and as a consequence all remaining units are considered divergent.

The ConvergenceClubs package

ConvergenceClubs aims to make the clustering procedure described above easy to perform and simply reproducible.

The log–t test is performed by function `estimateMod()`. It takes as main input the vector of cross–sectional variances H for the units to be tested for convergence, which can be obtained through function `computeH()`:

```
# Compute cross-sectional variances
computeH(X, quantity = "H", id)

# Perform the log-t test
estimateMod(H, time_trim=1/3, HACmethod = c("FQSB", "AQSB"))
```

The former takes a matrix or data.frame object containing time series data and returns either the vector of cross–sectional variances H or the matrix of transition paths h, or both, depending on the value of argument `quantity`. These quantities can also be computed on a subset of units by selecting the unit IDs through argument `id`. Function `estimateMod()` takes two additional arguments, `time_trim` and `HACmethod`, described later. These two functions are available for the user who wants to test the convergence hypothesis on a set of units. This is especially useful to assess the opportunity of carrying out the clustering procedure during the initial phase of a study.

Nonetheless, the log–t test over the whole sample is automatically performed before starting the clustering procedure by function `findClubs()`. This is the main function of the package, as it carries out Phillips and Sul's clustering algorithm:

```
findClubs(X, dataCols, unit_names = NULL,
          refCol, time_trim = 1/3, cstar = 0, HACmethod = c("FQSB", "AQSB"))
```

where `X` is a data frame containing the data, `dataCols` is an integer vector indicating the column indices of the time series data, and `unit_names` is an integer scalar, indicating the index of the column of `X` that includes id codes for the units (e.g. the name of the countries/regions). The parameters of the clustering procedure are regulated by the following arguments.

- `refCol`: takes an integer value representing the index of the column to use for ordering data;
- `time_trim`: accepts numeric scalars between 0 and 1, and indicates the portion of time periods to trim when running the log–t regression model. By default, `time_trim=1/3`, which means that the first third of the time series period is discarded, as suggested by Phillips and Sul (2007, 2009);
- `cstar`: takes a scalar indicating the threshold value of the sieve criterion c^* to include units in the detected core (primary) group (step 3 of Phillips and Sul (2007, 2009) clustering algorithm). The default value is 0;
- `HACmethod`: accepts a character string indicating whether a *Fixed Quadratic Spectral Bandwidth* (`HACmethod="FQSB"`) or an *Adaptive Quadratic Spectral Bandwidth* (`HACmethod="AQSB"`) should be used for the truncation of the Quadratic Spectral kernel in estimating the log–t regression model with heteroskedasticity and autocorrelation consistent standard errors. The default method is `FQSB`.

The clustering procedure is performed by iteratively calling two internal functions: `coreG()` and `club()`, which implement steps 2 and 3 of Phillips and Sul clustering algorithm, respectively.

Function `findClubs()` returns an object belonging to the *S3* class `"convergence.clubs"`. Objects belonging to this class are lists that include results about clubs and divergent units that have been detected by the clustering procedure. Their structure can be analysed through function `str()`, and their elements can be accessed as commonly done with list elements.

Information about clubs and divergent units can be easily displayed by means of functions `print()` and `summary()`, for which the package provides specific methods for class `"convergence.clubs"`. A `plot()` method is available for class `"convergence.clubs"`, which provides a way to visualise the transition paths of the units included in convergence clubs, and also the average transition paths for each club:

```
plot(x, y = NULL, nrows = NULL, ncols = NULL, clubs, avgTP = TRUE, avgTP_clubs,
     y_fixed = FALSE, legend = FALSE, save = FALSE, filename, path, width = 7,
     height = 7, device = c("pdf", "png", "jpeg"), res, ...)
```

Plot customisation (i.e. clubs to be displayed or the number of rows and columns of the graphical layout) and options to export it to a file are discussed in more details in the package manual (Sichera and Pizzuto, 2019). Finally, the merging algorithms described in the previous section are implemented in function `mergeClubs()`:

```
mergeClubs(clubs, time_trim, mergeMethod = c("PS", "vLT"),
           threshold = -1.65, mergeDivergent = FALSE, estar = -1.65)
```

Merging is performed on argument `clubs`, an object of class `"convergence.clubs"`, by means of either the Phillips and Sul (2009) or the von Lyncker and Thoennessen (2017) algorithm, selected through argument `mergeMethod`. Through argument `threshold` it is possible to change the significance level of the log–t test for club merging. Moreover, argument `mergeDivergent` determines whether the test for diverging units according to von Lyncker and Thoennessen (2017) should be performed, while argument `estar` is used to set the value of the critical parameter e^*. Function `mergeClubs()` returns an object of class `"convergence.clubs"` as well, thus information about the new clubs can be accessed and summarised as previously discussed.

A detailed example of all functionalities of the package is presented in the next section.

Application to the country GDP dataset

In this section we provide an example that replicates the results of Phillips and Sul (2009). The dataset `GDP`, available in package **ConvergenceClubs**, covers a panel of 152 countries for the period 1970-2003.

First, we filter the data using the Hodrick-Prescott filter methodology by means of function `hpfilter` in package **mFilter** (Balcilar, 2018). Filtered data are also available in the package through dataset `filteredGDP`.

```
### Load ConvergenceClubs package
library(ConvergenceClubs)

### Load GDP data
data("GDP")

### Filter data
logGDP <- log(GDP[,-1])
filteredGDP <- apply(logGDP, 1,
                     function(x){mFilter::hpfilter(x, freq=400, type="lambda")$trend} )
filteredGDP <- data.frame(Countries = GDP[,1], t(filteredGDP), stringsAsFactors=FALSE )
colnames(filteredGDP) <- colnames(GDP)

## Filtered data are available in the package
data(filteredGDP)
```

By using the `estimateMod()` function we perform the log–t test over the whole sample in order to verify whether all units converge.

```
### log-t test over all units
H <- computeH(filteredGDP[,-1], quantity = "H")
round(estimateMod(H, time_trim=1/3, HACmethod = "FQSB"), 3)
#    beta  std.err   tvalue   pvalue
#  -0.875    0.005 -159.555    0.000
```

The null hypothesis of convergence is rejected at 5% level since the t-value is smaller than -1.65. Therefore, we proceed with the identification of convergence clubs, which is performed using the `findClubs()` function. As for the arguments, we set:

- `unit_names=1` indicates that Countries' IDs are represented in the first column of the dataset;
- `dataCols=2:35` indicates the columns (years) for which the test should be performed;
- `refCol=35` represents the final period according to which data should be ordered (see step 1 of the clustering algorithm).
- `time_trim=1/3` represents the portion of time periods to trim when running the log–t regression model;
- `cstar= 0` is the threshold value of the sieve criterion c^* (see step 3 of the clustering algorithm);
- `HACmethod = 'FQSB'` indicates that the Fixed Quadratic Spectral Bandwidth is used for the truncation of the Quadratic Spectral kernel in estimating the log–t regression model.

```
### Cluster Countries using GDP from year 1970 to year 2003, with 2003 as reference year
clubs <- findClubs(filteredGDP, dataCols=2:35, unit_names = 1, refCol=35,
                   time_trim=1/3, cstar=0, HACmethod = 'FQSB')

class(clubs)
# [1] "convergence.clubs" "list"
```

As we can see, `clubs` is an object of class `"convergence.clubs"`, that is a common list, whose structure can be displayed through function `str()`, and whose elements can be accessed as usual:

```
str(clubs, give.attr=FALSE)
```

A method for function `summary()` is provided for class `"convergence.clubs"`. It produces a summary table with the key results of the clustering procedure:

```
summary(clubs)

# Number of convergence clubs: 7
# Number of divergent units: 0
#
#         | # of units  | beta     | std.err  | tvalue
# -------- ------------- ---------- ---------- ----------
#  club1  |   50        |   0.382  |   0.041  |   9.282
#  club2  |   30        |   0.24   |   0.035  |   6.904
#  club3  |   21        |   0.11   |   0.032  |   3.402
#  club4  |   24        |   0.131  |   0.064  |   2.055
```

```
#  club5  |   14        |   0.19   |   0.111  |   1.701
#  club6  |   11        |   1.003  |   0.166  |   6.024
#  club7  |   2         |  -0.47   |   0.842  |  -0.559
```

The summary shows that there are 7 clubs and no divergent units. For each club, the summary also reports how many units are included, the beta coefficient of the log–t test, its standard error, and the value of the t–statistics. This exercise exactly replicates the results obtained by Phillips and Sul (2009). A minor difference concerns the last two clubs (6 and 7). In the original paper, Phillips and Sul showed a divergence group of 13 countries. However, another iteration of the algorithm using these 13 countries suggests the presence of two clubs consisting of 11 and 2 countries, respectively (on this point see also Schnurbus et al. (2017) and Du (2018)).

As shown in the following example, information about the club composition can be obtained using the print() function. For brevity, only the output for the first club is shown.

```
## Print results
print(clubs)

# or just
clubs

# ================================================================================================
# club 1
# ------------------------------------------------------------------------------------------------
# United.States, Norway, Bermuda, United.Arab.Emirates, Qatar, Luxembourg, Singapore,
# Switzerland, Hong.Kong, Denmark, Ireland, Austria, Australia, Canada, Macao,
# Netherlands, Kuwait, Iceland, United.Kingdom, Germany, France, Sweden, Belgium, Japan,
# Brunei, Finland, Italy, Cyprus, Puerto.Rico, Israel, New.Zealand, Taiwan, Spain, Malta,
# Korea..Republic.of, Portugal, Oman, Mauritius, Antigua, St..Kitts...Nevis, Chile,
# Malaysia, Equatorial.Guinea, Dominica, St.Vincent...Grenadines, Botswana, Thailand,
# Cape.Verde, China, Maldives
#
# beta:      0.3816
# std.err:   0.0411
# tvalue     9.2823
# pvalue:    1
#
# [...]
```

Transition path plots can be generated through the function `plot()`. Here we show some examples.

```
# Plot Transition Paths for all units in each club and average Transition Paths
# for all clubs
plot(clubs)

# Plot Transition Paths
plot(clubs, avgTP = FALSE, nrows = 4, ncols = 2, plot_args = list(type='l'))

# Plot only average Transition Paths of each club
plot(clubs, clubs=NULL, avgTP = TRUE, legend=TRUE, plot_args = list(type='o'))
```

The second and third commands produce figs. 2 and 3, respectively. In the first case, we can see how economies approach the steady–state of each club. Conversely, in the second case, the comparison among the average transitional behaviour of each club is shown.

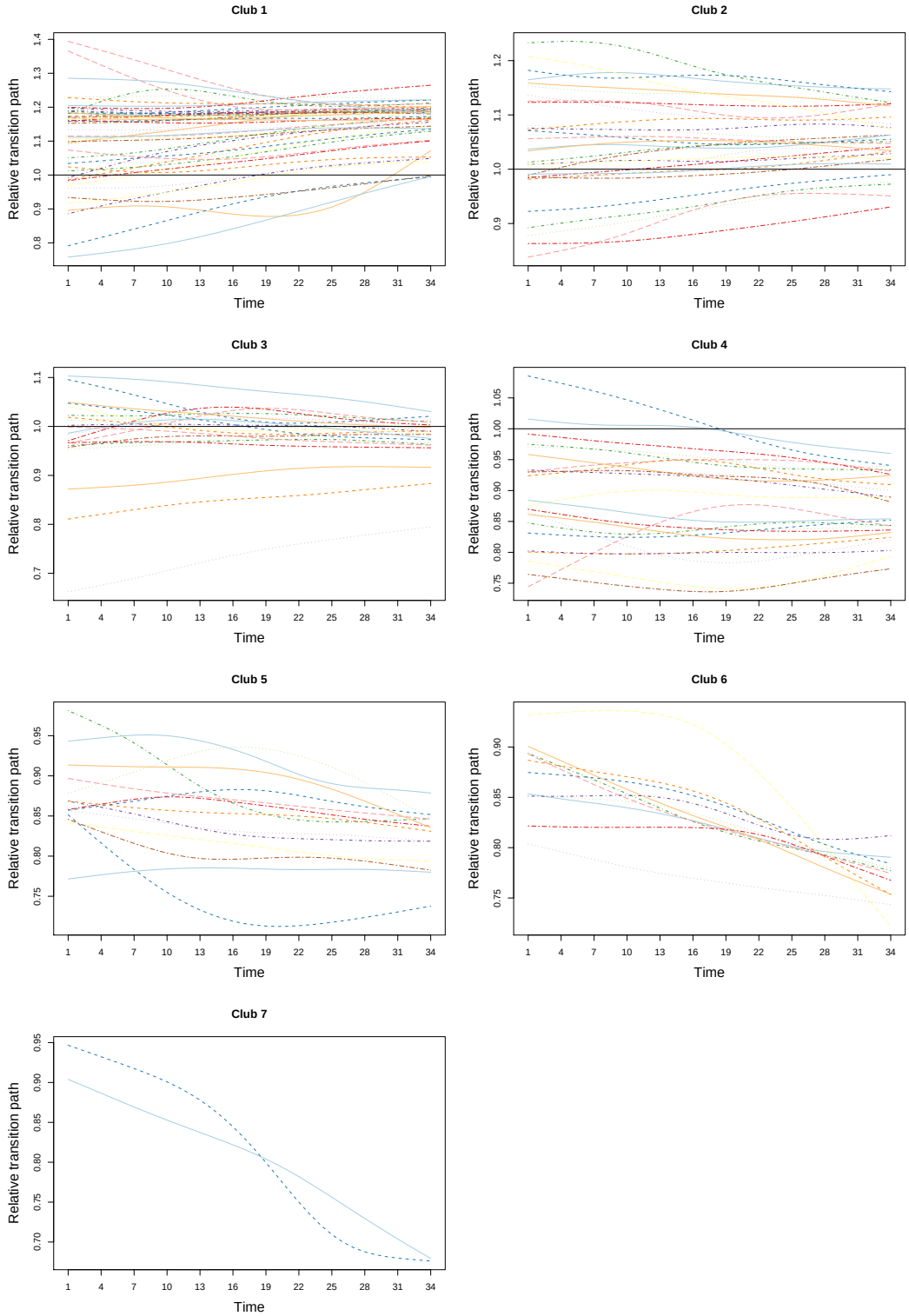


Figure 2: Transition paths within each convergence club.

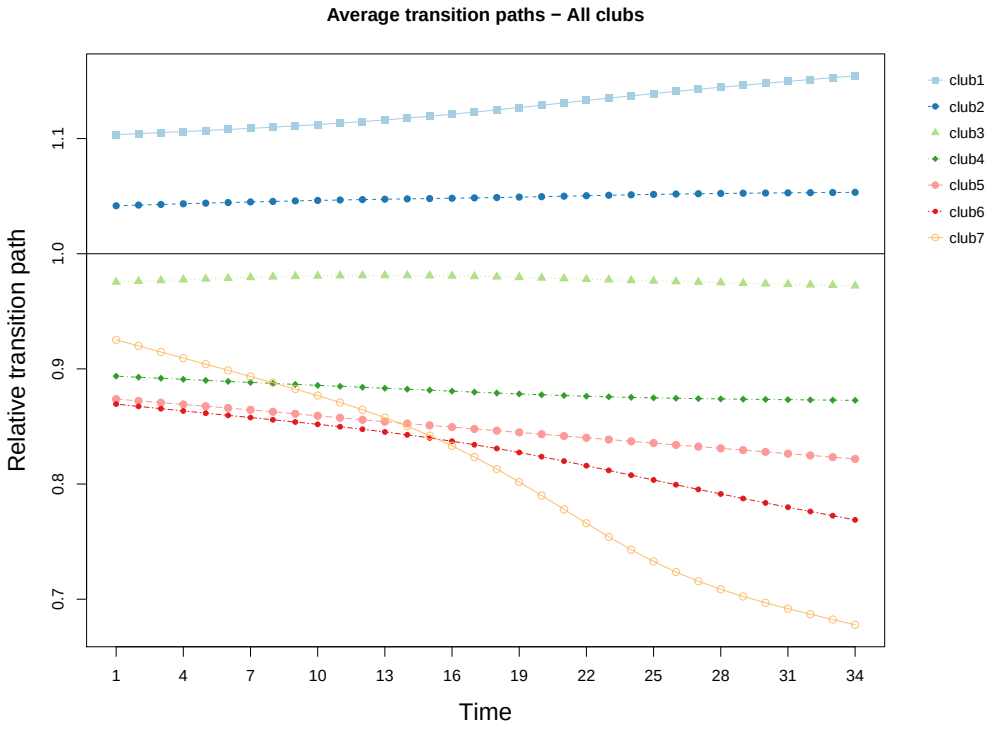


Figure 3: Average transition paths for each convergence club.

Finally, we assess if it is possible to merge some clubs together by using function `mergeClubs()`. Phillips and Sul (2009) merging algorithm is chosen through the argument `mergeMethod='PS'`.

```
# Merge clusters using Phillips and Sul (2009) method
mclubs <- mergeClubs(clubs, mergeMethod='PS')
summary(mclubs)

# Number of convergence clubs: 6
# Number of divergent units: 0
#
#         | merged clubs   | # of regions   | beta      | std.err   | tvalue
# -------- --------------- --------------- ---------- ---------- ----------
#  club1  | clubs: 1       |   50          |   0.382   |   0.041   |   9.282
#  club2  | clubs: 2       |   30          |   0.24    |   0.035   |   6.904
#  club3  | clubs: 3       |   21          |   0.11    |   0.032   |   3.402
#  club4  | clubs: 4, 5    |   38          |  -0.044   |   0.07    |  -0.636
#  club5  | clubs: 6       |   11          |   1.003   |   0.166   |   6.024
#  club6  | clubs: 7       |   2           |  -0.47    |   0.842   |  -0.559
```

According to the Phillips and Sul merging algorithm, former clubs 4 and 5 have been merged forming a new club (club 4), which now includes 38 Countries (24+14).

Conclusions

In this paper, we have discussed the implementation in R of the Phillips and Sul (2007, 2009) clustering procedure by presenting the **ConvergenceClubs** package. The package allows for simple and intuitive application of this methodology, which has become predominant in the analysis of the convergence patterns of economies due to its positive attributes. We have provided functions to perform the log–t test and cluster units, as well as to merge existing clubs. We have also described functions to summarise and plot the information obtained through the application of the clustering algorithm, as well as a detailed example of the package functionalities.

Bibliography

Aptech Systems. The GAUSS system version 17, 2016. URL https://store.aptech.com/gauss-platform-category.html. Maple Valley, WA. [p142]

M. Balcilar. *mFilter: Miscellaneous Time Series Filters*, 2018. URL https://CRAN.R-project.org/package=mFilter. R package version 0.1-4. [p146]

W. J. Baumol. Productivity growth, convergence, and welfare: What the long–run data show. *The American Economic Review*, 76(5):1072–1085, 1986. URL http://www.jstor.org/stable/1816469. [p142]

L. Corrado, M. Weeks, and R. Martin. Identifying and Interpreting Regional Convergence Clusters Across Europe. *The Economic Journal*, 115(502):C133–C160, 03 2005. URL https://doi.org/10.1111/j.0013-0133.2005.00984.x. [p142]

K. Du. Econometric convergence test and club clustering using Stata. *The Stata Journal*, 17(4):882–900, 2018. URL https://doi.org/10.1177/1536867X1801700407. [p142, 148]

J. Le Gallo and S. Dall'Erba. Spatial econometric analysis of the evolution of the European regional convergence process, 1980–1999. Technical report, EconWPA, 2005. URL https://econwpa.ub.uni-muenchen.de/econ-wp/urb/papers/0311/0311001.pdf. [p142]

G. Myrdal. *Economic theory and underdeveloped regions*. Gerald Duckworth & Co Ltd, 1957. [p142]

P. C. B. Phillips and D. Sul. Transition modeling and econometric convergence tests. *Econometrica*, 75 (6):1771–1855, 2007. URL https://doi.org/10.1111/j.1468-0262.2007.00811.x. [p142, 143, 144, 146, 150]

P. C. B. Phillips and D. Sul. Economic transition and growth. *Journal of Applied Econometrics*, 24(7): 1153–1185, 2009. URL https://doi.org/10.1002/jae.1080. [p142, 143, 144, 146, 148, 150]

D. T. Quah. Regional convergence clusters across Europe. *European Economic Review*, 40(3):951 – 958, 1996. URL https://doi.org/10.1016/0014-2921(95)00105-0. Papers and Proceedings of the Tenth Annual Congress of the European Economic Association. [p142]

J. Schnurbus, H. Haupt, and V. Meier. Economic transition and growth: A replication. *Journal of Applied Econometrics*, 32(5):1039–1042, 2017. URL https://doi.org/10.1002/jae.2544. [p142, 148]

R. Sichera and P. Pizzuto. *ConvergenceClubs: Finding Convergence Clubs*, 2019. URL https://cran.r-project.org/web/packages/ConvergenceClubs. R package version 2.2.1. [p142, 146]

R. M. Solow. A Contribution to the Theory of Economic Growth. *The Quarterly Journal of Economics*, 70 (1):65–94, 02 1956. URL https://doi.org/10.2307/1884513. [p142]

StataCorp. Stata statistical software: release 15, 2017. URL https://www.stata.com. College Station, TX: StataCorp LLC. [p142]

K. von Lyncker and R. Thoennessen. Regional club convergence in the EU: evidence from a panel data analysis. *Empirical Economics*, 52(2):525–553, Mar 2017. URL https://doi.org/10.1007/s00181-016-1096-2. [p142, 145, 146]

Roberto Sichera
Department of Economics,
Business and Statistics (SEAS)
Viale delle Scienze, Ed. 13
Palermo, Italy
ORCID: 0000-0001-5307-6656
roberto.sichera@unipa.it

Pietro Pizzuto
Department of Economics,
Business and Statistics (SEAS)
Viale delle Scienze, Ed. 13
Palermo, Italy
ORCID: 0000-0001-5055-8916
pietro.pizzuto02@unipa.it

PPCI: an R Package for Cluster Identification using Projection Pursuit

by David P. Hofmeyr and Nicos G. Pavlidis

Abstract This paper presents the **R** package **PPCI** which implements three recently proposed projection pursuit methods for clustering. The methods are unified by the approach of defining an optimal hyperplane to separate clusters, and deriving a projection index whose optimiser is the vector normal to this separating hyperplane. Divisive hierarchical clustering algorithms that can detect clusters defined in different subspaces are readily obtained by recursively bi-partitioning the data through such hyperplanes. Projecting onto the vector normal to the optimal hyperplane enables visualisations of the data that can be used to validate the partition at each level of the cluster hierarchy. Clustering models can also be modified in an interactive manner to improve their solutions. Extensions to problems involving clusters which are not linearly separable, and to the problem of finding maximum hard margin hyperplanes for clustering are also discussed.

Introduction

Clustering refers to the problem of identifying distinct groups (clusters) of relatively homogeneous points within a collection of data, with no explicit knowledge about the group associations of any of the points. Various definitions of what constitutes a cluster have led to a multitude of clustering algorithms (Jain et al., 1999), with no universal consensus and no definition which is appropriate for all applications. Without a ground truth solution clusters must be determined by the relative spatial relationships between points. However, the spatial structure can be less informative for determining clusters in the presence of irrelevant/noisy features, as well as correlations among subsets of features. Such characteristics abound especially in high dimensional applications, and make the clustering problem especially challenging. To accurately cluster such data sets it becomes necessary to identify subspaces which allow a strong separation of clusters. The best subspace to separate clusters will clearly depend on the cluster definition employed, and moreover a single subspace of fixed dimension may not allow a complete separation of all clusters.

A principled approach to finding high quality subspaces for clustering is via projection pursuit. Projection pursuit refers to a class of dimension reduction techniques which seek to optimise over all linear projections of the data a given measure of interestingness, called a *projection index* (Huber, 1985). Principal Component Analysis (PCA) is arguably the most popular projection pursuit method. In PCA the projection index can be formulated as the variance of the projected data. This index is known to be maximised by the eigenvector associated with the largest eigenvalue of the covariance matrix. For most other projection indices no closed form solution is available and these objectives are instead numerically optimised. Although PCA has been successfully applied in numerous clustering problems, there is no guarantee that any number of principal components will be relevant for preserving/exposing cluster structure. This is unsurprising as it is trivial to construct data sets in which clusters are only separated along directions of low data variability. As will be seen in the remainder, when the clustering objective is included in the projection index, substantial improvements can be made.

As far as we are aware the only existing **R** package which combines projection pursuit and clustering is **ProjectionBasedClustering** (Thrun et al., 2018). **ProjectionBasedClustering** provides both linear and non-linear dimension reduction techniques, including Independent Component Analysis (Hyvärinen et al., 2004, ICA) and t-distributed Stochastic Neighbour Embedding (Maaten and Hinton, 2008, t-SNE). None of these incorporates a clustering criterion directly into the dimension reduction formulation. As a result there is no guarantee that the lower dimensional embeddings of the data will exhibit any cluster structure. Moreover different dimension reduction techniques may lead to very different low dimensional embeddings. This is problematic from the user's perspective. Even if the user knows the type of clusters which are of interest it is unclear which dimension reduction technique is most appropriate for their problem. The **subspace** package (Hassani and Hansen, 2015) also performs projection based clustering, including methods such as CLIQUE (Agrawal et al., 1998) and SubClu (Kailing et al., 2004). The approach adopted by these methods differs fundamentally from ours in that clusters are defined through grid cells which have high data density when projected onto multiple axes of the input space. There is thus no search for an optimal subspace/projection of the data.

In this paper we present the **R** package **PPCI** which provides implementations of three recently developed projection pursuit methods for clustering. The projection indices underlying these methods are based on three popular clustering objectives, including those underlying k-means; density cluster-

ing; and clustering by normalised graph cuts. With no guarantee of the existence of a single subspace to distinguish all clusters, we instead adopt the approach of repeatedly identifying a single univariate subspace which allows the separation of at least one cluster from the remainder. The separations of clusters are thus induced by hyperplanes in the original space. The overall clustering model has a divisive hierarchical structure, in which each cluster is formed by the intersection of finitely many half-spaces.

The **PPCI** package can be installed from the Comprehensive R Archive Network (CRAN) from within the **R** console:

```
> install.packages("PPCI")
> library(PPCI)
```

The remaining paper contains a formulation of the general problem of divisive hierarchical clustering using projection pursuit, and a description of the three methods included in **PPCI**. Thereafter we provide instructive examples and important extensions of these approaches.

Projection pursuit, hyperplanes and divisive hierarchical clustering

In this section we introduce a general framework for finding optimal univariate projections for separating clusters. Since the induced cluster boundaries are hyperplanes with normal vector equal to the optimal projection vector, we approach this problem from the point of view of defining optimal hyperplanes to bi-partition the data, and from these derive associated projection indices. We then discuss briefly how to combine such hyperplanes to produce a divisive hierarchical clustering model.

Finding optimal hyperplanes via projection pursuit

To begin with, assume the data set consists of a finite set of vectors in $\mathbb{R}^d$, $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$. For a unit-length vector $\mathbf{v} \in \mathbb{B}^d = \left\{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x}\|_2 = 1\right\}$, we use $\mathbf{v}^\top \mathcal{X} = \{\mathbf{v}^\top \mathbf{x}_i\}_{i=1}^n$ to denote the projection of $\mathcal{X}$ onto $\mathbf{v}$. Now, a hyperplane in $\mathbb{R}^d$ is a translated subspace of co-dimension one, that is parameterised by a pair $(\mathbf{v}, b) \in \mathbb{B}^d \times \mathbb{R}$, as the set,

$$H(\mathbf{v}, b) = \left\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{v}^\top \mathbf{x} = b\right\}.$$

A hyperplane that intersects the interior of the convex hull of $\mathcal{X}$ induces a binary partition based on the half-space to which each observation is allocated,

$$\begin{aligned}
\mathcal{X} &= \mathcal{X}^+_{\mathbf{v},b} \cup \mathcal{X}^-_{\mathbf{v},b}, \\
\mathcal{X}^+_{\mathbf{v},b} &:= \{\mathbf{x} \in \mathcal{X} | \mathbf{v}^\top \mathbf{x} \geqslant b\}, \\
\mathcal{X}^-_{\mathbf{v},b} &:= \{\mathbf{x} \in \mathcal{X} | \mathbf{v}^\top \mathbf{x} < b\}.
\end{aligned}$$

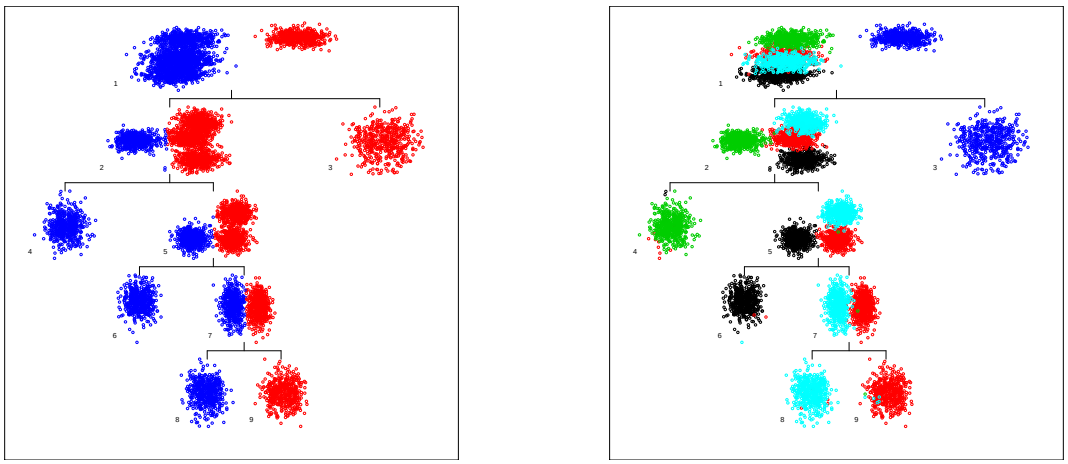
Hyperplanes are linear cluster separators as the boundary between the two clusters is defined by the linear equation $\mathbf{v}^\top \mathbf{x} = b$. It is simple to construct examples where linear cluster separators are too restrictive, however hyperplanes have been very successful in accurately separating clusters in many real world applications, and especially in high dimensional data sets (Boley, 1998; Tasoulis et al., 2010, 2012; Zhang et al., 2009).

One of the benefits of employing hyperplanes for clustering is that projecting onto a vector (in this case the vector normal to the hyperplane) achieves the maximum reduction of dimensionality. Furthermore the associated clustering problems within the corresponding one-dimensional subspace become tractable. Now, if the *projection vector*, $\mathbf{v}$, is a combination of features that are not relevant for distinguishing clusters then there will be no hyperplane orthogonal to $\mathbf{v}$ which induces a high quality binary partition of $\mathcal{X}$. If instead $\mathbf{v}$ is a combination of features along which clusters are separable, then there will exist a b for which $H(\mathbf{v}, b)$ induces a high quality binary partition of $\mathcal{X}$.

Let Q be a measure of quality of a binary partition, where quality can be measured either in terms of a high degree of clusterability, or low degree of connectedness between the two components. The quality of a hyperplane $H(\mathbf{v}, b)$ can then be written as,

$$\phi(\mathbf{v}, b|\mathcal{X}) = Q(\mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}, \mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}).$$

When the number of clusters is large, the objective $\phi(\mathbf{v}, b|\mathcal{X})$ has numerous local optima. As a result optimising $\phi(\mathbf{v}, b|\mathcal{X})$ directly with respect to both parameters, $\mathbf{v}$ and b, can lead to partitions of



(a) Without cluster labels

(b) With true cluster labels

Figure 1: Divisive hierarchical clustering model for simulated data set based on Maximum Clusterability Divisive Clustering algorithm

only moderate quality. To mitigate this we use a projection pursuit formulation in which the projection index, $\Phi(\mathbf{v}|\mathcal{X})$ in Eq. (2), is the quality of the best hyperplane with normal vector $\mathbf{v}$. If Q is a measure of clusterability then the optimal projection vector, $\mathbf{v}_{\text{opt}}$, maximises $\Phi(\mathbf{v}|\mathcal{X})$ over all unit-length vectors,

$$\mathbf{v}_{\text{opt}} = \underset{\mathbf{v}\in\mathbb{B}^d}{\operatorname{argmax}}\, \Phi(\mathbf{v}|\mathcal{X}), \tag{1}$$

$$\Phi(\mathbf{v}|\mathcal{X}) = \max_{b\in\mathbb{R}} \phi(\mathbf{v}, b|\mathcal{X}). \tag{2}$$

If instead Q measures the connectedness between the elements of the binary partition then Eqs. (1) and (2) become minimisation problems. Since Q is arbitrary this provides a framework for deriving projection indices for essentially any clustering objective. By considering only the best hyperplane orthogonal to a candidate projection vector this formulation allows b to change discontinuously, and hence avoids many local optima of $\phi(\mathbf{v}, b|\mathcal{X})$. Since $\Phi(\mathbf{v}|\mathcal{X})$ is a maximum of $\phi(\mathbf{v}, \cdot|\mathcal{X})$ it is not guaranteed to be continuously differentiable everywhere. However, with the exception of pathological cases in which the elements of $\mathcal{X}$ have a very precise geometry, it is difficult to construct examples in which $\Phi(\mathbf{v}|\mathcal{X})$ is not continuously differentiable almost everywhere. Lewis and Overton (2013) strongly advocate using BFGS to optimise such non-smooth locally Lipschitz functions, over more computationally expensive methods like gradient sampling (Burke et al., 2005). We therefore use the existing optimised implementation of BFGS provided in **R**'s base **stats** package.

Divisive hierarchical clustering using hyperplanes

A divisive hierarchical clustering model is constructed by recursively applying a partitioning method to (subsets of) the data set. Using hyperplanes to recursively bi-partition the data yields a cluster hierarchy with a binary tree structure. The leaves of the tree correspond to the final clusters, with each defined as the intersection of a finite number of half spaces. If the number of clusters is specified then the only requirements for constructing a divisive hierarchical clustering model are the cluster quality measure Q, which leads to the projection index Φ, and an indexing (or selection) strategy that determines which leaf (cluster) of the current hierarchy should be partitioned at each step of the recursion. Examples of indexing strategies include splitting the largest cluster, $i = \operatorname{argmax}_{j\in\text{leaves}(T)}\{|\mathcal{C}_j|\}$, where T is the cluster hierarchy and $\mathcal{C}_j \subset \mathcal{X}$ is the data subset allocated to leaf j, or splitting the leaf whose bi-partition achieves the maximum clusterability, $i = \operatorname{argmax}_{j\in\text{leaves}(T)} \Phi(\mathbf{v}_{\text{opt}}|\mathcal{C}_j)$.

Example 1. *We apply the Maximum Clusterability Divisive Clustering algorithm (Hofmeyr and Pavlidis, 2015, MCDC) to 10 dimensional simulated data arising from a Gaussian mixture containing 5 components, each representing a cluster. The function* `mcdc()` *implements this algorithm and is described in detail later. The resulting hierarchical clustering model is visualised through the* `tree_plot()` *function, and depicted in Figure 1. Details of the arguments of this function are given in Table 1*

Each scatterplot in Figure 1 depicts the data subset assigned to the corresponding node in the hierarchy projected into a two-dimensional subspace. The horizontal axis corresponds to the optimal projection vector, $\mathbf{v}_{\text{opt}}$ in Eq. (1), while the vertical axis is the direction of maximum variance orthogonal to $\mathbf{v}_{\text{opt}}$. If the `labels`

Argument	Description
`sol`	A clustering solution from any clustering algorithm in **PPCI**.
`labels`	(optional) A vector of class labels. Points with the same label are plotted with the same colour.
`node.numbers`	(optional) Logical. If `node.numbers==TRUE` then the order in which the nodes were added to the hierarchy is indicated in the plot.

Table 1: Arguments accepted by `tree_plot` function.

argument is not specified, as in Figure F.1(a), colours indicate the binary partitions induced by the corresponding hyperplane separator. Since leaf nodes correspond to clusters the projected data are assigned a single colour. When the `labels` *argument is specified the points in each scatterplot are coloured according to their corresponding labels, as shown in Figure F.1(b).*

```
> set.seed(1)
> means <- matrix(rnorm(50), ncol = 10)
> X <- matrix(, 2500, 10)
> for(i in 1:5) X[((i-1)*500+1):(i*500),] <- t(matrix(rnorm(5000, 0, .5), ncol = 500) + means[i,])
> sol <- mcdc(X, 5)
> tree_plot(sol)
> tree_plot(sol, labels = rep(1:5, each = 500))
```

Clustering and Projection Pursuit in PPCI

In this section we describe the projection pursuit and clustering methods provided in **PPCI**, as well as their implementation in **R**. The main functions are described in Table 2. The projection pursuit algorithms find optimal projections and corresponding optimal hyperplanes to bi-partition the data, while the clustering algorithms obtain a complete divisive hierarchical clustering model.

Proj. Pursuit	Clustering	Description of method
`mdh`	`mddc`	Uses hyperplanes with minimal integrated density to separate clusters. Such hyperplanes avoid intersecting high density clusters, as defined in connection with density clustering. The implementation is based on the method of Pavlidis et al. (2016).
`mch`	`mcdc`	Uses hyperplanes which maximise the variance ratio clusterability of the induced binary partition. This approach is closely related to the k-means objective. The implementation is based on that of Hofmeyr and Pavlidis (2015)
`ncuth`	`ncutdc`	Uses hyperplanes with minimum normalised cut measured across them, using the method of Hofmeyr (2017). This leads to partitions with low between and high within cluster similarity.

Table 2: Projection pursuit and clustering algorithms in **PPCI**.

The only mandatory argument for the projection pursuit methods (`mdh`, `mch` and `ncuth`) is the data matrix `X`, while the clustering algorithms (`mddc`, `mcdc` and `ncutdc`) require both the data matrix, `X`, and the number of clusters to extract, `K`. In addition all methods accept numerous optional arguments which allow the user to modify the parameters associated with the projection pursuit. When omitted all optional arguments are assigned default values. Table 3 provides a complete list of the arguments for these methods. For full details use the `help()` command within the **R** console, providing the relevant function name as argument, e.g. '`help(mdh)`'.

The output of the projection pursuit algorithms (`mdh`, `mch` and `ncuth`) is an unnamed list of length m, where m is the number of initialisations considered. Each element of the output is a named list containing the details of the corresponding optimal solution. The output of the clustering algorithms (`mddc`, `mcdc` and `ncutdc`) is a named list containing details of the clustering solution. The most important components of the output are the `$cluster` field, which contains the cluster assignment vector, and the `$Nodes` field, which is an unnamed list in which each element contains the details of the corresponding node in the hierarchical clustering model.

Argument	Description
`X`	Data matrix ($n \times d$) with observations row-wise.
`K`	(`mddc`, `mcdc` and `ncutdc` only) Number of clusters to extract.
`v0`	(optional) Used to obtain data driven initialisations for projection pursuit. A function accepting a single matrix argument (the data being partitioned) and returning a $d \times m$ matrix (for user chosen m). Each column of the output of `v0` is used as an initialisation. Projection pursuit algorithms (`mdh`, `mch` and `ncuth`) also accept directly the $d \times m$ matrix whose columns are used as initialisations.
`split.index`	(optional. `mddc`, `mcdc` and `ncutdc` only) Used to determine the order in which clusters are split (in decreasing order of split indices). A numeric valued function of the optimal projection vector (`v`), the data matrix (`X`) and parameter list `P`.
`minsize`	(optional) Integer valued minimum cluster size.
`verb`	(optional) Verbosity level. Values greater than zero produce plots to illustrate progress of the algorithm.
`labels`	(optional) Vector of class labels. Only used in plots produced for verbosity levels greater than zero. Does not influence clustering/projection pursuit itself.
`maxit`	(optional) Maximum number of iterations in projection pursuit.
`ftol`	(optional) Relative tolerance level for optimisation.
`bandwidth`	(optional. `mdh` and `mddc` only) Used to determine data driven bandwidth parameter for kernel density estimator. In `mddc` a function taking only a matrix argument (the data being partitioned) and returning a scalar. In `mdh` a scalar to be used directly in kernel estimator.
`alphamin`	(optional. `mdh` and `mddc` only) Initial (scaled) bound on the distance of the optimal hyperplane from the mean of the data being partitioned (α in Eq. (5)).
`alphamax`	(optional. `mdh` and `mddc` only) Final/maximum (scaled) bound on the distance of the optimal hyperplane from the mean of the data being partitioned (α in Eq. (5)).
`s`	(optional. `ncuth` and `ncutdc` only) Used to determine data driven scaling parameter for pairwise similarities. In `ncutdc` a function taking only a matrix argument (the data being partitioned) and returning a scalar. In `ncuth` a scalar to be used directly in similarity calculations.

Table 3: Arguments accepted by clustering and projection pursuit functions.

In the following subsections we provide a brief description of each of the methods implemented in **PPCI**, as well as examples illustrating the usage in **R**.

Minimum Density Hyperplanes (`mdh` and `mddc`)

In density clustering the data set, $\mathcal{X}$, is assumed to represent a sample of realisations of a random variable $\mathbf{X}$ on $\mathbb{R}^d$, with unknown probability density function p. Clusters are defined either as "regions of attraction" of each mode of p (Comaniciu and Meer, 2002; Stuetzle and Nugent, 2010; Menardi and Azzalini, 2014); or as maximally connected components of the level sets $\left\{\mathbf{x} \in \mathbb{R}^d \mid p(\mathbf{x}) > \lambda\right\}$, for a suitable choice of the level parameter λ (Hartigan, 1975; Cuevas et al., 2001; Rinaldo and Wasserman, 2010). An immediate consequence of this definition is that cluster boundaries traverse regions of low probability density, known as the *low density separation* assumption (Chapelle and Zien, 2005). A number of influential methods for clustering and semi-supervised classification use low density hyperplanes, including maximum margin clustering (Xu et al., 2004) and semi-supervised support vector machines (Chapelle and Zien, 2005; Chapelle et al., 2008). Although all these methods use maximum margin hyperplanes to approximate low density separators, the consistency of this approach remains an open question (Ben-David et al., 2009).

The Minimum Density Hyperplane (Pavlidis et al., 2016, MDH) directly estimates the hyperplane with minimum density based on an estimated density $\hat{p}$. Following Ben-David et al. (2009) the estimated density on $H(\mathbf{v}, b)$, $\hat{I}(\mathbf{v}, b)$ in Eq. (3), is defined as the surface integral of $\hat{p}$ on $H(\mathbf{v}, b)$. By estimating p through a Gaussian kernel mixture $\hat{I}(\mathbf{v}, b)$ can be computed exactly using only the

Argument	Description
`sol`	A solution from any projection pursuit algorithm in **PPCI**.
`X`	The data matrix used to obtain `sol`.
`labels`	(optional) A vector of class labels. Points with the same label are plotted with the same colour.

Table 4: Arguments accepted by `hp_plot` function.

projections of $\mathcal{X}$ onto $\mathbf{v}$,

$$\hat{p}(\mathbf{x}) = \frac{1}{n(2\pi h^2)^{d/2}} \sum_{i=1}^{n} e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2h^2}} \Rightarrow \hat{I}(\mathbf{v}, b) := \int_{H(\mathbf{v},b)} \hat{p}(\mathbf{x}) d\mathbf{x} = \frac{1}{n\sqrt{2\pi h^2}} \sum_{i=1}^{n} e^{-\frac{(b-\mathbf{v}^\top \mathbf{x}_i)^2}{2h^2}}. \tag{3}$$

It is clear that for any vector $\mathbf{v}$ one has $\lim_{b\to\pm\infty} \hat{I}(\mathbf{v}, b) = 0$, that is, hyperplanes passing through the tail of the estimated density can be made to have arbitrarily small density. To avoid such solutions which are not meaningful for clustering, a penalty term is added to $\hat{I}(\mathbf{v}, b)$ to constrain the distance of the optimal hyperplane from the mean of the data (which without loss of generality can be assumed to be zero). Specifically, the Minimum Density Projection Pursuit (Pavlidis et al., 2016, MDP^2) algorithm is based on the following optimisation problem,

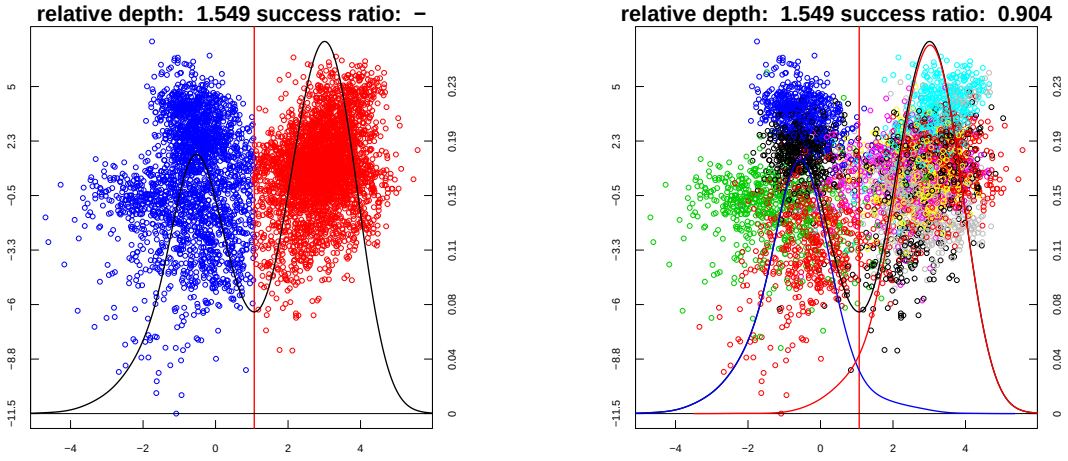
$$\min_{\mathbf{v}} \Phi(\mathbf{v}|\mathcal{X}) = \min_{b\in\mathbb{R}} \phi(\mathbf{v}, b|\mathcal{X}), \tag{4}$$

$$\phi(\mathbf{v}, b|\mathcal{X}) = \hat{I}(\mathbf{v}, b) + C \max\{0, -\alpha\sigma_{\mathbf{v}} - b, b - \alpha\sigma_{\mathbf{v}}\}^{1+\epsilon}, \tag{5}$$

for any $\epsilon \in (0, 1)$, and a constant C. The term $\sigma_{\mathbf{v}}$ denotes the standard deviation of the projected data $\mathbf{v}^\top\mathcal{X}$. The value of α controls the distance of the optimal solution to the mean of the data, and is dynamically adjusted during the execution of the algorithm.

Example 2. *In this example we first apply* `mdh()` *and then* `mddc()` *to obtain a binary partition, and then a complete clustering of the optical recognition of handwritten digits (optidigits) data set from the UCI repository (Lichman, 2013). The data set we consider combines the training and test set data sets in UCI and contains 5620 observations. Observations correspond to vectorised images of handwritten digits 0–9. The images have been compressed to 8×8 pixels resulting in a 64 dimensional data set.*

```
> data(optidigits)
> sol <- mdh(optidigits$x)
> hp_plot(sol, optidigits$x)
> hp_plot(sol, optidigits$x, labels = optidigits$c)
> success_ratio(sol[[1]]$cluster, optidigits$c)
 [1] 0.9040637
```



(a) Labels not specified (b) Labels specified

Figure 2: Two-dimensional visualisation of binary partition of the optical handwritten digits recognition data set through `mdh()`.

The function `hp_plot` *provides a visualisation of hyperplane separators obtained from any projection pursuit algorithm in* **PPCI**. *The arguments for this function are described in Table 4. The output of the first call to this function is depicted in Figure F.2(a). If the* `labels` *argument is not specified the points are coloured according to the half space to which they belong. The projected density is depicted with a solid black line, with its scale depicted on the right vertical axis. If the* `labels` *argument is specified, as in the second call to* `hp_plot` *above, the colours of the projected points represent the true cluster assignments, as shown in Figure F.2(b). In this case each true cluster can be assigned to the half space that contains the majority of its observations. The projected density can then be seen as a two component mixture density arising from the clusters assigned to the two half spaces. The red and blue solid lines illustrate these two component densities. The quality of a binary partition of a data set containing more than two clusters can be evaluated through the success ratio (Pavlidis et al., 2016). The success ratio is reported in the title of Figure F.2(b), and can be computed through the* `success_ratio(clusters,labels)` *function.*

Next we use the `mddc()` *algorithm to obtain a complete clustering of the optidigits data set.*

```
> sol <- mddc(optidigits$x, 10)
> tree_plot(sol)
> cluster_performance(sol$cluster, optidigits$c)
 adj.rand    purity v.measure       nmi
 0.6451702 0.7919929 0.7202152 0.7202187
```

The visualisation of the cluster hierarchy through the `tree_plot` *function has been discussed in Example 1). The* `cluster_performance(clusters,labels)` *function implements four external cluster validity measures: purity (Zhao and Karypis, 2004), normalised mutual information (NMI) (Strehl and Ghosh, 2002), adjusted Rand index (Hubert and Arabie, 1985), and V-measure (Rosenberg and Hirschberg, 2007).*

Maximum clusterability clustering (`mch` and `mcdc`)

The term *clusterability* has been used to refer to the strength, or conclusiveness, of the cluster structure in a data set (Ackerman and Ben-David, 2009). The *variance ratio* clusterability is defined as (Zhang, 2001),

$$\max_{\substack{\mathcal{X}=\mathcal{C}_1\cup\cdots\cup\mathcal{C}_k\\ \mathcal{C}_i\cap\mathcal{C}_j=\emptyset, i\neq j}} \frac{\sum_{i=1}^{k} |\mathcal{C}_i| \, \|\boldsymbol{\mu}_{\mathcal{C}_i} - \boldsymbol{\mu}\|^2}{\sum_{i=1}^{k} \sum_{j:\mathbf{x}_j\in\mathcal{C}_i} \|\mathbf{x}_j - \boldsymbol{\mu}_{\mathcal{C}_i}\|^2}, \tag{6}$$

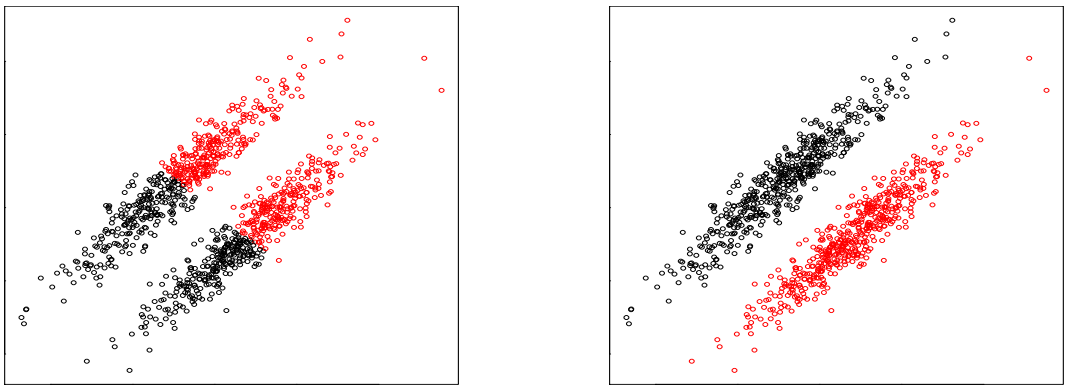
where $\boldsymbol{\mu}_{\mathcal{C}_i} = \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{x}_j\in\mathcal{C}_i} \mathbf{x}_j$ is the mean of the i-th cluster and $\boldsymbol{\mu} = \frac{1}{n}\sum_{j=1}^{n} \mathbf{x}_j$ is the mean of the data. This represents the ratio of the between and within cluster scatter values, and is related to Fisher's discriminant function used in Linear Discriminant Analysis. It is easy to show that the solution that maximises the variance ratio clusterability also minimises the k-means objective. Determining the clusterability of a data set is therefore of the same complexity as identifying the optimal k-means solution, which is known to be NP-hard. This renders clusterability in the general setting primarily of theoretical interest. In the univariate setting, however, the k-means solution becomes tractable and so the variance ratio may be used practically to define a projection index. This is important as the values of the variance ratio and of the k-means objective are largely determined by directions in which the data exhibit high variability. This is problematic when the clusters are not separable along these directions. The Maximum Clusterability Divisive Clustering algorithm (MCDC) recursively partitions a data set by identifying the univariate subspace that maximises the variance ratio of the projected data. Since the variance ratio measure is scale invariant, this objective is able to overcome situations where directions which separate clusters are of relatively low variability.

It is straightforward to see that the optimal partition of $\mathbf{v}^\top\mathcal{X}$ based on the variance ratio splits the projected data at a point. The projection index can thus be written as

$$\Phi(\mathbf{v}|\mathcal{X}) = \max_b \phi(\mathbf{v}, b|\mathcal{X}),$$

$$\phi(\mathbf{v}, b|\mathcal{X}) := \frac{\left|\mathcal{X}^+_{\mathbf{v},b}\right| \left(\mu_{\mathbf{v}^\top\mathcal{X}^+_{\mathbf{v},b}} - \mu_{\mathbf{v}^\top\mathcal{X}}\right)^2 + \left|\mathcal{X}^-_{\mathbf{v},b}\right| \left(\mu_{\mathbf{v}^\top\mathcal{X}^-_{\mathbf{v},b}} - \mu_{\mathbf{v}^\top\mathcal{X}}\right)^2}{\frac{n}{n-1}\sum_{i=1}^{n}\left(\mathbf{v}^\top\mathbf{x}_i - \mu_{\mathbf{v}^\top\mathcal{X}}\right)^2 + \sum_{\mathbf{x}_i\in\mathcal{X}^+_{\mathbf{v},b}}\left(\mathbf{v}^\top\mathbf{x}_i - \mu_{\mathbf{v}^\top\mathcal{X}^+_{\mathbf{v},b}}\right)^2 + \sum_{\mathbf{x}_j\in\mathcal{X}^-_{\mathbf{v},b}}\left(\mathbf{v}^\top\mathbf{x}_j - \mu_{\mathbf{v}^\top\mathcal{X}^-_{\mathbf{v},b}}\right)^2}.$$

The term $\frac{n}{n-1}\sum_{i=1}^{n}\left(\mathbf{v}^\top\mathbf{x}_i - \mu_{\mathbf{v}^\top\mathcal{X}}\right)^2$ in the denominator is added to ensure that $\Phi(\mathbf{v}, b|\mathcal{X})$ is bounded, without affecting the optimal solution.



(a) *k*-means solution (b) MCDC solution

Figure 3: Data set containing two well distinguished clusters. High within cluster variability along the principal axis dominates the between cluster variability causing *k*-means to fail, (a). MCDC on the other hand is able to recover the true solution, (b).

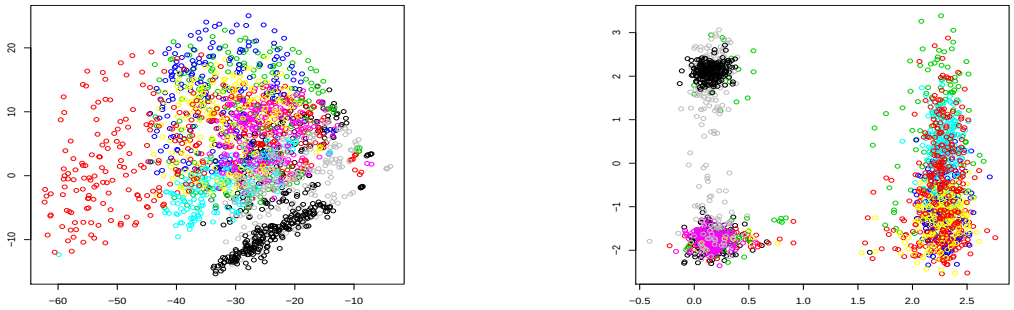
Example 3. *We consider a simple two-dimensional example, seen in Figure 3, to illustrate the effectiveness of using variance ratio as a projection index. This example contains two well separated and easily distinguished clusters. However, the elongation of the clusters along their principal axes means that the variability of the data along this direction dominates the variability in the direction which separates the clusters. The k-means solution, Figure F.3(a), is severely influenced by this fact. On the other hand by maximising the variance ratio of the projected data the true clusters can be recovered, Figure F.3(b).*

```
> set.seed(1)
> S <- matrix(c(1, .7, .7, 1), 2, 2)
> X <- matrix(rnorm(2000), ncol = 2) %*% S
> X[,1] <- X[,1] + rep(c(.8, -.8), each = 500)
> X[,2] <- X[,2] + rep(c(-.8, .8), each = 500)
>
> km <- kmeans(X, 2, nstart = 10)
> sol <- mch(X)
>
> par(mfrow = c(1, 2))
> plot(X, col = km$cluster, main = "kmeans solution")
> plot(X, col = sol[[1]]$cluster, main = "MCH solution")
```

Example 4. *We next consider a face recognition task with ten clusters in which centroid based algorithms like k-means perform suboptimally because the within cluster covariance structure dominates the between cluster separation. We apply both k-means and MCDC to a subset of the Yale face database B (Georghiades et al., 2001), containing 2000 portrait images of 10 different human subjects in different lighting conditions and with different orientations. The images have been compressed to* 30×20 *pixels and vectorised, resulting in 600 dimensional data.*

```
> require(rARPACK)
> set.seed(1)
> data(yale)
> km <- kmeans(yale$x, 10, nstart = 10)
> sol <- mcdc(yale$x, 10)
>
> pc <- rARPACK::eigs_sym(cov(yale$x), 2)$vectors
> par(mfrow = c(1, 2))
> plot(yale$x%*%pc, xlab = "PC1", ylab = "PC2",
     main = "Principal component projection", col = yale$c)
> plot(yale$x%*%cbind(sol$Nodes[[1]]$v, sol$Nodes[[2]]$v), xlab = "VR1",
     ylab = "VR2", main = "Projection based on variance ratio", col = yale$c)
>
> cluster_performance(sol$cluster, yale$c)
 adj.rand    purity v.measure       nmi
 0.7206458 0.8220000 0.8309754 0.8309877
> cluster_performance(km$cluster, yale$c)
```

```
adj.rand    purity v.measure       nmi
0.5006819 0.6675000 0.6798313 0.6800078
```



(a) Principal component projection

(b) Projection based on variance ratio

Figure 4: Two dimensional projections of vectorised images taken from the Yale faces database B

Figure 4 shows two dimensional projections of these data. In Figure F.4(a) the data are projected along the first two principal components, while in Figure F.4(b) the projection vectors are taken as the optimal projection vectors from the first two nodes in the MCDC model. Although there is evidence of the presence of multiple clusters in the principal component projections, these high variance projections do not admit a separation of any clusters from the rest. In contrast the projections based on variance ratio show very strong evidence of at least three clusters. The remaining clusters are identified further down the hierarchical model. The presence of elongated cluster shapes, evident in Figure F.4(a), prevents k-means from achieving performance as high as that of MCDC.

Minimum normalised cut hyperplanes (`ncuth` and `ncutdc`)

Arising from graph partitioning algorithms, the normalised cut objective has become popular for data clustering (von Luxburg, 2007). The normalised cut (NCut) associated with a partition of $\mathcal{X}$ into clusters $\mathcal{C}_1, ..., C_k$ is given by (Shi and Malik, 2000),

$$\text{NCut}(\mathcal{C}_1, \ldots, \mathcal{C}_k) = \sum_{m=1}^{k} \frac{\text{Cut}(\mathcal{C}_m, \mathcal{X} \setminus \mathcal{C}_m)}{\text{volume}(\mathcal{C}_m)}, \tag{7}$$

$$\text{Cut}(\mathcal{C}, \mathcal{X} \setminus \mathcal{C}) := \sum_{\substack{i,j:\mathbf{x}_i \in \mathcal{C}, \\ \mathbf{x}_j \notin \mathcal{C}}} \text{similarity}(\mathbf{x}_i, \mathbf{x}_j), \quad \text{volume}(\mathcal{C}) := \sum_{\substack{i,j:\mathbf{x}_i \in \mathcal{C} \\ \mathbf{x}_j \in \mathcal{X}}} \text{similarity}(\mathbf{x}_i, \mathbf{x}_j). \tag{8}$$

By minimising the normalised cut one tends to find solutions for which $\text{Cut}(\mathcal{C}_m, \mathcal{X} \setminus \mathcal{C}_m)$ is small and $\text{volume}(\mathcal{C}_m)$ is large, for all m. Now, since $\text{volume}(\mathcal{C}_m) = \text{Cut}(\mathcal{C}_m, \mathcal{X} \setminus \mathcal{C}_m) + \sum_{i,j:\mathbf{x}_i,\mathbf{x}_j \in \mathcal{C}_m} \text{similarity}(\mathbf{x}_i, \mathbf{x}_j)$, where the latter term is the total internal similarity of points in $\mathcal{C}_m$, this implies that the similarity within clusters is high but the similarity between clusters is low. This makes NCut an attractive objective for clustering. However, the NCut problem is NP-hard, and unlike the k-means objective it is not straightforward to obtain a high quality locally optimal solution efficiently. Instead it is common to consider a continuous relaxation of the problem known as spectral clustering (Shi and Malik, 2000; von Luxburg, 2007). This reduces the complexity to $\mathcal{O}(kn^2)$, but this approach remains applicable only to problems of moderate size. In the linear cluster separation framework utilised here substantial improvements can be achieved.

The most critical choice in clustering algorithms based on graph cuts is the determination of pairwise similarities. Similarities are typically defined as a function of the distances between pairs of points, i.e., $\text{similarity}(\mathbf{x}_i, \mathbf{x}_j) = k(\|\mathbf{x}_i - \mathbf{x}_j\|)$, where $k : \mathbb{R}^+ \to \mathbb{R}^+$ is a decreasing function. When computing similarities between the projected data $\mathbf{v}^\top \mathcal{X}$, if the Laplace kernel, $k(x) = \exp(-|x|/\sigma)$, is used then the optimal NCut solution by a hyperplane orthogonal to $\mathbf{v}$ can be determined in $\mathcal{O}(n \log n)$ time (Hofmeyr, 2017). The associated Normalised Cut Divisive Clustering (NCUTDC) algorithm is a computationally efficient divisive clustering algorithm relying on hyperplane separators, arising from

the objectives,

$$\Phi(\mathbf{v}|\mathcal{X}) = \min_b \phi(\mathbf{v}, b|\mathcal{X}) \tag{9}$$

$$\phi(\mathbf{v}, b|\mathcal{X}) := \text{NCut}(\mathbf{v}^\top \mathcal{X}^+_{\mathbf{v},b}, \mathbf{v}^\top \mathcal{X}^-_{\mathbf{v},b}). \tag{10}$$

As in the minimum density hyperplane case, the projection pursuit problem is more conveniently formulated as a minimisation problem. It is important to note that in (Hofmeyr, 2017) this algorithm is called NCutH. To be consistent with the naming conventions used in this paper we call the divisive algorithm NCUTDC while NCUTH refers to the projection pursuit algorithm for a single hyperplane.

Example 5. *We return to the optidigits data set, and apply both spectral clustering and NCUTDC. We use the implementation of spectral clustering in the* **R** *package* **kernlab** *(Karatzoglou et al., 2004).*

```
> require(kernlab)
> data(optidigits)
>
> system.time(sol1 <- ncutdc(optidigits$x, 10))
> system.time(sol2 <- kernlab::specc(optidigits$x, 10))
> cluster_performance(sol1$cluster, optidigits$c)
   adj.rand    purity v.measure       nmi
  0.6554853 0.7870107 0.7171148 0.7171234
> cluster_performance(sol2, optidigits$c)
   adj.rand    purity v.measure       nmi
  0.3551385 0.4967972 0.4658001 0.4659070
```

NCUTDC runs orders of magnitude faster than spectral clustering, and achieves substantially higher performance.

It is important to note that because the projection index underlying NCUTDC uses similarities computed on the projected data, this algorithm cannot operate on an arbitrary graph or similarity matrix.

Modifying and validating a clustering solution

If the data are assumed to arise from a mixture of simple parametric distributions, then well established model selection methods can be used to validate a clustering model. In the absence of such strong assumptions the validation of a clustering model remains a challenging problem. Low-dimensional visualisations that reveal the cluster structure present in multivariate data can therefore be critical in determining an appropriate model. In this section we discuss how clustering models obtained through one of the hierarchical algorithms implemented in **PPCI** can be validated through visualisations, and modified interactively. The projection pursuit algorithms discussed in this paper identify vectors that maximise the clusterability of a data set, and are thus natural candidates for creating such visualisations. In previous sections we looked briefly at visualisations of both hyperplane partitions, as well as entire clustering models. Here we use these visualisations to identify important modifications and validate clustering solutions. A hierarchical clustering model can be modified either by pruning parts of the model when it is apparent that a single cluster was divided by the partition at an internal node, or by extending the model by splitting leaves which contain more than one cluster. A model can also be refined by changing the partition at an internal node via the modification of the parameters of the projection pursuit algorithm. Since this affects the entire sub-hierarchy extending from that node, the model is first pruned and then iteratively extended until the model is deemed valid. A solution may be seen to be valid if there are no internal nodes at which a single cluster is divided by the binary partition at that node, and there are no leaf nodes which contain multiple clusters. We will illustrate the modification of a clustering hierarchy by means of a detailed example.

Example 6. *In this example we again revisit the optical recognition of handwritten digits data set (Lichman, 2013). For illustrative purposes, we do not assume as we did in Example 2 that the number of clusters is known. We initially partition the data set into seven clusters using minimum density hyperplanes, and then visualise the solution using the* `tree_plot()` *function. This visualisation is shown in Figure 5.*

```
> data(optidigits)
> sol <- mddc(optidigits$x, 7)
> tree_plot(sol)
```

The first thing to notice is the partition at the node numbered 4. There is a clear presence of multiple clusters, however the separating hyperplane appears to pass through regions of relatively high density, and may not

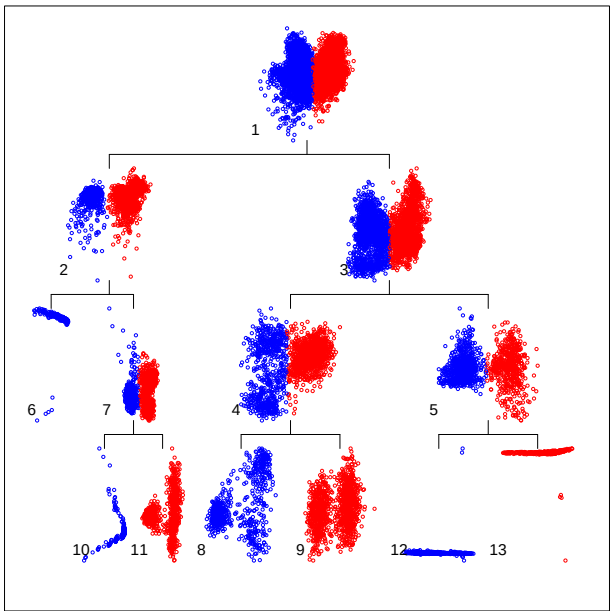
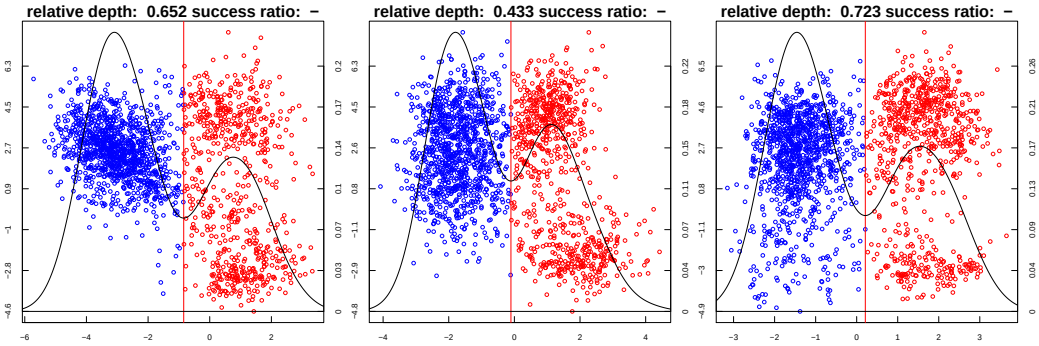


Figure 5: Initial clustering of optidigits data set through MDDC with 7 clusters.



(a) First principal component (b) Second principal component (c) Third principal component

Figure 6: Three potential binary partitions for node 4 based on three different initialisations.

be optimally separating the clusters. The default initialisation which gave rise to this projection was the first principal component of the data assigned to the node. We isolate the data assigned to this node to see if a better solution can be obtained by considering additional initialisations (we now consider the first three principal components). Each node in the model is stored as a list containing the field `$ixs` *which specifies the indices of the observations allocated to the node. We use the optional argument* `v0` *to investigate additional initialisations. When multiple initialisations are considered within one of the projection pursuit algorithms, all the resulting solutions are stored for inspection. Within a hierarchical clustering algorithm only the best solution is included in the model.*

```
> node4_x <- optidigits$x[sol$Nodes[[4]]$ixs,]
> v0 <- function(X) eigen(cov(X))$vectors[,1:3]
> node4_alt <- mdh(node4_x, v0 = v0)
> hp_plot(node4_alt[[1]], node4_x)
> hp_plot(node4_alt[[2]], node4_x)
> hp_plot(node4_alt[[3]], node4_x)
```

The outputs can be seen in Figure 6. The solution obtained by initialising on the third principal component (Figure F.6(c)) appears superior to the current solution (Figure F.6(a)). The clustering solution is thus first pruned at node 4, and then extended with a new initialisation. The function `tree_prune()` *removes the sub-hierarchy rooted at a specific node. The* `tree_split()` *function can then be used to extend a clustering model by splitting a leaf node further. The arguments accepted by both functions are shown in Table 5. Both functions return an object of the same type as the original solution. The function* `tree_split()` *accepts all optional arguments associated with the clustering algorithm which produced the solution* `sol`*. This allows the user to refine the solution at the node by modifying the way in which the projection pursuit is conducted. In*

Argument	Description
`sol`	A clustering solution arising from one of `mddc`, `mcdc` or `ncutdc`.
`node`	In `tree_prune` the node at which to prune the clustering model. In `tree_split` the node at which to extend the model by further splitting the data at the node.
`...`	(optional. `tree_split` only) Any collection of optional arguments associated with the clustering algorithm which produced the solution `sol`.

Table 5: Arguments accepted by `tree_prune` and `tree_split` functions.

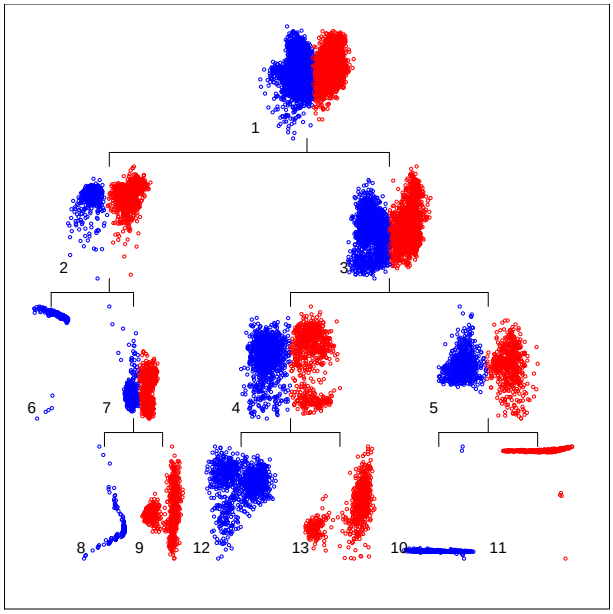


Figure 7: Clustering solution after the modification of the partition at node 4.

the example below we set the initial projection vector to the third principal component by using the optional argument `v0`*. It is important to note that the numbers assigned to the nodes in a model reflect the order in which they were added to the model. As such, pruning a model may alter the numbers of multiple nodes, including potentially those on different branches in the hierarchy. The modified clustering solution is shown in Figure 7.*

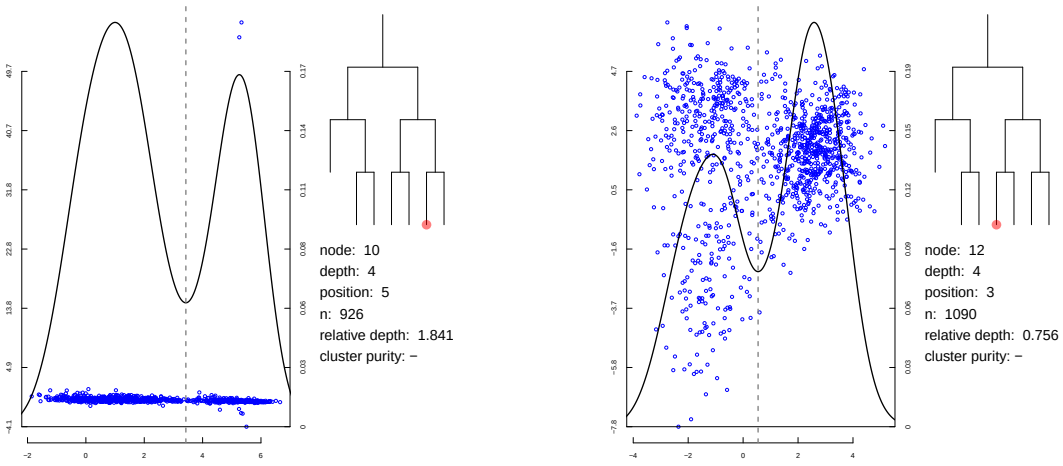
```
> sol <- tree_prune(sol, 4)
> v0 <- function(X) matrix(eigen(cov(X))$vectors[,3], ncol = 1)
> sol <- tree_split(sol, 4, v0 = v0)
> tree_plot(sol)
```

At this stage the internal nodes appear to produce satisfactory partitions of their data. However, there is evidence of multiple leaf nodes which each contain more than one cluster. A single node can be closely inspected using the function `node_plot(sol,node)`*. This produces an output similar to the* `hp_plot` *function, but includes information about the position of the node within the hierarchical clustering model as well. All leaf nodes can therefore be inspected separately. Examples including nodes numbered 10 and 12 are seen in Figure 8.*

```
> node_plot(sol, 10)
> node_plot(sol, 12)
```

Both nodes show strong evidence of multiple clusters, through their highly bimodal projected densities. Moreover, the potential partition at the leaf node numbered 12 does not appear to offer an optimal partition of the clusters. Alternative initialisations can be considered as above for node 4, leading to the conclusion that initialisation on the second principal component produces a better partition. The importance of closer inspection of individual nodes is also apparent in the case of node 10, where the high variance projection orthogonal to $\mathbf{v}_{\text{opt}}$*, depicted in the vertical axis in Figure F.8(a), is influenced by a few outlying points, making the two dimensional plot less relevant for visualising the cluster structure in the data. After inspecting all leaves, the nodes numbered 9, 10, 12 and 13 are further partitioned, with node 12 using the alternative initialisation discussed above.*

```
> sol <- tree_split(sol, 9)
> sol <- tree_split(sol, 10)
```



(a) Visualisation of node 10 (b) Visualisation of node 12

Figure 8: Visualisations of individual nodes in cluster hierarchy.

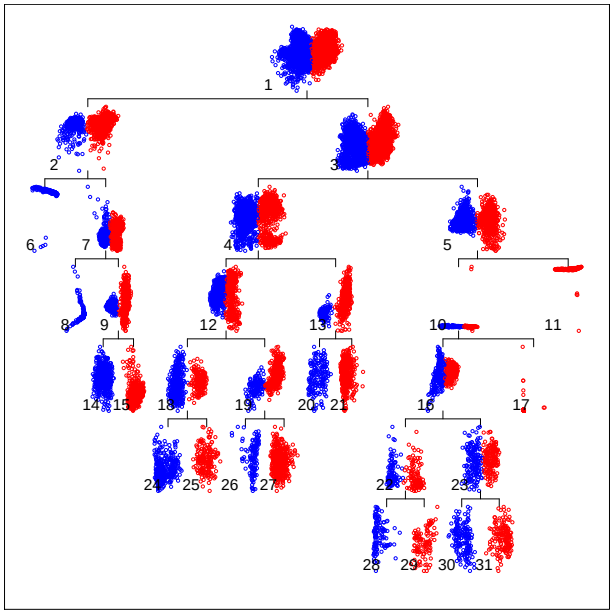


Figure 9: Final clustering solution after modifications. There is no evidence of individual clusters being split, and no evidence of clusters not identified in the model.

```
> v0 <- function(X) matrix(eigen(cov(X))$vectors[,2], ncol = 1)
> sol <- tree_split(sol, 12, v0 = v0)
> sol <- tree_split(sol, 13)
```

Inspecting the newly produced leaf nodes shows that leaves numbered 16, 18 and 19 likely contain multiple clusters, and so are further partitioned.

```
> sol <- tree_split(sol, 16)
> sol <- tree_split(sol, 18)
> sol <- tree_split(sol, 19)
```

Once again the resulting leaf nodes are inspected, and nodes 22 and 23 are partitioned.

```
> sol <- tree_split(sol, 22)
> sol <- tree_split(sol, 23)
```

The complete solution at this stage can be seen in Figure 9. The solution is valid in the sense that there appear to be no internal nodes at which a single cluster is split, and there are no leaf nodes which show evidence of multiple clusters. Finally, we compare the performance of this model with the solution assuming the number of clusters is known.

```
> sol2 <- mddc(optidigits$x, 10)
> cluster_performance(sol$cluster, optidigits$c)
 adj.rand    purity v.measure       nmi
0.7150910 0.8989324 0.7641322 0.7656866
> cluster_performance(sol2$cluster, optidigits$c)
 adj.rand    purity v.measure       nmi
0.6451702 0.7919929 0.7202152 0.7202187
```

The solution obtained by means of modifying the initial model overestimates the number of clusters by 60%. Despite this it substantially improves the overall agreement between the cluster assignment and the true class labels when compared with the solution assuming the number of clusters is known.

Extensions

In this section we consider two extensions of the projection pursuit methods discussed in the previous sections. First we discuss how to estimate large margin hyperplanes for clustering through either `mdh()` or `ncuth()`, and subsequently consider the binary partition of data sets whose clusters cannot be linearly separated.

Maximum margin hyperplanes for clustering

Maximum Margin Clustering (MMC) (Xu et al., 2004), seeks the maximum margin hyperplane to perform a bi-partition of unlabelled data. MMC can be equivalently viewed as identifying the binary labelling of $\mathcal{X}$ that will maximise the margin of a Support Vector Machine (SVM) classifier estimated on the labelled data set. Unlike the supervised SVM problem, which corresponds to a convex optimisation problem that can be efficiently solved, estimating MMC involves an integer optimisation problem. Exact methods for this problem are applicable only to very small data sets. Existing MMC algorithms that can handle reasonably sized data sets are not guaranteed to identify the globally optimal solution (Zhang et al., 2009).

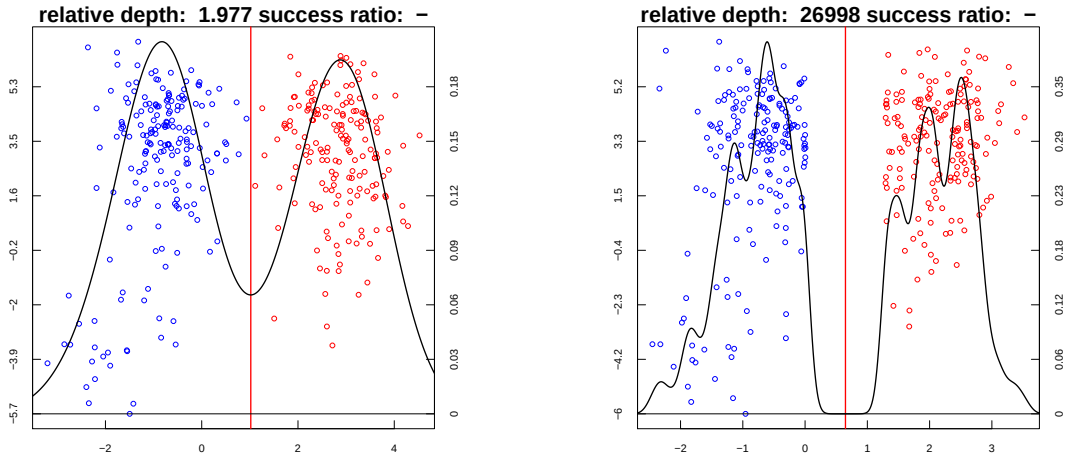
The minimum density hyperplane and the minimum normalised cut hyperplane converge to the maximum hard margin hyperplane through the data, as the bandwidth (scaling) parameter is reduced towards zero (Pavlidis et al., 2016; Hofmeyr, 2017). A simple and effective approach to obtain large margin hyperplanes for clustering is to apply either of the above methods for a decreasing sequence of bandwidth/scaling parameters.

Example 7. *In this example we attempt to separate digits 3 and 9 from the optidigits data set. This is one of the most difficult binary classification benchmarks considered in Zhang et al. (2009). We only use the test set of the optidigits data set to render our results directly comparable to those reported by Zhang et al. (2009). To obtain large margin hyperplane separators we apply the* `mdh()` *function recursively. At each iteration after the first, the initial projection vector is set to the optimal vector identified in the previous iteration, while the bandwidth parameter is multiplied by 0.9. The bandwidth can be specified with the optional argument* `bandwidth`*. So as not to affect the initialisation through the penalty term in the MDH formulation, the initial value of α is set to the final value from the previous solution (using the optional argument* `alphamin`*). This and the previous bandwidth value may be extracted from the* `$params` *field of the solution. The outputs of all of the binary-partitioning hyperplane algorithms in* **PPCI** *contain the field* `$params`*, a named list storing the parameters used in the projection pursuit.*

```
> ids <- (3824:5620)[which(optidigits$c[3824:5620]%in%c(3, 9))]
>
> x39 <- optidigits$x[ids,]
> hp0 <- mdh(x39)
>
> hp <- hp0
>
> repeat{
>   hp_new <- mdh(x39, v0 = hp[[1]]$v, bandwidth = hp[[1]]$params$h*0.9,
        alphamin = hp[[1]]$params$alpha)
>   if(hp[[1]]$v%*%hp_new[[1]]$v>(1-1e-10)) break
>   else hp <- hp_new
> }
>
> hp_plot(hp0[[1]], x39)
> hp_plot(hp[[1]], x39)
```

```
>
> 1-cluster_performance(hp[[1]]$cluster, optidigits$c[ids])[2]
      purity
  0.02479339
```

The first and last MDHs obtained from the above example are illustrated in Figure 10. As the figure shows, the optimal MDH for the smallest bandwidth achieves a large margin, unlike the MDH for the default setting of h. Notice that in Figure F.10(b) the very small value of the bandwidth causes the density on the separating hyperplane to be very close to zero, and consequently the relative depth is extremely large.



(a) MDH for default bandwidth (b) MDH for smallest bandwidth considered

Figure 10: Large margin hyperplane separator for the optdigits 3-9 problem obtained by applying `mdh()` for a decreasing sequence of bandwidths

For this data set (Zhang et al., 2009, Table IV) report the clustering error of k-means, kernel k-means, normalised cut spectral clustering (Shi and Malik, 2000), generalised maximum margin clustering (Valizadegan and Jin, 2006), and three versions of the iterative support vector machine (regression) they propose. The best performance is achieved by generalised maximum margin clustering with an error of 0.083. Both versions of k-means, and spectral clustering achieve an error around 0.2. The more recent cutting plane maximum margin clustering algorithm (Wang et al., 2010) achieves a much larger error of 0.3609 on this data set.[1] *The large margin hyperplane obtained through repeatedly reducing the bandwidth parameter of the MDH algorithm achieves an error that is more than three times smaller than that of the best competing algorithm.*

Non-linear separators using Kernel PCA

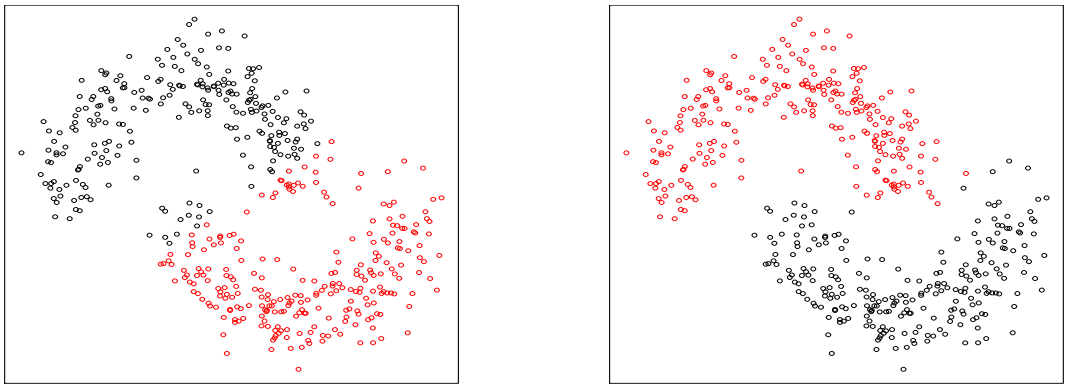
The main limitation of linear cluster separators is their inability to accurately separate clusters whose convex hulls overlap. Although many real world applications involve data in which clusters can be separated well by hyperplanes, this is not always the case, especially in lower dimensional examples. Using Kernel Principal Components Analysis (KPCA) (Schölkopf et al., 1998) the data can be embedded in a high-dimensional feature space in which clusters are linearly separable. Hyperplanes in the feature space correspond to nonlinear cluster boundaries in the original space. Below we illustrate how combining KPCA with a linear cluster separator enables the separation of clusters which cannot be linearly separated within the input space. We then apply this approach to the application of identifying different hand-written digits based on pen trajectories. We selected the hyperparameter for the kernel embedding using trial and error; choosing a value for which the cluster sizes were reasonably well balanced. The setting of kernel hyperparameters remains a challenging problem in the unsupervised setting.

Example 8. *We employ a data set containing two clusters, each in the shape of a half moon, arranged so that they cannot be separated by a hyperplane (Jain and Law, 2005). We use KPCA to embed the data in a high-dimensional feature space in which the clusters are linearly separable.*

```
> require(kernlab)
> set.seed(1)
> th <- runif(500)*pi + rep(c(pi,0),each=250)
```

[1] Code for this method is available at https://sites.google.com/site/binzhao02/

```
>
> x <- cbind(cos(th) + rep(c(0.5,-0.5),each=250), sin(th))
> x <- x + matrix(0.18*rnorm(1000), ncol=2)
> x2 <- kernlab::kpca(x, kernel = "rbfdot", kpar = list(sigma = 3))@rotated
> sol1 <- ncuth(x)
> sol2 <- ncuth(x2)
> par(mfrow = c(1, 2))
> par(mar = c(2.5, 2.5, 3, 2.5))
> plot(x, col = sol1[[1]]$cluster, main = "Linear Separator")
> plot(x, col = sol2[[1]]$cluster, main = "Non-linear Separator")
```



(a) Linear Separator (b) Nonlinear Separator

Figure 11: Nonlinear cluster separation after preprocessing the data through Kernel PCA

Example 9. *We next apply the KPCA embedding to the Pen Based Recogntion of Handwritten Digits data set (Lichman, 2013). Due to the size of the data set, which includes 10992 observations, we build the kernel embedding using a compressed version of the data set obtained through k-means.*

```
> require(kernlab)
> set.seed(1)
> data(pendigits)
> kp <- kernlab::kpca(kmeans(pendigits$x, 200)$centers, kernel = "rbfdot",
        kpar = list(sigma = 0.1))
> x2 <- predict(kp, pendigits$x)
> sol1 <- ncutdc(pendigits$x, 10)
> sol2 <- ncutdc(x2, 10)
> cluster_performance(sol1$cluster, pendigits$c)
   adj.rand    purity v.measure       nmi
  0.6180774 0.7801128 0.7097384 0.7097736
> cluster_performance(sol2$cluster, pendigits$c)
   adj.rand    purity v.measure       nmi
  0.7130409 0.8265102 0.7940943 0.7941021
```

The performance is substantially improved by first embedding the data in the feature space and then applying NCUTDC.

Conclusions

This paper discusses three recently proposed projection pursuit methods for clustering, and the implementation in the **R** package **PPCI**. The projection pursuit methods seek univariate projections that maximise the clusterability of the binary partition of the projected data, or alternatively minimise the connectedness between the elements of this bi-partition. Identifying projection directions that explicitly optimise a clustering criterion enables significant improvements over PCA and other classical dimension reduction techniques. The cluster boundaries induced by these methods are hyperplanes, which can be combined to produce divisive hierarchical clustering models capable of identifying clusters defined in different subspaces and with different scales. An important advantage of using hyperplanes for clustering is that projecting onto the vector normal to the hyperplane provides the maximum dimensionality reduction. Visualising the data through such projections allows the user to

obtain insights concerning the validity of the clustering model. We discuss how kernel PCA can be combined with the implemented methods to handle clusters which cannot be linearly separated. Two of the implemented projection pursuit algorithms are asymptotically equivalent to the maximum hard margin separator as their smoothing parameters are reduced to zero. We illustrate how large margin hyperplanes for clustering can be obtained using these methods.

Bibliography

M. Ackerman and S. Ben-David. Clusterability: A theoretical study. In D. van Dyk and M. Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 1–8, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL http://proceedings.mlr.press/v5/ackerman09a.html. [p158]

R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International Conference of Management of Data*, pages 94–105. ACM, 1998. [p152]

S. Ben-David, T. Lu, D. Pal, and M. Sotakova. Learning low density separators. In D. van Dyk and M. Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 25–32, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL http://proceedings.mlr.press/v5/ben-david09a.html. [p156]

D. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, Dec 1998. ISSN 1573-756X. doi: 10.1023/A:1009740529316. URL https://doi.org/10.1023/A:1009740529316. [p153]

J. V. Burke, A. S. Lewis, and M. L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM Journal on Optimization*, 15(3):751–779, 2005. doi: 10.1137/030601296. [p154]

O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In R. G. Cowell and Z. Ghahramani, editors, *aistats05*, pages 57–64. Society for Artificial Intelligence and Statistics, 2005. URL http://www.gatsby.ucl.ac.uk/aistats/fullpapers/198.pdf. [p156]

O. Chapelle, V. Sindhwani, and S. S. Keerthi. Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9:203–233, Feb 2008. URL http://www.jmlr.org/papers/v9/chapelle08a.html. [p156]

D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002. ISSN 0162-8828. doi: 10.1109/34.1000236. [p156]

A. Cuevas, M. Febrero, and R. Fraiman. Cluster analysis: a further approach based on density estimation. *Computational Statistics and Data Analysis*, 36(4):441–459, 2001. doi: 10.1016/S0167-9473(00)00052-9. [p156]

A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):643–660, Jun 2001. doi: 10.1109/34.927464. [p159]

J. A. Hartigan. *Clustering Algorithms*. Wiley Series in Probability and Mathematical Statistics. Wiley, New York, 1975. [p156]

M. Hassani and M. Hansen. *subspace: Interface to OpenSubspace*, 2015. R package version 1.0.4. [p152]

D. Hofmeyr and N. Pavlidis. Maximum clusterability divisive clustering. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 780–786, Dec 2015. doi: 10.1109/SSCI.2015.116. [p154, 155]

D. P. Hofmeyr. Clustering by minimum cut hyperplanes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(8):1547–1560, Aug 2017. ISSN 0162-8828. doi: 10.1109/TPAMI.2016.2609929. [p155, 160, 161, 165]

P. J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 06 1985. doi: 10.1214/aos/1176349519. [p152]

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, Dec 1985. doi: 10.1007/BF01908075. [p158]

A. Hyvärinen, J. Karhunen, and E. Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004. [p152]

A. K. Jain and M. H. C. Law. Data clustering: A user's dilemma. In S. K. Pal, S. Bandyopadhyay, and S. Biswas, editors, *Pattern Recognition and Machine Intelligence: First International Conference, PReMI 2005, Kolkata, India, December 20-22, 2005. Proceedings*, pages 1–10, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi: 10.1007/11590316_1. [p166]

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3): 264–323, 1999. doi: 10.1145/331499.331504. [p152]

K. Kailing, H.-P. Kriegel, and P. Kröger. Density-connected subspace clustering for high-dimensional data. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 246–256. SIAM, 2004. [p152]

A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab - an S4 package for kernel methods in R. *Journal of Statistical Software, Articles*, 11(9):1–20, 2004. doi: 10.18637/jss.v011.i09. [p161]

A. S. Lewis and M. L. Overton. Nonsmooth optimization via quasi-newton methods. *Mathematical Programming*, 141(1):135–163, Oct 2013. doi: 10.1007/s10107-012-0514-2. [p154]

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml. [p157, 161, 167]

L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov): 2579–2605, 2008. [p152]

G. Menardi and A. Azzalini. An advancement in clustering via nonparametric density estimation. *Statistics and Computing*, 24(5):753–767, 2014. doi: 10.1007/s11222-013-9400-x. [p156]

N. G. Pavlidis, D. P. Hofmeyr, and S. K. Tasoulis. Minimum density hyperplanes. *Journal of Machine Learning Research*, 17(156):1–33, 2016. URL http://jmlr.org/papers/v17/15-307.html. [p155, 156, 157, 158, 165]

A. Rinaldo and L. Wasserman. Generalized density clustering. *The Annals of Statistics*, 38(5):2678–2722, 2010. doi: 10.1214/10-AOS797. [p156]

A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, volume 7, pages 410–420, 2007. URL http://www1.cs.columbia.edu/~amaxwell/pubs/v_measure-emnlp07.pdf. [p158]

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998. doi: 10.1162/089976698300017467. [p166]

J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, Aug 2000. doi: 10.1109/34.868688. [p160, 166]

A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, Dec 2002. URL http://www.jmlr.org/papers/v3/strehl02a.html. [p158]

W. Stuetzle and R. Nugent. A generalized single linkage method for estimating the cluster tree of a density. *Journal of Computational and Graphical Statistics*, 19(2):397–418, 2010. doi: 10.1198/jcgs.2009.07049. [p156]

S. K. Tasoulis, D. K. Tasoulis, and V. P. Plagianakos. Enhancing principal direction divisive clustering. *Pattern Recognition*, 43(10):3391–3411, 2010. doi: 10.1016/j.patcog.2010.05.025. [p153]

S. K. Tasoulis, D. K. Tasoulis, and V. P. Plagianakos. Random direction divisive clustering. *Pattern Recognition Letters*, 2012. doi: http://dx.doi.org/10.1016/j.patrec.2012.09.008. [p153]

M. Thrun, F. Lerch, and F. Pape. *ProjectionBasedClustering: Projection Based Clustering*, 2018. URL https://CRAN.R-project.org/package=ProjectionBasedClustering. R package version 1.0.6. [p152]

H. Valizadegan and R. Jin. Generalized maximum margin clustering and unsupervised kernel learning. In P. B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1417–1424. MIT Press, 2006. URL http://papers.nips.cc/paper/3072-generalized-maximum-margin-clustering-and-unsupervised-kernel-learning.pdf. [p166]

U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. ISSN 0960-3174. doi: 10.1007/s11222-007-9033-z. [p160]

F. Wang, B. Zhao, and C. Zhang. Linear time maximum margin clustering. *IEEE Transactions on Neural Networks*, 21(2):319–332, 2010. doi: 10.1109/TNN.2009.2036998. [p166]

L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1537–1544. MIT Press, 2004. URL http://papers.nips.cc/paper/2602-maximum-margin-clustering.pdf. [p156, 165]

B. Zhang. Dependence of clustering algorithm performance on clustered-ness of data. *HP Labs Technical Report HPL-2001-91*, 2001. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.285&rep=rep1&type=pdf. [p158]

K. Zhang, I. W. Tsang, and J. T. Kwok. Maximum margin clustering made practical. *IEEE Transactions on Neural Networks*, 20(4):583–596, 2009. doi: 10.1109/TNN.2008.2010620. [p153, 165, 166]

Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004. doi: 10.1023/B:MACH.0000027785.44527.d6. [p158]

David P. Hofmeyr
Stellenbosch University
Stellenbosch
South Africa
dhofmeyr@sun.ac.za

Nicos G. Pavlidis
Lancaster University
Lancaster
United Kingdom
n.pavlidis@lancaster.ac.uk

dr4pl: A Stable Convergence Algorithm for the 4 Parameter Logistic Model

by Hyowon An, Justin T. Landis, Aubrey G. Bailey, James S. Marron and Dirk P. Dittmer

Abstract The *4 Parameter Logistic (4PL) model* has been recognized as a major tool to analyze the relationship between doses and responses in pharmacological experiments. A main strength of this model is that each parameter contributes an intuitive meaning enhancing interpretability of a fitted model. However, implementing the 4PL model using conventional statistical software often encounters numerical errors. This paper highlights the issue of convergence failure and presents several causes with solutions. These causes include outliers and a non-logistic data shape, so useful remedies such as robust estimation, outlier diagnostics and constrained optimization are proposed. These features are implemented in a new R package dr4pl (Dose-Response analysis using the 4 Parameter Logistic model) whose code examples are presented as a separate section. Our R package dr4pl is shown to work well for data sets where the traditional dose-response modelling packages drc and nplr fail.

Introduction

Dose-response models are a primary tool of pharmacological and toxicological studies. After fitting a suitably chosen model to experimental data, the fitted parameters are used to evaluate the potency of a drug or compare the potencies of different drugs. Based on outcomes of the analysis, drug candidates may be advanced to further development. One of the most popular models among dose-response models is the *4 Parameter Logistic (4PL) model* whose detailed explanation is given in Subsection 4 Parameter Logistic model. In this paper, we focus on some important challenges in the 4PL model and present methods to handle those challenges.

Fitting dose-response models to data is usually a nonlinear regression problem. One of the problems with nonlinear regression is that numerical errors often occur during a model fitting optimization process. The 4PL model is not an exception. Examples of problem data sets and diagnostics of their problems are presented in Subsection Motivation: convergence failure.

In this paper, we analyze the convergence failure problem within the context of the 4PL model. Descriptions of the 4PL model and existing methods are presented in Section Existing methods. Problems are clearly illustrated using the contour of the likelihood function, shown in Section Diagnosis of convergence failure. This suggests a useful set of constraints to put on the parameter space, and their effectiveness is analyzed in that section as well. Non-convergence is indicated by our R package **dr4pl** (Landis et al., 2018) when crossing one of these boundaries, in which case robust parameter estimates are returned as described in Section Remedy for outliers. Section Remedy for the support problem presents a new method of selecting initial parameter estimates which is not currently implemented in other dose-response modelling R packages. This method will be shown to be effective in handling a non-logistic data shape in that section. Walk-through of R functions in **dr4pl** with real-world data sets will be presented in Section Walk-through in R.

Existing methods

This section describes the 4PL model and existing methods that have been implemented for the 4PL model. Example data sets that motivate our research are presented as well.

4 Parameter Logistic model

The response in a pharmacological process usually slowly trends downward at low dose levels, then rapidly drops at a certain dose level and finally stabilizes. Tracking this sort of change in response results in a reverse S shaped curve motivating sigmoidal curves as primary models of interest. Robertson (1908) suggested that the biochemical nature of such processes is well modelled by the differential equation

$$\frac{\mathrm{d}g}{\mathrm{d}z} = \frac{\theta_3}{\theta_1} g(z)\left(\theta_1 - g(z)\right), \tag{1}$$

where z is a dose level, $g(z)$ is the response at the level z and $\theta_1, \theta_3 \in \mathbb{R}$ are parameters. Solving this differential equation yields

$$g(z|\theta_1, \theta_2, \theta_3) = \theta_1 - \frac{\theta_1}{1 + (z/\theta_2)^{\theta_3}},$$

where $\theta_2 > 0$ is a parameter coming from a boundary condition given to Equation (1). Since this model allows the response value $g(z)$ to range only between 0 and θ_1, a common extension uses a shift parameter $\theta_4 \in \mathbb{R}$ yielding the well-known *4 parameter logistic* (*4PL*) *model* or *Hill model* given in Hill (1910) as

$$\begin{aligned} f(x|\theta) &= \theta_1 + \frac{\theta_4 - \theta_1}{1 + 10^{\theta_3(x - log_{10}\theta_2)}} \\ &= \theta_1 + \frac{\theta_4 - \theta_1}{1 + (z(x)/\theta_2)^{\theta_3}}, \end{aligned} \tag{2}$$

where $x \in \mathbb{R}$ is a log 10 dose, $z(x) = 10^x > 0$ is the corresponding dose and the parameter vector $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$ determines the relationship between the log 10 dose x and the response $f(x)$. Working on this log 10 scale has been shown to accurately model many physical reactions.

A major strength of the 4PL model is that each of the four parameters has an intuitive meaning. The parameters θ_1 and θ_4 are called the *upper asymptote* and *lower asymptote*, respectively, since we have from Equation (2) that

$$\begin{aligned} &\text{if } \theta_3 > 0, \text{ then } \lim_{x \to -\infty} f(x|\theta) = \theta_4, \ \lim_{x \to \infty} f(x|\theta) = \theta_1, \\ &\text{if } \theta_3 < 0, \text{ then } \lim_{x \to -\infty} f(x|\theta) = \theta_1, \ \lim_{x \to \infty} f(x|\theta) = \theta_4. \end{aligned}$$

In this paper, we assume that $\theta_1 > \theta_4$ holds. This implies that $\theta_3 > 0$ corresponds to an increasing curve while $\theta_3 < 0$ corresponds to a decreasing curve. A response usually decreases as a dose increases in a pharmacological experiment, so we focus on decreasing curves throughout this paper. However, all the functionalities such as model fitting and confidence interval calculation are implemented in the R package **dr4pl** for both decreasing and increasing curves. The parameter θ_2 is called the *half maximum inhibitory concentration* (*IC50*) or *effective concentration* (*EC50*), depending on whether the curve is decreasing or increasing, which represents the dose level whose response lies halfway between the upper and lower asymptotes. That is, we have $f(\theta_2) = (\theta_1 + \theta_4)/2$. The parameter θ_3 is related to the slope of the function f at θ_2 as given in Appendix A of Giraldo et al. (2002) by

$$\left.\frac{\mathrm{d}}{\mathrm{d}x} f(x|\theta)\right|_{x = log_{10}\theta_2} = (\ln 10)\,(\theta_4 - \theta_1)\,\theta_3,$$

where ln indicates the natural logarithm.

The effects of the IC50 parameter θ_2 and the slope parameter θ_3 on the 4PL model is illustrated in Figure 1. For each plot in the figure, the x-axis represents dose levels on a log 10 scale and the y-axis represents response values. For both the left and right plots, the upper asymptote θ_1 is fixed at 100 and the lower asymptote θ_4, which is represented by a horizontal gray dashed line, is fixed at 0. The left plot shows the change in the dose response curve as the IC50 parameter θ_2 changes. Note that the blue dotted curve with the largest θ_2 value of 10 is located on the right while the red solid curve with the smallest θ_2 value of 0.1 is located on the left. However, the rate of decrease θ_3 of each curve is the same. The right plot shows that the blue dotted curve with the most negative θ_3 value of -2 most rapidly decreases around the IC50 value while the red solid curve with the least negative θ_3 value of -0.5 most slowly decreases. Note that all three curves in the right plot achieve the half response θ_2 at the same dose level, i.e. they have the same IC50 values.

Motivation: convergence failure

As mentioned in Section Introduction, dose-response models are prone to numerical errors in their optimization processes like other nonlinear regression models; see Bates and Watts (1988) and Seber and Wild (1989) for theoretical and practical issues related to nonlinear regression models. The specific type of errors in dose-response models depends on the data and statistical software at hand, and here we focus on R packages developed for bioassay data. The R package **drc** (Ritz et al., 2005) provided various non-linear regression models for bioassay data analysis. The R package **nplr** (Commo and Bot, 2016) focused on the family of n-parameter logistic regression models with an option to choose an optimal value n based on a goodness-of-fit measure. Unfortunately, these two R packages have only scant solutions to the convergence failure problem. For example, **drc** provides a function argument `errorm` to control whether an error or a warning message is drawn when convergence failure happens,

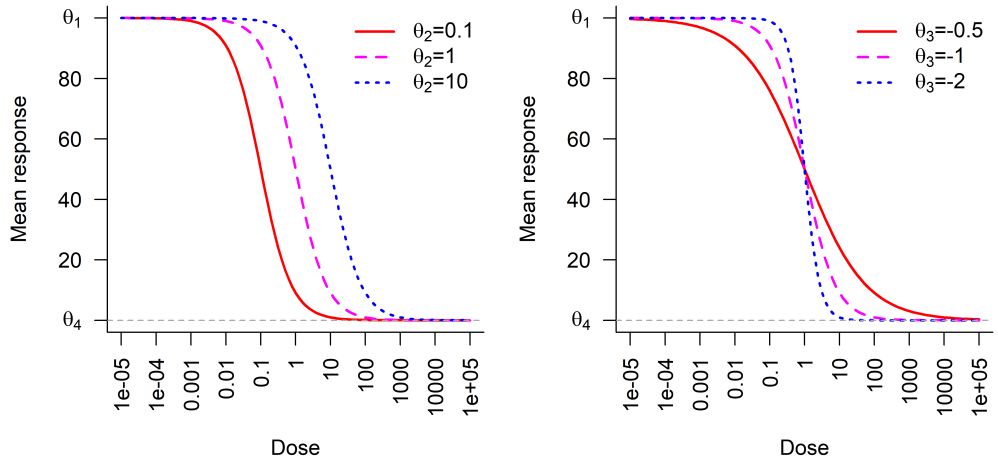


Figure 1: 4PL models with different values of the IC50 parameter θ_2 (left plot) and the slope parameter θ_3 (right plot). The other parameters are fixed at $\theta_1 = 100, \theta_4 = 0$ with $\theta_3 = -1$ in the left plot and $\theta_2 = 1$ in the right plot. The left plot implies that θ_2 controls horizontal shifts while the right plot implies that θ_3 drives rates of decrease.

and **nplr** provides a weighting scheme that reduces the effect of outliers. Both methods did not suffice to prevent convergence failure as shown in the following.

Figure 2 shows four real-world data sets. The R package **drc** fails to converge on all these four data sets and the R package **nplr** fails on the last data set. In each plot of the figure, the x-axis represents doses on a log 10 scale and the y-axis represents responses. Data points flagged as problematic by our R package are shown as red triangles. These include strong outliers in every case, and are found as described in Subsection Robust regression and outlier detection. Error Case 1 in the upper left plot has an outlier at the dose level 10^{-5} which seems to result from an error in measurement. Error Case 2 in the upper right plot has more complicated problems. There exists a strong outlier at the dose level 1.35, and the two responses measured at the dose level 0.135 seem to be measured at a wrong scale. Both of them have abnormally large values and their values are far more different from each other than the responses measured at other dose levels. Even though these two responses were not detected as outliers by our method, this plot indicates that the data set of Error Case 2 requires users' attention.

The data in the lower two plots have a different problem in addition to outliers. They seem to have insufficient data points on the right side of their IC50 parameters. This situation is well known in dose-response experiments in cases where solubility issues limit the maximal drug concentration that can be used in dose titration. Error Case 3 in the lower left plot shows two strong outliers with abnormally small response values at the dose level 10. In addition, the response starts decreasing at the dose levels 10 and 100 but no more responses are measured at higher dose levels. This is because the drug fell out of solution at higher concentrations in this experiment. We call this type of problem a *support problem* because of a lack of data points on the right side, and possibly also on the left side, of the IC50 parameter. This lack in Error Case 3 can result in instability of an optimization process due to difficulty in estimation of the IC50 parameter θ_2 and the slope parameter θ_3. Indeed, the package **drc** draws an error message for that data. Error Case 4 in the lower right plot has a similar support problem also resulting in a **drc** error.

Convergence failure of numerical optimization methods can be overcome in various ways. Experimenters often try other starting parameter values or adopt a simpler mathematical model. However, both solutions have a drawback in dose-response modelling since they require arbitrary decisions by experimenters. This limits their utility for automated, high-throughput screening and may be cost-prohibitive depending on the expense per data point. It can be foolhardy to decide which starting values should be tried next unless problems with data are diagnosed. Performing a grid search on the parameter space can be an alternative as claimed in Wang et al. (2010), but even this does not fundamentally solve the problems with the data sets in Figure 2. Shifting to a simpler model can result in loss of interpretability of the 4PL model. As pointed out in Di Veroli Giovanni et al. (2015) and Lim (2015), the 4PL model conveys useful intuitive content through their parameters.

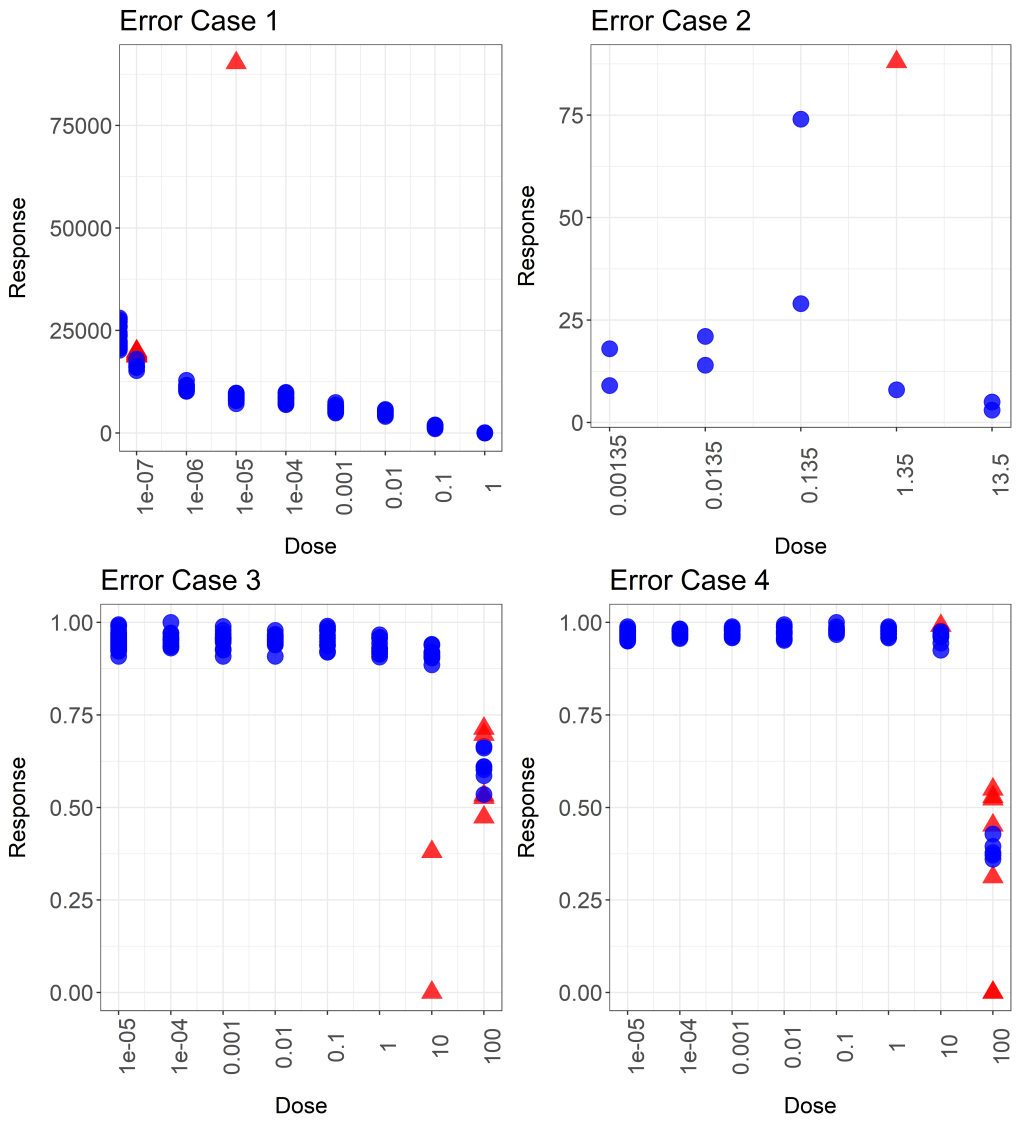


Figure 2: Data that generate numerical errors in the R packages **drc** and **nplr**. The *y*-axis represents responses and the *x*-axis represents doses in a *log* 10 scale. All data sets have at least one strong outlier which are denoted by red triangles. Data in the lower row also have insufficient numbers of data points on the right sides of the dose-response curves.

Diagnosis of convergence failure

The 4PL model given in Equation (2) is fitted to a data set $\{(x_i, y_i) \,|i = 1, 2, \cdots, n\}$ where x_i is the log 10 dose level of the i-th observation and y_i is the corresponding response value. The most commonly chosen loss function is the sum-of-squares errors essentially given as

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f\left(x_i|\theta\right)\right)^2, \tag{3}$$

where the function f was given in Equation (2) as

$$f(x|\theta) = \theta_1 + \frac{\theta_4 - \theta_1}{1 + 10^{\theta_3(x - log_{10}\theta_2)}}. \tag{4}$$

Since this function is nonlinear in the parameters θ, minimization of $\mathcal{L}$ is a typical nonlinear regression problem. Gadagkar and Call (2015) mentioned the optimization problem is indeed a non-convex optimization, so different initial parameter estimates can result in different fitted models or even convergence failure. Example data sets with such a convergence failure problem were given in Figure 2. In this section, we present one possible solution to the convergence problem based on statistical theory, which can help users diagnose problems with their data.

Shape of a loss function

To find a reason behind the convergence failure, we draw contour plots of the sum-of-squares loss function given in Equation (3). To visualize the loss function $\mathcal{L}\left(\theta|x\right)$, we first fix the values of θ_1 and θ_4 at $y_{\max}$ and $y_{\min}$ defined as

$$\begin{aligned} y_{\max} &= \max_{1\le i\le n} y_i + 0.001 \cdot \left(\max_{1\le i\le n} y_i - \min_{1\le i\le n} y_i\right), \\ y_{\min} &= \min_{1\le i\le n} y_i - 0.001 \cdot \left(\max_{1\le i\le n} y_i - \min_{1\le i\le n} y_i\right), \end{aligned} \tag{5}$$

respectively. The reason for choosing the above estimates is that the R package **drc** uses them as a default. Then we draw contour plots of the loss functions in terms of θ_2 and θ_3 yielding Figure 3 in which the contour plots of the data of Figure 2 are shown. In each plot, the x-axis represents the IC50 parameter θ_2 in a log 10 scale and the y-axis represents the slope parameter θ_3. The parameter values corresponding to the same loss function value are connected as a solid line and the error value is given as text inserted into that line. It can be seen from the upper two plots that the loss function monotonically decreases as the IC50 parameter θ_2 decreases towards zero. This results in numerical errors in computing the estimated mean response in Equation (4), since as $\theta_2 \to 0$, the $log_{10}(\theta_2) \to -\infty$. The lower two plots also indicate that the loss function tends to decrease as the values of the IC50 parameter θ_2 increase towards infinity. Similarly to the upper two plots, this causes numerical errors in finding the minimizer of the loss function.

Hill bounds: safeguards on the parameter space

The primary resolution of the convergence failure problem comes from imposing appropriate boundaries on the parameter space to prevent parameter estimates from diverging to infinity or converging to 0. These bounds should be broad enough to contain the global minimum of the loss function when it has any local minimum. To obtain reasonable sizes of bounds on the parameters, we focus on the *Hill plot*. Recall from Equation (2) that the mean response is modelled as

$$f(x|\theta) = \theta_1 + \frac{\theta_4 - \theta_1}{1 + 10^{\theta_3(x - log_{10}\theta_2)}},$$

where x is a log 10 dose. Note that the nonlinear nature of the 4PL model comes from the fact that the linear function of x, $\theta_3\left(x - log_{10}\theta_2\right)$, appears as an exponential in the denominator of the expression. This motivates us to rearrange the terms to yield the *Hill equation* modeled as

$$log_{10}\left(\frac{f(x|\theta) - \theta_4}{\theta_1 - f(x|\theta)}\right) = \theta_3 x - \theta_3 log_{10}(\theta_2)\,. \tag{6}$$

The left hand side of this equation is the logit-transformed mean response and the right hand side is the linear function of log 10 dose x. Since both the asymptotes θ_1 and θ_4 are unknown, we use $y_{\max}$

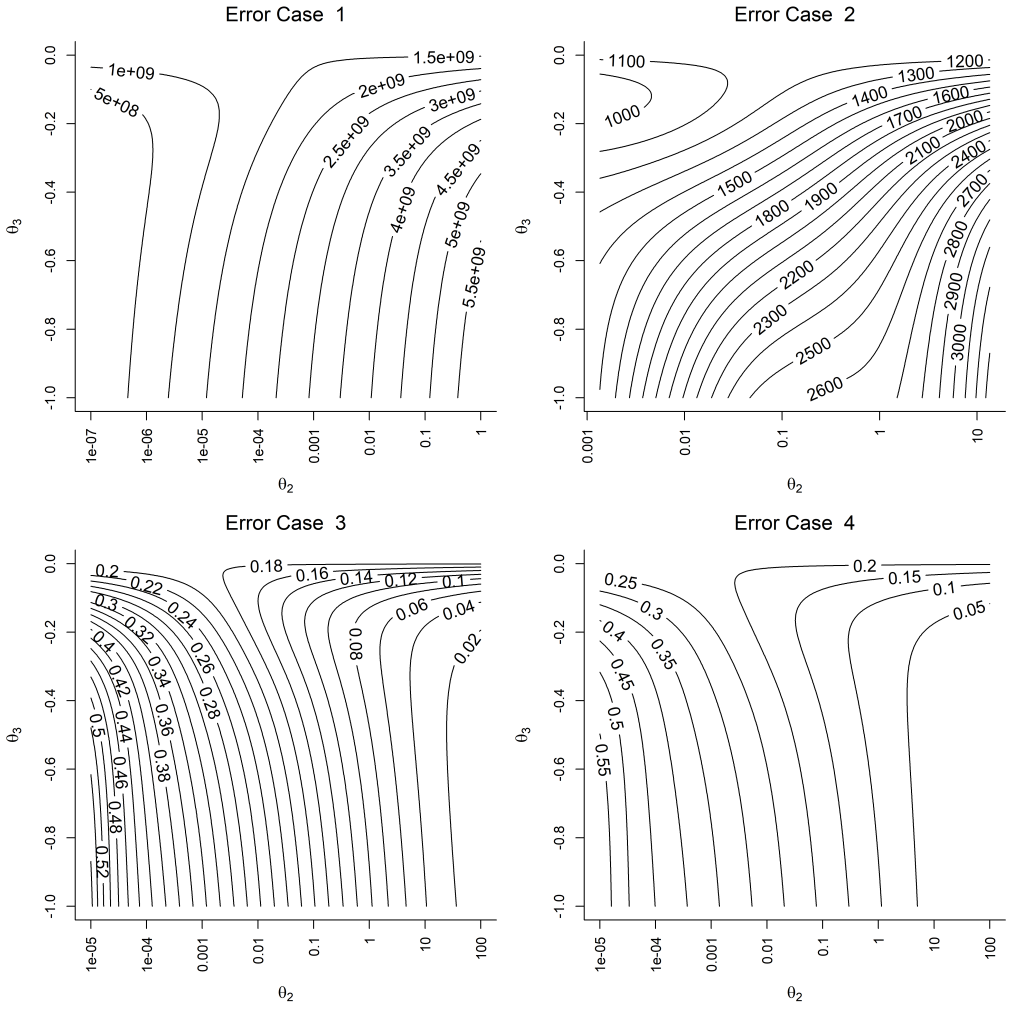


Figure 3: Contour plots of data shown in Figure 2. The x-axis represents the IC50 parameter θ_2 and the y-axis represents the slope parameter θ_3. The loss functions in the upper two plots decrease as the IC50 θ_2 decreases, and the loss functions in the lower two plots decrease as θ_2 increases; a minimum is not achieved in the plots.

and $y_{\min}$ defined in Equation (5) as their estimates in real data approximation as

$$log_{10}\left(\frac{y_i - y_{\min}}{y_{\max} - y_i}\right) \approx -\theta_3 log_{10}(\theta_2) + \theta_3 x_i. \tag{7}$$

See Chapter 2 of Pratt and Taylor (1990) for detailed discussion of the Hill plot and Hill equation.

To obtain reasonable bounds on the parameters θ_2 and θ_3, we first view Equation (7) as a linear regression problem. If we re-parameterize Equation (7) as

$$log_{10}\left(\frac{y_i - y_{\min}}{y_{\max} - y_i}\right) = \beta_2 + \beta_3 x_i + \epsilon_i, \tag{8}$$

where $\beta_2 = -\theta_3 log_{10}(\theta_2)$, $\beta_3 = \theta_3$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, we can obtain interval estimators of β_2 and β_3 based on $(1-\alpha)$ confidence intervals based on the normality assumption. That is, we compute

$$\begin{aligned} &\left(\tilde{\beta}_2 - z_{1-\alpha/2}\text{s.e.}\left(\tilde{\beta}_2\right), \tilde{\beta}_2 + z_{1-\alpha/2}\text{s.e.}\left(\tilde{\beta}_2\right)\right), \\ &\left(\tilde{\beta}_3 - z_{1-\alpha/2}\text{s.e.}\left(\tilde{\beta}_3\right), \tilde{\beta}_3 + z_{1-\alpha/2}\text{s.e.}\left(\tilde{\beta}_3\right)\right), \end{aligned} \tag{9}$$

where $\tilde{\beta}_2$ and $\tilde{\beta}_3$ are the least squares solutions of Equation (8), z_p is the p-th quantile of the standard normal distribution and s.e. is the abbreviation for the standard errors of the regression coefficients. If the linear model of Equation (7) well approximates the original 4PL model of Equation (2), then parameter estimates will rarely be out of the bounds in Equation (9) during the optimization process.

However, using these bounds in optimization have limitations in practice in that they may miss important nonlinear structure inherent in the 4PL model. In real data analysis, such confidence intervals imposed too narrow bounds on the parameters and hindered optimization processes from convergence even when data have good logistic shapes. Section 5.1 of Seber and Wild (1989) present approximate confidence intervals of the true parameters of nonlinear regression models based on the derivatives of the mean response function f. When data have sufficiently many data points, the least squares estimators $\hat{\theta}$ have asymptotically normal distributions

$$\hat{\theta} - \theta^* \xrightarrow{d} \mathcal{N}\left(0, C^{-1}\right), \tag{10}$$

where θ^* is the true parameter vector and $C = J^T J$ with J being the Jacobian matrix whose (i,j)-th element is given as $\partial f(x_i|\theta)/\partial\theta_j$. In actual computation of confidence intervals, the parameter estimators $\hat{\theta}$ are used to approximate J, i.e. we substitute $\hat{J}$ whose (i,j)-th element is $\partial f(x_i|\theta)/\partial\theta_j\big|_{\theta=\hat{\theta}}$ for J.

Based on this asymptotic theory, the approximate confidence intervals for the true parameters are given in Section 5.1 of Seber and Wild (1989) as

$$\left(\hat{\theta}_j + t_{n-p}^{\alpha/(2\cdot p)} s\sqrt{\hat{c}_{j,j}},\ \hat{\theta}_j + t_{n-p}^{1-\alpha/(2\cdot p)} s\sqrt{\hat{c}_{j,j}}\right), \tag{11}$$

for $1 \le j \le p$ where p is the number of parameters, s^2 is the residual sum of squares, t_n^α is the α-th quantile of the t-distribution with n degrees of freedom and $\hat{c}_{j,j}$ is the j-th diagonal element of the matrix $\hat{C}^{-1} = \left(\hat{J}^T\hat{J}\right)^{-1}$. In the context of the 4PL model, we evaluate the interval estimators based on parameter estimators $\tilde{\theta}_1, \tilde{\theta}_2, \tilde{\theta}_3, \tilde{\theta}_4$ obtained from the Hill equation in Equation (7) and standard errors obtained from Equation (11). That is, we have

$$\left(\tilde{\theta}_j + t_{n-4}^{\alpha/8} s\sqrt{\tilde{c}_{j,j}}, \tilde{\theta}_j + t_{n-4}^{1-\alpha/8} s\sqrt{\tilde{c}_{j,j}}\right), \tag{12}$$

for $1 \le j \le 4$ where $\tilde{c}_{jj}$ is the j-th diagonal element of the variance-covariance matrix $\tilde{C}^{-1}$ evaluated at the initial parameter estimators $\tilde{\theta}$ obtained from the Hill equation. Since the bounds in Equation (12) originate from the Hill model, we call these the *Hill bounds*. Subsection 5.2.2 of Seber and Wild (1989) suggests using $\hat{H}/2$ instead of $\hat{J}^T\hat{J}$ when computing confidence intervals where

$$\hat{H}_{j,k} = \left.\frac{\partial^2 \mathcal{L}}{\partial\theta_j \partial\theta_k}\right|_{\theta_j=\hat{\theta}_j, \theta_k=\hat{\theta}_k},$$

is an estimated Hessian matrix of the loss function $\mathcal{L}$. We implement both versions of the Hill bounds based on $\tilde{H}/2$ and $\tilde{J}^T\tilde{J}$ in our R package with the default being the latter, since the latter tends to yield broader Hill bounds. Users can try using both options in real data analysis. The confidence level α is set at 0.0001 as a default and can be given by users as a tuning parameter.

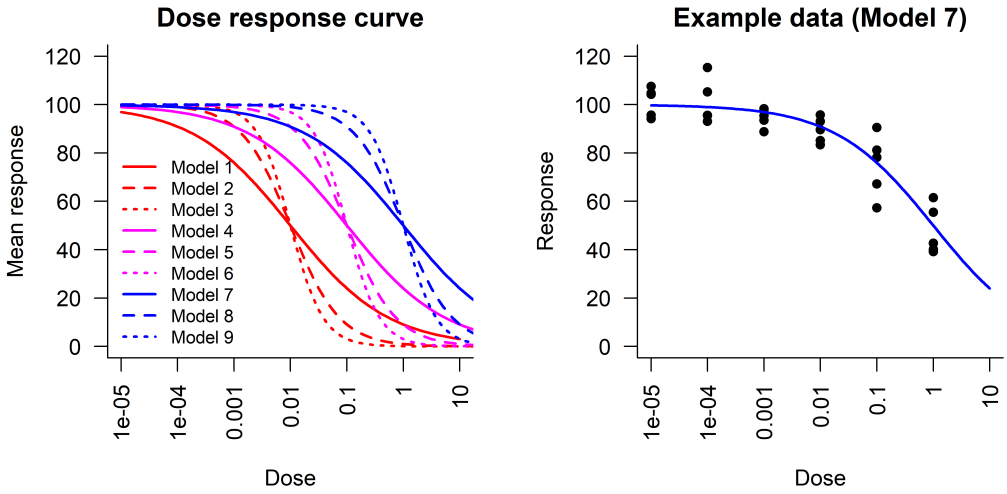


Figure 4: Dose response models presented in Table 2 and an example data set generated by Model 7. The left plot shows the dose response curves of the models in Table 2. The right plot shows data generated by Model 7 of Table 2.

One can wonder if bounds on the upper asymptote θ_1 and lower asymptote θ_4 are needed as well. The following theorem shows that such bounds are unnecessary since diverging values of θ_1 and θ_4 always increase loss function values.

Theorem H.3.1. *Let $\theta_2 > 0$ and $\theta_3 \in \mathbb{R}$ be fixed. Then there exist constants $\theta_1', \theta_4' \in \mathbb{R}$ such that*

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta_1} > 0 \; \forall \theta_1 \geq \theta_1', \; \frac{\partial \mathcal{L}(\theta)}{\partial \theta_4} < 0 \; \forall \theta_4 \leq \theta_4'$$

for fixed $\theta_4 \in \mathbb{R}$ and $\theta_1 \in \mathbb{R}$, respectively. That is, loss function values increase as the values of parameters θ_1 and θ_4 grow to the infinity and the negative infinity, respectively, which prevents their estimates from diverging.

Proof. *See Appendix.* □

Analysis of the Hill bounds

In this subsection, we evaluate whether the Hill bounds are appropriate as a criterion for convergence failure. We simulate data following the 4PL model and check whether the Hill bounds prevent optimization processes from finding the global minimum of the loss function $\mathcal{L}$. To evaluate this, we generate data and count the numbers of times the Hill bounds miss the true parameters. We also count the numbers of times that optimization procedures encounter the Hill bounds at the end of optimization. Lower counts in both cases indicate that the Hill bounds do not hinder optimization processes from converging to desired parameter estimates.

Figure 4 shows parameter settings of our simulation. The left plot shows nine dose-response curves that generate data whose values of parameters are shown in Table 1. The levels of slope parameter θ_3 are denoted by line types while the levels of IC50 parameter are denoted by colors. The right plot shows the dose-response curve of Model 7 with data generated by the model. This model is chosen since it exhibits the most severe support problem among all the nine models. Even though data in the right plot are generated by a normal distribution, it seems that any optimization procedure can have trouble fitting the 4PL model to that data. We simulate 1,000 data sets with five observations at each dose level resulting in 25 observations in total. The standard deviation of the underlying normal distribution of a 4PL model is fixed at 5 throughout the simulations.

Table 2 shows the proportions of times that the Hill bounds miss true parameters of the nine models given in Figure 4. For each simulated data set, we counted the number of times that the Hill bounds miss any of the IC50 and slope parameters. Note from the table that the Hill bounds contain the true parameters for most of the simulated data. The only exception is when data have a severe support problem ($\theta_2 = 1$) and the slope of the dose-response curve is very gradual ($\theta_3 = -0.5$). Since we do not have well-shaped data for this case, we take the fact that the Hill bounds miss the true parameters for these data as an advantage. When the Hill bounds are attained at the end of the optimization, it is recommended to look at the scatter plot, to investigate potential support problems.

Model	Parameters $(\theta_1, \theta_2, \theta_3, \theta_4)$	Model	Parameters $(\theta_1, \theta_2, \theta_3, \theta_4)$
1	(100,0.01,-0.5,0)	6	(100,0.1,-1.5,0)
2	(100,0.01,-1,0)	7	(100,1,-0.5,0)
3	(100,0.01,-1.5,0)	8	(100,1,-1,0)
4	(100,0.1,-0.5,0)	9	(100,1,-1.5,0)
5	(100,0.1,-1,0)		

Table 1: Model numbers and their parameters used in our simulation.

Parameters	$\theta_3 = -0.5$	$\theta_3 = -1$	$\theta_3 = -1.5$
$\theta_2 = 0.01$	0.054	0.014	0
$\theta_2 = 0.1$	0	0	0.002
$\theta_2 = 1$	0.181	0	0

Table 2: Proportions of times out of 1,000 simulations that the Hill bounds miss any of the true IC50 and slope parameters. The Hill bounds contain the true parameters for most of the cases, except when data have a support problem (bottom row) and the slope is gradual (leftmost column).

As illustrated in the top two rows of the table, in cases where the range of the data is enough for estimation on both sides of the IC50, the Hill bounds successfully contain the true parameters.

Table 3 presents the proportion of times the Hill bounds are hit at the end of optimization processes out of 1,000 simulations. The final parameter estimates are inside the Hill bounds for most cases. This indicates that the Hill bounds are not too restrictive for optimization procedures to find a local minimum of the loss function $\mathcal{L}$. However, when there are severe support problems as shown in the bottom left and bottom middle cells, the Hill bounds restrict optimization processes to be inside the bounds. As implied in the previous paragraph, this can help users realize that their data suffer from support problems and actions should be taken.

Remedy for outliers

As seen in the previous section, it seems that the convergence failure results from either outliers or the support problem. We tackle each of the causes with a different approach. Outlier problems are often dealt with by robust estimation methods. The paper Riazoshams et al. (2009) reviewed various methods of outlier detection in the context of nonlinear regression, especially when the form of the mean response function f is known. On the other hand, Dixon's Q test in Dean and Dixon (1951) can be used to detect an outlier without any assumptions on the form of f, but this method is most effective when a data size is less than or equal to 10. In this paper, we use a novel method presented in Motulsky and Brown (2006) that combines robust estimation and outlier detection. This method is currently adopted in GraphPad Prism and called the `ROUT` method therein as a default outlier detection method. The indices of outliers in data and a scatter plot with a robust fit are provided to users to enhance their understanding of problems.

Parameters	$\theta_3 = -0.5$	$\theta_3 = -1$	$\theta_3 = -1.5$
$\theta_2 = 0.01$	0.038	0.009	0.1
$\theta_2 = 0.1$	0.12	0.002	0.195
$\theta_2 = 1$	0.398	0.258	0.079

Table 3: Proportions of times out of 1,000 simulations that the Hill bounds are attained by optimization procedures. The Hill bounds do not prevent optimization processes from achieving local minima of $\mathcal{L}$ for most of cases, except when data have support problems as shown in the bottom row.

Robust regression and outlier detection

As noted in Subsection Motivation: convergence failure, one of the main reasons for convergence failure is strong outliers. In this case, the contour of the loss function $\mathcal{L}(\theta|x,y)$ can exhibit monotonicity as a function of the IC50 parameter θ_2 as shown in Figure 2. This implies that the Hill bounds of Subsection Hill bounds: safeguards on the parameter space can be attained at the end of an optimization process. The R package **dr4pl** reports in this case that there seems to exist a problem in data and it requires a user's attention. Meanwhile, the R package fits a robust regression model to data and reports indices of outliers in data with a plot highlighting the outliers in red triangles, which will be shown in Figure 6. By studying the robust fit and the indicated outliers, the user can get some insight into the data and decide whether the observations denoted as outliers should be removed or not.

To determine which data observations are outliers, we use the method given in Motulsky and Brown (2006). Define the i-th residual of a fitted dose-response model

$$r_i = y_i - f(x_i|\hat{\theta}),$$

where the mean response f is given in Equation (2) and $\hat{\theta}$ is the estimator obtained by a robust regression model. The method of that paper approximates the distribution of r_i by a t-distribution and then adjusts p-values by the *false discovery rate* (*FDR*). See Benjamini and Yekutieli (2001) for explanation of FDR. Adapting their algorithm in the context of the 4PL model yields the following.

1. Fit a dose-response model to data using a robust regression model to obtain residuals r_i for $i = 1, 2, \cdots, n$. Detailed description of a robust regression model is given in the following paragraphs.
2. Compute the *median absolute deviation* (*MAD*) of absolute valued residuals $r_i^{(\mathrm{a})} = |r_i|$ as

   $$\hat{\sigma} = \mathrm{med}_{1\le i\le n}\left\{\left|r_i^{(\mathrm{a})} - \mathrm{med}_{1\le i\le n}\left\{r_i^{(\mathrm{a})}\right\}\right|\right\},$$

   where med stands for the sample median.
3. Rank the absolute values of the residuals from the smallest to the largest, so that $r_{i:n}^{(\mathrm{a})}$ is the i-th order statistic of the absolute valued residuals.
4. Loop from $i = \lfloor 0.7 \times n \rfloor$ to n
   (a) Compute $\alpha_i = 0.01(n-(i-1))/n$ and $t_{i:n} = r_{i:n}^{(a)}/\hat{\sigma}$ for $i = 1, 2, \cdots, n$.
   (b) Compute the two-tailed p-value p_i of the statistic $t_{i:n}$ from the t distribution with $n-4$ degrees of freedom.
   (c) If any of p_j is less than or equal to α_j, then end the loop and report all the observations for $1 \le i \le j$ as outliers.

Contrary to the suggestion of Motulsky and Brown (2006) to use their quantile based measure of scale,

$$\tilde{\sigma} = \frac{n}{n-4} q_{0.6827},$$

where $q_{0.6827}$ is the 0.6827-th quantile of absolute valued residuals, we use the median absolute deviation. The reason is that for small sample sizes around 10, the constant factor $n/(n-4)$ in this equation significantly increases a scale estimate, often resulting in non-rejection of any observations in a data set. For example, the method of Motulsky and Brown (2006) with their estimator $\tilde{\sigma}$ did not detect any observation in Error Case 2 in Figure 2 as an outlier. We adopt the constant 0.01 in the definition of α_i as recommended by Motulsky and Brown (2006).

For robust regression, we adopt M-estimators. The letter M implies that the M-estimators are obtained from equations that are in similar forms with *m*aximum likelihood estimators. The M-estimators are obtained by minimizing

$$\mathcal{L}_\rho(\theta|x,y) = \frac{1}{n}\sum_{i=1}^{n}\rho(r_i),$$

where $\rho : \mathbb{R} \to \mathbb{R}_+$ is a differentiable function. See Huber and Ronchetti (2009) for detailed discussion of the M-estimators. Note that the loss function in Equation (3) is a special case of the loss function $\mathcal{L}_\rho$ when $\rho(r) = r^2$. Various forms of the functions ρ for the M-estimation suggested in the literature are

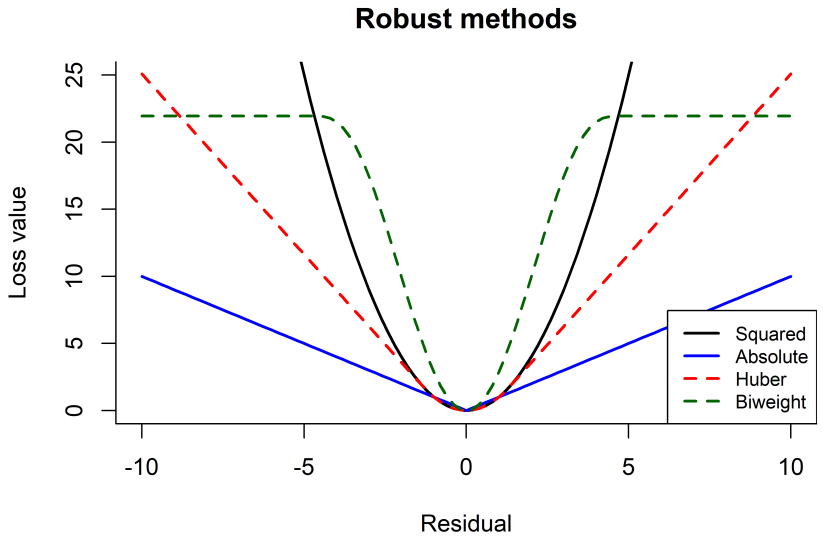


Figure 5: The functions ρ for different M-estimators. The x-axis represents residual values while the y-axis represents the values of the function ρ.

as follows,

$$\begin{aligned} \text{Absolute} : \rho_1(r) &= |r|, \\ \text{Huber} : \rho_H(r) &= \begin{cases} r^2, & |r| \le A_H, \\ 2A_H|r| - A_H^2, & |r| > A_H, \end{cases} \\ \text{Tukey's biweight} : \rho_T(r) &= \begin{cases} r^6/A_T^4 - 3r^4/A_T^2 + 3r^2, & |r| \le A_T, \\ A_T^2, & |r| > A_T, \end{cases} \end{aligned} \tag{13}$$

where $A_H = 1.345$ was suggested by Huber (1964) and $A_T = 4.685$ was suggested by Mosteller and Tukey (1977). The constant A_T was shown in Mosteller and Tukey (1977) to possess 95% efficiency when errors follow a normal distribution, while guaranteeing resistance to contamination of up to 10% of outliers in terms of the breakdown point. These definitions are valid when the standard deviation of the residuals r_i is 1, otherwise a robust estimator of the scale of residuals should divide the residuals.

Figure 5 shows the shape of the ρ functions corresponding to different M-estimators. The x-axis represents residual values while the y-axis represents the values of the function ρ. Note from the figure that the Huber loss and squared loss coincide with each other for residual values less than A_H, but they deviate from each other as the value of the residual r grows larger. The linearity of Huber's loss above A_H prevents a regression model from being heavily affected by outliers that have large absolute values of residuals. Tukey's biweight loss can be thought of as being more robust since it gives constant values to all residuals whose absolute values exceed the cutoff A_T. This implies that a strong outlier with a large absolute value will have the same effect on a regression fit with an observation with an absolute valued residual equal to A_T.

Using M-estimators in nonlinear regression has a few problems. First of all, the conditions for M-estimators to have asymptotic normality are complicated and difficult to be verified as mentioned in Section 12.3 of Seber and Wild (1989). This prevents us from obtaining approximate confidence intervals of true parameters based on M-estimators and their standard errors. On top of that, there are no known general results about consistency of M-estimators in nonlinear regression, which hinders us from trusting fitted parameter estimates by M-estimators. However, the absolute loss function was shown in Oberhofer (1982) to exhibit weak consistency in a nonlinear regression setting. Hence, we set the absolute loss estimator to be a default robust estimation method in our R package when the Hill bounds are attained.

Figure 6 shows the results of applying the R package **dr4pl** to the error cases of Figure 2. Unlike the R packages **drc** and **nplr**, the package **dr4pl** succeeded in fitting a robust regression model to all four data sets. The x-axis represents doses in a log 10 scale and the y-axis represents responses. The black solid line shows a fit obtained by the absolute loss function and red triangles show the outliers detected by the method described in this section. Note from all the plots that the outliers that we pointed out in Section Motivation: convergence failure are marked as outliers. This implies that the method of Motulsky and Brown (2006) is successful in solving the convergence failure problem of the

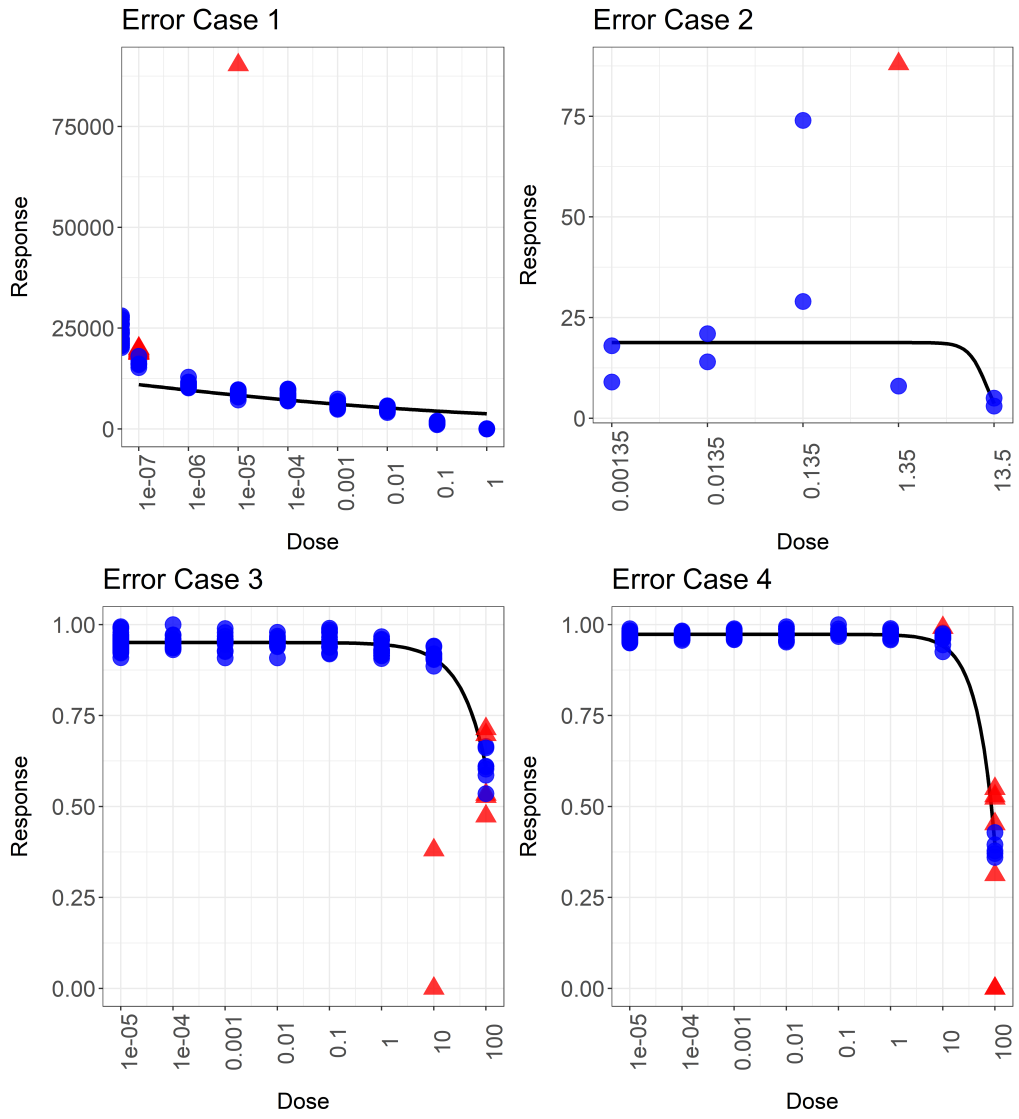


Figure 6: The dose-response curves fitted by the absolute loss function in Equation (13) denoted by black curves and outliers detected by the method of Motulsky and Brown (2006) denoted by red triangles. Note that the curves are reasonably estimated.

4PL model.

Remedy for the support problem

The next cause of the convergence failure is the support problem whose possible solution is presented in this section. Final parameter estimates of the 4PL model obtained by an optimization procedure depend on initial estimates supplied to the procedure. The traditional R packages **drc** and **nplr** use the *logistic method* which computes the initial parameter estimates as

$$\tilde{\theta}_1^{(\mathrm{L})} = y_{\max}, \tilde{\theta}_2^{(\mathrm{L})} = 10^{-\tilde{\beta}_2/\tilde{\beta}_3}, \tilde{\theta}_3^{(\mathrm{L})} = \tilde{\beta}_3, \tilde{\theta}_4^{(\mathrm{L})} = y_{\min}$$

where $y_{\min}, y_{\max}, \tilde{\beta}_2$ and $\tilde{\beta}_3$ are defined in Subsection Hill bounds: safeguards on the parameter space. As demonstrated in Figure 2, those initial parameter estimates do not always lead the optimization process to convergence. We present possible improvements to the logistic method and show a new initialization method that is effective in handling the support problem with data.

Logistic method combined with a grid search

The paper Wang et al. (2010) presented a grid algorithm for fitting the 4PL model to many data sets generated by a high throughput screening experiment. A main idea of their research is that fitting the Hill equation in Equation (6) with multiple initial values of the upper and lower asymptotes, θ_1 and θ_4, and starting from the best parameter estimates can yield a better result. That is, we search the grids G_1 and G_4 on θ_1 and θ_4 for the initial estimates $\left(\tilde{\theta}_1, \tilde{\theta}_2, \tilde{\theta}_3, \tilde{\theta}_4\right)$ with the least loss value where $\tilde{\theta}_1 \in G_1, \tilde{\theta}_4 \in G_4$ and the other two estimates $\tilde{\theta}_2$ and $\tilde{\theta}_3$ are obtained by the Hill equation in Equation (6). The paper suggested naive grids on θ_1 and θ_4 based on constant multiples of $y_{\max}$ and $y_{\min}$, but we suggest new grids that statistically make sense.

The basic idea of choosing grids resembles the idea of the Hill bounds in that it uses asymptotic normality of parameter estimators presented in Equation (10). Since we use $\tilde{\theta}_1^{(\mathrm{L})} = y_{\max}$ and $\tilde{\theta}_4^{(\mathrm{L})} = y_{\min}$ as initial estimators of the upper and lower asymptote parameters in the logistic method, we can obtain approximate standard errors of those estimators from Equation (12). Based on these standard errors, we set the grids on θ_1 and θ_4 as

$$\tilde{\theta}_1 \in \left\{\tilde{\theta}_1^{(\mathrm{L})} - 2\mathrm{s.e.}\left(\tilde{\theta}_1^{(\mathrm{L})}\right), \tilde{\theta}_1^{(\mathrm{L})} - \mathrm{s.e.}\left(\tilde{\theta}_1^{(\mathrm{L})}\right), \tilde{\theta}_1^{(\mathrm{L})}, \tilde{\theta}_1^{(\mathrm{L})} + \mathrm{s.e.}\left(\tilde{\theta}_1^{(\mathrm{L})}\right), \tilde{\theta}_1^{(\mathrm{L})} + 2\mathrm{s.e.}\left(\tilde{\theta}_1^{(\mathrm{L})}\right)\right\},$$
$$\tilde{\theta}_4 \in \left\{\tilde{\theta}_4^{(\mathrm{L})} - 2\mathrm{s.e.}\left(\tilde{\theta}_4^{(\mathrm{L})}\right), \tilde{\theta}_4^{(\mathrm{L})} - \mathrm{s.e.}\left(\tilde{\theta}_4^{(\mathrm{L})}\right), \tilde{\theta}_4^{(\mathrm{L})}, \tilde{\theta}_4^{(\mathrm{L})} + \mathrm{s.e.}\left(\tilde{\theta}_4^{(\mathrm{L})}\right), \tilde{\theta}_4^{(\mathrm{L})} + 2\mathrm{s.e.}\left(\tilde{\theta}_4^{(\mathrm{L})}\right)\right\},$$

where $\mathrm{s.e.}\left(\tilde{\theta}_j^{(\mathrm{L})}\right) = s\sqrt{\tilde{c}_{j,j}}$ for $j = 1, 4$, and the two estimators s and $\tilde{c}_{j,j}$ are defined in Equations (11) and (12), respectively. If any of data observations is greater than $\tilde{\theta}_1$ or smaller than $\tilde{\theta}_4$, then those observations are removed from data and the logistic method proceeds with remaining observations.

Mead's method

An advanced method of obtaining initial parameter estimates is illustrated in Subsection 8.3.2 of Ratkowsky (1983). The method is basically a variant of the method given in Mead (1970) which used a weighted least squares criterion to obtain an approximate solution of the ordinary least squares problem. We call this *Mead's method* in this paper. Unlike the logistic method, Mead's method estimates the upper asymptote and the IC50 parameter from a linear model by first fixing the slope and lower asymptote. Since Mead's method performs a grid search on the slope parameter, it can better obtain initial parameter estimates when there are support problems like those in the lower row of Figure 2. In such cases, obtaining a proper initial estimate of the slope parameter is difficult and performing a grid search for it can be useful.

Mead's method starts from a model

$$Y = f(x|\mu)\epsilon, \tag{14}$$

where

$$f(x|\mu) = \frac{1}{\mu_1 + \mu_2\mu_3^x}, \tag{15}$$

the parameters satisfy $\mu_1, \mu_2, \mu_3 > 0$, x is a *log* 10 dose and ϵ follows a log-normal distribution, i.e. $log(\epsilon) \sim \mathcal{N}(0, \sigma^2)$. We call this model the *multiplicative error model*. This model is easy to work with because its parameters are related to characteristics of dose-response curves like the 4PL model. For example,

$$\text{if } 0 < \mu_3 < 1, \text{ then } \lim_{x\to-\infty} f(x|\mu) = 0, \ \lim_{x\to\infty} f(x|\mu) = \frac{1}{\mu_1},$$
$$\text{if } \mu_3 > 1, \text{ then } \lim_{x\to-\infty} f(x|\mu) = \frac{1}{\mu_1}, \ \lim_{x\to\infty} f(x|\mu) = 0.$$

This implies that $1/\mu_1$ is the upper asymptote of the model and zero is its lower asymptote. The IC50 of the multiplicative error model is more complicated than that of the 4PL model as can be seen in the following,

$$f\left(x = log_{\mu_3}\left(\frac{\mu_1}{\mu_2}\right)\middle|\,\mu\right) = \frac{1}{2\mu_1}.$$

The parameters of a multiplicative error model are related to those of a 4PL model by

$$\theta_1 = \frac{1}{\mu_1}, \ \theta_2 = \left(\frac{\mu_1}{\mu_2}\right)^{1/log_{10}\mu_3}, \ \theta_3 = log_{10}\mu_3, \ \theta_4 = 0.$$

It seems that the 4PL model has been preferred due to these more intuitive meanings of the parameters. This equation shows that the parameter μ_3 is related to the slope of a dose-response curve.

To find estimators of μ_1, μ_2, μ_3 in Equations (14) and (15), we consider a weighted least squares loss function

$$\mathcal{L}(\mu|x,y) = \sum_{i=1}^{n} \frac{\sigma^2}{\text{Var}\,(Y_i)} \left\{y_i - E\,(Y_i)\right\}^2. \tag{16}$$

For a differentiable function $g : \mathbb{R} \to \mathbb{R}$ we have

$$g\,(Y_i) - g\,(E\,(Y_i)) \approx \left.\frac{\mathrm{d}g(y)}{\mathrm{d}y}\right|_{y=E(Y_i)} \left\{Y_i - E\,(Y_i)\right\}.$$

Hence, we can approximate $\text{Var}\,(g\,(Y_i))$ by

$$\text{Var}\,(g\,(Y_i)) \approx \left.\frac{\mathrm{d}g(y)}{\mathrm{d}y}\right|^2_{y=E(Y_i)} \text{Var}\,(Y_i).$$

If we let $g(y) = log(y)$, then we have

$$\text{Var}\,(Y_i) \approx \text{Var}\,(logY_i)\left\{E\,(Y_i)\right\}^2 = \sigma^2\left\{E\,(Y_i)\right\}^2.$$

Substituting this into Equation (16) yields

$$\mathcal{L}(\mu|x,y) \approx \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{y_i}{E\,(Y_i)} - 1\right\}^2.$$

Substituting the expression $E\,(Y_i) = f(x_i|\mu)$ into this equation yields an approximate linear model

$$\mathcal{L}(\mu|x,y) \approx \frac{1}{n}\sum_{i=1}^{n}\left\{y_i\left(\mu_1 + \mu_2\mu_3^{x_i}\right) - 1\right\}^2. \tag{17}$$

To obtain estimates of μ_1 and μ_2, we take $\mu_3 \in G$ with the grid

$$G = \{q_{0.04}, q_{0.08}, \cdots, q_{0.92}, q_{0.96}\}$$

where q_p is the p-th quantile of the Cauchy distribution. The Cauchy distribution was chosen because it can provide a broad range of quantile values due to its heavy tailedness. For each $\mu_3 \in G$, the loss function $\mathcal{L}(\mu|x,y)$ can be minimized over μ_1 and μ_2 by the standard linear regression. That is, a linear regression model with the response 1, the first predictor y_i and the second predictor $y_i\mu_3^{x_i}$ without an intercept can be fitted to obtain parameter estimates. Note that this regression model is fitted for all $\mu_3 \in G$ and parameter estimates $\tilde{\mu}_1, \tilde{\mu}_2, \tilde{\mu}_3$ with the smallest value of $\mathcal{L}(\cdot|x,y)$ are chosen as initial parameter estimates. A nonlinear optimization method starts from these estimates to find final parameter estimates.

Comparison of the logistic and Mead methods

As pointed out in the previous subsection, Mead's method is expected to find better initial parameter estimates than the logistic method when data have the support problem. By simulating data and fitting 4PL models with different initialization techniques, we can see whether the Mead method actually results in final fitted models with lower loss values than the logistic method. We also compare the two methods when data do not exhibit a support problem. By doing that, we can get insight into their relative strengths under various situations.

In this simulation study, we generate data from the models of Subsection Analysis of the Hill bounds whose mean response curves are presented in the left plot of Figure 4 and whose parameter values are given in Table 2. As mentioned in that subsection, Models 1, 2 and 3 represent models without any support problem, while Models 7, 8 and 9 represent severe support problems. Models 4, 5 and 6 exhibit mild levels of support problems. For each model, 1,000 data sets are generated and the standard deviation of normally distributed errors was fixed at 5.

Table 4 shows the numbers of times that the Mead method outperforms the logistic method in the sense that it has a lower loss value for final fitted models out of 1,000 simulations. The top row of the table represents Models 1, 2 and 3 from left to right. As can be seen from the top row, the Mead method generally performs worse than the logistic method except when the slope is very steep ($\theta_3 = -1.5$). In the middle and bottom rows, the Mead method performs better than the logistic method. The degree

Parameters	$\theta_3 = -0.5$	$\theta_3 = -1$	$\theta_3 = -1.5$
$\theta_2 = 0.01$	442	443	697
$\theta_2 = 0.1$	651	568	682
$\theta_2 = 1$	661	767	843

Table 4: The numbers of times that the Mead method outperforms the logistic method in the sense that it has a lower loss value for final fitted models. The bottom row ($\theta_2 = 1$) represents data with a support problem. It can be seen that the superiority of Mead's method is maximal when data have a support problem.

of outperformance becomes larger when the support problem is more severe ($\theta_2 = 1$) or the slope parameter is more negative ($\theta_3 = -1.5$). This coincides with our conjecture made in the previous subsection that Mead's method can be particularly useful when data do not sufficiently exist on the right side of the dose-response curves so that estimating the slope and lower asymptote parameters is difficult. In addition, the degree of inferiority of the Mead method to the logistic method when there is no support problem (upper left cell) is not severe, which supports our use of the Mead method as the default initialization method in **dr4pl**.

Walk-through in R

In this section, we walk through example R code in which dose-response data are analyzed using our R package **dr4pl**. By following steps shown below, users can understand how to use **dr4pl** to handle data sets that have the convergence failure issue and what kind of detailed analysis can be done by setting options of R functions in **dr4pl**.

Main R function dr4pl

We start with Error Case 1 shown in Figures 2 and 6. As can be seen in those figures, this data set contains an extreme outlier at the dose level 10^{-5}. If the R package **drc** is used to fit the 4PL model to this data set, the following error message is returned.

```
> library(drc)
>
> drm(Response~Dose, data = drc_error_1, fct = LL.4())
Error in optim(startVec, opfct, hessian = TRUE, method = optMethod, control =
  list(maxit = maxIt,  :  non-finite finite-difference value [4]
Error in drmOpt(opfct, opdfct1, startVecSc, optMethod, constrained, warnVal,  :
  Convergence failed
```

Note that this error message indicates convergence failure but no explanation about its cause is given. On the contrary, fitting the 4PL model with **dr4pl** returns an object of the class `"dr4pl"` which contains useful information about the convergence failure and a possible solution.

```
> library(dr4pl)
>
> dr4pl.error.1 <- dr4pl(Response~Dose, data = drc_error_1, method.init = "logistic",
+                        use.Hessian = TRUE)
> class(dr4pl.error.1)
[1] "dr4pl"
> dr4pl.error.1$convergence  # TRUE indicates convergence success,
+                            # FALSE indicates convergence failure.
[1] FALSE
```

In this code, some input arguments such as `method.init` and `use.Hessian` are specified as different values from the default values of **dr4pl**. For example, the default value of `method.init` is `Mead` but it is specified as `logistic` in the code. This is to ensure that the function `dr4pl` works with the same parameter setting as **drc**. The R package **drc** uses the logistic method as its default initialization method and the Hessian matrix in its confidence interval estimation. Note that the function `dr4pl` does not throw an error but returns the member `dr4pl.error.1$convergence` which indicates that convergence failure happens during the optimization process.

As a possible solution to the convergence failure, the object `dr4pl.error.1` provides a member object `dr4pl.robust` that results from robust estimation and the outlier detection algorithm explained in Subsection Robust regression and outlier detection. We call the object `dr4pl.error.1$dr4pl.robust` the *robust child object* of its *parent object* `dr4pl.error.1`.

```
> dr4pl.robust.1 <- dr4pl.error.1$dr4pl.robust
> class(dr4pl.error.1$dr4pl.robust)
[1] "dr4pl"
> dr4pl.robust.1$convergence
[1] TRUE
```

It can be seen from these code lines that the robust child object does not report the convergence failure issue. To check results of robust estimation and outlier detection, we check the plot of the object `dr4pl.robust.1`.

```
> dr4pl.robust.1$idx.outlier
 [1] 132 140 127 134 133 137 141 139 130 129 128 102
> plot(dr4pl.robust.1, indices.outlier = dr4pl.robust.1$idx.outlier)
```

The member `idx.outlier` of the object `dr4pl.robust.1` shows the indices of the outliers in the input data frame `drc_error_1` detected by our algorithm. It can be seen that the last sentence in this code generates the same plot as the upper left plot of Figure 6. As mentioned in Subsection Robust regression and outlier detection, the absolute loss is given as the default robust estimation method. This implies that adopting the absolute loss successfully resolves the convergence failure problem of Error Case 1.

Our R package **dr4pl** provides other real-world data sets than Error Cases 1 - 4. We analyze the data set `sample_data_5` to compare the two different initialization methods, the logistic and Mead methods, which were presented in Section Remedy for the support problem.

```
> dr4pl.logistic.5 <- dr4pl(Response~Dose,
+                           data = sample_data_5,
+                           method.init = "logistic")
> ggplot.logistic <- plot(dr4pl.logistic.5, text.title = "Logistic method")
>
> dr4pl.Mead.5 <- dr4pl(Response~Dose,
+                       data = sample_data_5)
> ggplot.Mead <- plot(dr4pl.Mead.5, text.title = "Mead's method")
>
> grid.arrange(ggplot.logistic, ggplot.Mead, nrow = 1, ncol = 2)
```

Running this code generates the two plots in Figure 7. Note that outliers are not denoted by red triangles in these plots since they are generated by the parent objects but not by their robust child objects. It can be seen from the figure that the data set has the support problem in the sense that it is hard to determine an IC50 parameter value visually and data points on the right side of possible IC50 parameter values are not sufficiently provided. Mead's method shown in the right plot seems to yield a closer fit to the right-most data point. This coincides with our explanation in Subsection Mead's method that the method can result in better final parameter estimates particularly when data suffer from the support problem.

Auxiliary R functions

The R function `summary` of the R package **dr4pl** shows the 95% asymptotic confidence intervals of the final parameter estimates of a `"dr4pl"` object. The following code lines show example output of `summary` applied to the data set `sample_data_3`.

```
> dr4pl.3 <- dr4pl(Response~Dose, data = sample_data_3)
> summary(dr4pl.3)
$call
dr4pl.formula(formula = Response ~ Dose, data = sample_data_3)

$coefficients
                Estimate       StdErr         2.5 %        97.5 %
UpperLimit 59858.0700194 1.328219e-01 58678.1496415 61037.9903974
IC50           5.0794918 1.531608e-05     3.7133017     6.9483278
Slope         -0.6117371 1.440712e-05    -0.7397224    -0.4837518
```

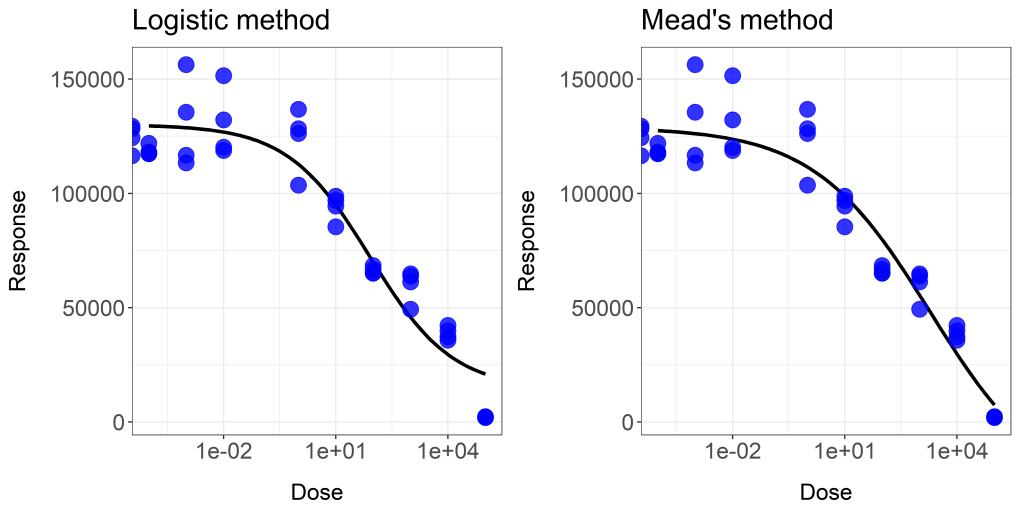


Figure 7: Dose-response curves fitted by **dr4pl** with two initialization methods: the logistic (left plot) and Mead (right plot) method. It can be seen that Mead's method yields a closer fit to the right-most data point than the logistic method.

```
LowerLimit   980.8205686 1.354448e-01  -222.4002527  2184.0413899

attr(,"class")
[1] "summary.dr4pl"
```

Recall from Subsection H.2.1 that the IC50 is the dose level whose response value is halfway between the upper and lower asymptotes θ_1 and θ_4. Similarly, other inhibitory concentrations of interest can be defined according to different proportions between θ_1 and θ_4 as ι_p which satisfies the following equation,

$$f(\iota_p) = p\theta_1 + (1-p)\theta_4$$

where $0 \leq p \leq 1$. It can be easily seen that $\iota_{0.5}$ corresponds to the IC50 parameter θ_2.

Like the R function `ED` of **drc** and the R function `getEstimates` of **nplr**, the R package **dr4pl** implements the function `IC` to compute inhibitory concentration parameters ι_p for $0 \leq p \leq 1$. The following code shows a simple use of `IC` to compute $\iota_{0.1}, \iota_{0.3}, \iota_{0.5}, \iota_{0.7}, \iota_{0.9}$ for the data `sample_data_3`.

```
> IC(dr4pl.3, c(10, 30, 50, 70, 90))
InhibPercent:10 InhibPercent:30 InhibPercent:50 InhibPercent:70 InhibPercent:90
    184.3784198      20.2930770       5.0794918       1.2714305       0.1399363
```

Summary and future extension

The 4PL model has often suffered from convergence failure problems preventing experimenters from obtaining fitted 4PL models to data sets. Without diagnosing problems with data sets, experimenters should often try many different initial parameter estimates, which still does not guarantee stable convergence of optimization processes. In this paper, we presented a novel method based on studying convergence failure problems. By imposing proper conditions on the parameter space of the 4PL models, our new R package **dr4pl** reports convergence failure during an optimization process and presents possible resolutions to users.

Another important issue can be memory usage and speed of **dr4pl**, relative to other R packages **nplr** and **drc**. Measuring time and memory consumed by the three R packages for the 13 data sets provided in **dr4pl** showed that **nplr** is much slower, and used much more memory, than either of the other two methods. The enhanced features of **dr4pl** resulted in its being only slightly slower, with somewhat more memory usage than **drc**.

Asymptotic confidence intervals of M-estimators have been developed in Lim et al. (2013). An interesting open problem is the numerical implementation of these in the 4PL context. Another area of potential future extension is modelling interaction between different drugs. The R packages **drc** and **MixLow** support functionality for drug interaction modelling. While we are not aware of data sets reported in the literature that cause the convergence failure problem when there are more than one

drug, it seems that outliers or the support problem may cause instability of optimization processes in those cases as well. Analysis of the Hill bounds can be a remedy for those, which is an important area of future research.

Appendix: proof of Theorem H.3.1

Let $\Theta = \{\theta = (\theta_1, \theta_2, \theta_3, \theta_4) | \theta_1, \theta_3, \theta_4 \in \mathbb{R}, \theta_2 > 0\}$. From Equation (3), it can be seen that

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta_j} = -\frac{2}{n} \sum_{i=1}^{n} \{y_i - f(x_i|\theta)\} \frac{\partial f(x_i|\theta)}{\partial \theta_j}, \tag{18}$$

for $j = 1, 2, 3, 4$. In addition, we have from Equation (2) that

$$\begin{aligned} \frac{\partial f(x|\theta)}{\partial \theta_1} &= 1 - \frac{1}{1 + (10^x/\theta_2)^{\theta_3}} > 0, \\ \frac{\partial f(x|\theta)}{\partial \theta_4} &= \frac{1}{1 + (10^x/\theta_2)^{\theta_3}} > 0, \end{aligned} \tag{19}$$

for all $x \in \mathbb{R}$ and $\theta \in \Theta$ since we have $\theta_2 > 0$. From these equations, it can be seen that the function f is increasing in θ_1 and θ_4 and

$$\lim_{\theta_1 \to \infty} f(x|\theta) = \infty, \ \lim_{\theta_4 \to -\infty} f(x|\theta) = -\infty.$$

for all $x \in \mathbb{R}$ and $\theta \in \Theta$. Fix $\theta_2 > 0, \theta_3 \in \mathbb{R}$. Then there exist two constants $-\infty < \theta_1', \theta_4' < \infty$ such that

$$f(x_i|\theta) > y_i \ \forall \theta_1 \geq \theta_1', \ f(x_i|\theta) < y_i \ \forall \theta_4 \leq \theta_4', \tag{20}$$

for fixed $\theta_4 \in \mathbb{R}$ and $\theta_1 \in \mathbb{R}$, respectively, for $i = 1, 2, \cdots, n$. Substituting Equations (19) and (20) into Equation (18) yields

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta_1} > 0 \ \forall \theta_1 \geq \theta_1', \ \frac{\partial \mathcal{L}(\theta)}{\partial \theta_4} < 0 \ \forall \theta_4 \leq \theta_4',$$

for those fixed $\theta_4 \in \mathbb{R}$ and $\theta_1 \in \mathbb{R}$, respectively.

Bibliography

D. M. Bates and D. G. Watts. Nonlinear regression analysis and its applications. *Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. John Wiley & Sons*, 1988. ISBN 0-471-81643-4. URL https://doi.org/10.1002/9780470316757. [p172]

Y. Benjamini and D. Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Ann. Statist.*, 29(4):1165–1188, 2001. ISSN 0090-5364. [p180]

F. Commo and B. M. Bot. nplr: n-parameter logistic regression, 2016. URL https://CRAN.R-project.org/package=nplr. R package version 0.1-7. [p172]

R. B. Dean and W. J. Dixon. Simplified statistics for small numbers of observations. *Analytical Chemistry*, 23(4):636–638, 1951. URL https://doi.org/10.1021/ac60052a025. [p179]

Y. Di Veroli Giovanni, C. Fornari, I. Goldlust, G. Mills, S. B. Koh, J. L. Bramhall, F. M. Richards, and D. I. Jodrell. An automated fitting procedure and software for dose-response curves with multiphasic features. *Scientific Reports (Nature Publisher Group)*, 5:14701, 2015. URL https://doi.org/10.1038/srep14701. [p173]

S. R. Gadagkar and G. B. Call. Computational tools for fitting the Hill equation to dose-response curves. *J Pharmacol Toxicol Methods*, 71:68–76, 2015. URL https://doi.org/10.1016/j.vascn.2014.08.006. [p175]

J. Giraldo, N. M. Vivas, E. Vila, and A. Badia. Assessing the (a) symmetry of concentration-effect curves: empirical versus mechanistic models. *Pharmacology & Therapeutics*, 95(1):21–45, 2002. URL https://doi.org/10.1016/s0163-7258(02)00223-1. [p172]

A. V. Hill. The possible effects of the aggregation of the molecules of haemoglobin on its dissociation curves. *J Physiol (Lond)*, 40:4–7, 1910. [p172]

P. J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35:73–101, 1964. ISSN 0003-4851. URL http://dx.doi.org.libproxy.lib.unc.edu/10.1214/aoms/1177703732. [p181]

P. J. Huber and E. M. Ronchetti. Robust statistics. *Wiley Series in Probability and Statistics. John Wiley & Sons, 2nd edition*, 2009. ISBN 978-0-470-12990-6. URL http://dx.doi.org.libproxy.lib.unc.edu/10.1002/9780470434697. [p180]

J. T. Landis, H. An, and A. G. Bailey. dr4pl: dose response data analysis using the 4 Parameter Logistic (4PL) model, 2018. URL https://bitbucket.org/dittmerlab/dr4pl. R package version 1.1.8. [p171]

C. Lim. Robust ridge regression estimators for nonlinear models with applications to high throughput screening assay data. *Stat. Med.*, 34(7):1185–1198, 2015. ISSN 0277-6715. URL https://doi.org/10.1002/sim.6391. [p173]

C. Lim, P. K. Sen, and S. D. Peddada. Robust nonlinear regression in applications. *J. Indian Soc. Agricultural Statist.*, 67(2):215–234, 291, 2013. ISSN 0019-6363. [p187]

R. Mead. Plant density and crop yield. *Applied statistics*, pages 64–81, 1970. URL https://doi.org/10.2307/2346843. [p183]

F. Mosteller and J. W. Tukey. Data analysis and regression: a second course in statistics. *Addison-Wesley Series in Behavioral Science: Quantitative Methods*, 1977. [p181]

H. J. Motulsky and R. E. Brown. Detecting outliers when fitting data with nonlinear regression - a new method based on robust nonlinear regression and the false discovery rate. *BMC Bioinformatics*, 7:123, 2006. URL https://doi.org/10.1186/1471-2105-7-123. [p179, 180, 181, 182]

W. Oberhofer. The consistency of nonlinear regression minimizing the L_1-norm. *Ann. Statist.*, 10(1):316–319, 1982. ISSN 0090-5364. [p181]

W. B. Pratt and P. Taylor. Principles of drug action: the basis of pharmacology. *Churchill Livingstone, 3rd edition*, 1990. URL https://doi.org/10.1021/jm00296a900. [p177]

D. A. Ratkowsky. Nonlinear regression modeling. A unified practical approach. *Marcel Dekker, Inc., New York*, 1983. [p183]

A. H. Riazoshams, B. M. Habshah Jr, and A. Adam. On the outlier detection in nonlinear regression. *International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering*, 3(12):1105–1111, 2009. [p179]

C. Ritz, J. C. Streibig, and others. Bioassay analysis using R. *Journal of Statistical Software*, 12(5):1–22, 2005. URL https://doi.org/10.18637/jss.v012.i05. [p172]

T. B. Robertson. On the normal rate of growth of an individual, and its biochemical significance. *Archiv für Entwicklungsmechanik der Organismen*, 25(4):581–614, 1908. URL https://doi.org/10.1007/bf02163864. [p171]

G. A. F. Seber and C. J. Wild. Nonlinear regression. *Wiley Series in Probability and Mathematical Statistics: Probability and Mathematical Statistics. John Wiley & Sons*, 1989. ISBN 0-471-61760-1. URL https://doi.org/10.1002/0471725315. [p172, 177, 181]

Y. Wang, A. Jadhav, N. Southal, R. Huang, and D. T. Nguyen. A grid algorithm for high throughput fitting of dose-response curve data. *Curr Chem Genomics*, 4:57–66, 2010. URL https://doi.org/10.2174/1875397301004010057. [p173, 183]

Hyowon An
Department of Statistics and Operations Research
The University of North Carolina at Chapel Hill
USA
ahwbest@gmail.com

Justin T. Landis
Lineberger Comprehensive Cancer Center
The University of North Carolina at Chapel Hill
USA
jtlandis314@gmail.com

Aubrey G. Bailey
Lineberger Comprehensive Cancer Center
The University of North Carolina at Chapel Hill
USA
aubreybailey@gmail.com

J. S. Marron
Department of Statistics and Operations Research
The University of North Carolina at Chapel Hill
USA
marron@unc.edu

Dirk P. Dittmer
Lineberger Comprehensive Cancer Center
The University of North Carolina at Chapel Hill
USA
dirkdittmer@me.com

cvcrand: A Package for Covariate-constrained Randomization and the Clustered Permutation Test for Cluster Randomized Trials

by Hengshi Yu, Fan Li, John A. Gallis and Elizabeth L. Turner

Abstract The cluster randomized trial (CRT) is a randomized controlled trial in which randomization is conducted at the cluster level (e.g., school or hospital) and outcomes are measured for each individual within a cluster. Often, the number of clusters available to randomize is small (≤ 20), which increases the chance of baseline covariate imbalance between comparison arms. Such imbalance is particularly problematic when the covariates are predictive of the outcome because it can threaten the internal validity of the CRT. Pair-matching and stratification are two restricted randomization approaches that are frequently used to ensure balance at the design stage. An alternative, less commonly-used restricted randomization approach is covariate-constrained randomization. Covariate-constrained randomization quantifies baseline imbalance of cluster-level covariates using a balance metric and randomly selects a randomization scheme from those with acceptable balance by the balance metric. It is able to accommodate multiple covariates, both categorical and continuous. To facilitate implementation of covariate-constrained randomization for the design of two-arm parallel CRTs, we have developed the **cvcrand** R package. In addition, **cvcrand** also implements the clustered permutation test for analyzing continuous and binary outcomes collected from a CRT designed with covariate-constrained randomization. We used a real cluster randomized trial to illustrate the functions included in the package.

Introduction

Cluster randomized trials (CRTs) randomize clusters of individuals, such as schools, hospitals or clinics (Brown and Li, 2015). The CRT design is chosen when there are concerns of treatment contamination, when it is logistically easier to conduct the trial using cluster randomization and when intervention of interest is delivered at the group level (Turner et al., 2017a). CRTs have been used in many disciplines including social sciences, public policy, medicine and implementation science (Hayes and Moulton, 2009).

In this paper, we focus on the two-arm parallel cluster randomized trial. Usually, there are a total of $\binom{n}{n_T} = \frac{n!}{n_T!(n-n_T)!}$ ways to allocate n_T clusters to the intervention arm, out of a total of n clusters. For example, in a CRT with 10 clusters, 5 of which are assigned to the treatment arm and 5 to the control arm, there are $\binom{10}{5} = 252$ unique allocations in the simple randomization space. Each allocation is called a randomization scheme and when simple randomization is used, one of the 252 allocations is randomly selected and implemented in the CRT. Because it is common that there are only a limited (usually fewer than 20) number of clusters available in a CRT (Fiero et al., 2015), there may be a non-negligible chance of imbalance between arms regarding the distribution of baseline covariates (Moulton, 2004). If the covariates are predictive of the outcome, such imbalance may threaten the internal validity, can lead to loss of power and usually requires statistical adjustment in the analysis stage (Ivers et al., 2012).

Several design strategies are available to avoid reliance only on statistical adjustment that accounts for baseline covariate imbalance in the analysis phase. Two most popular ones are pair-matching and stratification (Ivers et al., 2012), both of which are examples of restricted randomization. Matching pairs of clusters according to similarity in the baseline covariate profile (e.g., location), and performs randomization within each pair. Stratification is similar to matching but instead of only considering pairs of clusters, the procedure forms strata of 2 or more clusters where each stratum includes clusters with similar baseline covariate profiles. As with matching, the clusters within each stratum are then randomized into the two arms and, when there are an even number of clusters in each stratum, there is perfect balance of strata across treatment and control arms. There are several limitations of these procedures. The power of a pair-matched study might decrease due to a small number of pairs and a small correlation between the matching covariates and the outcome (Diehr et al., 1995). Loss to follow-up of a single cluster may require the exclusion of the matched pair in the analysis, and reduces study power (Ivers et al., 2012). In addition, the intracluster correlation coefficient is not easy to compute from the matched pairs (Donner and Klar, 2004; Klar and Donner, 1997; Campbell et al., 2012). In a CRT with a small number of clusters, stratified randomization is only possible with a small

number of stratification covariates. Otherwise, a single cluster might be in a stratum and will cause an imbalance between the arms (Ivers et al., 2012). Given that CRTs often identify and recruit all clusters at the start of the trial, minimization, a common restricted randomization for individually randomized trials, is rarely applicable to CRTs. To deal with these limitations especially with a small number of clusters and more than a few baseline covariates to balance, alternative restricted randomization methods are necessary.

Covariate-constrained randomization is an alternative restricted randomization procedure (Ivers et al., 2012; Raab and Butcher, 2001). Unlike matching and stratification, covariate-constrained randomization uses a measure called a balance metric to quantify the difference in mean covariate values between the two arms for a given randomization scheme across all baseline covariates that we wish to balance. The simple randomization space is then constrained by keeping the subset of randomization schemes with which covariates are considered sufficiently balanced by the balance metric. A final scheme is then selected from this constrained space, and tends to exhibit better baseline balance on average than a scheme randomly selected without constraints. Compared with pair-matching and stratification, covariate-constrained randomization may be preferred due to its capacity to accommodate multiple covariates, both categorical and continuous. Further, the ICC calculation remains unaffected under constrained randomization.

Although covariate-constrained randomization is a promising design strategy for CRTs, it is not commonly used in practice. One possible reason is that it requires more programming than simple randomization, pair-matching or stratification. Therefore, to facilitate its implementation in the design and analysis of cluster randomized trials, we have developed the `cvrall` and `cvrcov` functions in the **cvcrand** package. The `cvrall` function performs constrained randomization based on covariate balance measured by a scalar balance metric and can assign weights to reflect the relative importance of candidate covariates. The `cvrcov` function performs constrained randomization based on multivariate balance defined through each single covariate, similar to the routine provided in Greene (2017).

From an analysis perspective, when a CRT is designed using covariate-constrained randomization, this design feature should be reflected in the analysis of the individual-level outcome data collected during the trial (Li et al., 2016, 2017). To do so, a permutation-based approach can be used. The permutation test, discussed in Gail et al. (1996), should account for the variability within the constrained randomization space. In other words, the resulting clustered permutation test obtains the p-value for the treatment effect by referencing the observed test statistics to the permutation distribution within the constrained space. We provide the `cptest` function in the **cvcrand** package to facilitate the implementation of this permutation test.

Methods

To demonstrate the utility of the **cvcrand** package, we describe the concepts of covariate-constrained randomization and the clustered permutation test using an example of a real cluster randomized trial presented in Dickinson et al. (2015). This CRT aims to compare a collaborative centralized reminder approach with a practice-based reminder approach for increasing the immunization rate in children aged 19 to 35 months from 16 counties in Colorado. Each county represents a cluster of children and eight counties are randomized to each arm. The collaborative reminder approach depends on the joint efforts between health department leaders and physicians to develop a centralized notification, either using telephone or mail, for all parents whose pre-school children are not up-to-date on immunizations. Parents from the practice-based arm are invited to attend a web-based training for reminder using the Colorado Immunization Information System. Although counties are the randomization unit, the binary outcome, immunization status, is to be measured for participating children. A list of nine county-level covariates are collected (see Table 1 for the complete list, of which income is listed twice as it is coded as both a continuous variable and a derived categorical variable) prior to randomization, and balance on these covariates are desired during the randomization phase.

Covariate-constrained randomization

Covariate-constrained randomization, henceforth referred to simply as constrained randomization, is a promising balancing technique for cluster randomized trials (CRTs), especially for those with a limited number of clusters (Hayes and Moulton, 2009). Constrained randomization usually involves the following steps: (i) specifying the baseline covariates that one wishes to balance; (ii) enumerating all possible randomization schemes or randomly simulating a large number of randomization schemes within the simple randomization space (duplicates are removed if the schemes are randomly simulated); (iii) retaining a constrained randomization space with a subset of schemes where sufficient balance across baseline covariates is achieved according to some pre-specified balance metric; (iv) randomly selecting a scheme from the constrained randomization space for implementation.

Table 1: County-level variables in the motivating example.

Variable name	Variable description
`location`	location ("rural" or "urban")
`inciis`	percentage of children aged 19-35 months in the Colorado Immunization Information System (CIIS)
`numberofchildrenages1935months`	number of children aged 19-35 months
`uptodateonimmunizations`	percentage of up-to-date on immunizations
`africanamerican`	percentage of African American
`hispanic`	percentage of Hispanic ethnicity
`income`	average income ($)
`incomecat`	category of average income ("low", "medium", and "high")
`pediatricpracticetofamilymedicin`	pediatric practice-to-family medicine practice ratio
`communityhealthcenters`	number of community health centers

Stratification can be viewed as a special case of constrained randomization. For instance, we could consider stratifying on a single binary baseline covariate, geographic location (rural or urban), for the immunization trial introduced previously. Suppose that 6 counties are located in the rural area and that 10 counties are located in the urban area. Stratified randomization ensures that half of the clusters in each stratum, defined by distinct values of the geographic location variable, are assigned to treatment and the rest to control. If we measure balance by the absolute differences in the average covariate values between arms, it follows that the stratified randomization space coincides with a constrained randomization space with zero balance scores when each stratum contains an even number of clusters.

Constrained randomization generalizes stratification and extends naturally to situations where there are several, possibly continuous, baseline covariates. The generalization is featured by defining a balance metric accommodating multiple covariates. A balance metric gives a quantitative assessment about the balance between the two arms for each randomization scheme, and essentially any sensible balance metric can be used. We first develop the `cvrall` function that balances covariates by scalar balance scores as in Raab and Butcher (2001) and Li et al. (2016, 2017). Suppose we wish to balance K baseline covariates, either cluster attributes or individual characteristics aggregated at the cluster level (dummy variables are used for categorical covariates). We denote n as the total number of clusters, n_T, n_C as the number of treated and control clusters (i.e., $n = n_T + n_C$), x_{ik} as the kth covariate ($k = 1, \ldots, K$) of cluster i. The $l2$ balance metric, first introduced by Raab and Butcher (2001), can be written as

$$B_{(l2)} = \sum_{k=1}^{K} \omega_k \left(\bar{x}_{Tk} - \bar{x}_{Ck}\right)^2 \tag{1}$$

where $\bar{x}_{Tk} = \sum_{i=1}^{n_T} x_{ik}/n_T$ and $\bar{x}_{Ck} = \sum_{i=n_T+1}^{n} x_{ik}/n_C$ are the means of the kth cluster-level variable in the treatment arm and the control arm, respectively, and ω_k is a pre-determined weight for the kth variable. We choose ω_k to be the inverse of the variance of the kth variable across all clusters following Raab and Butcher (2001) and Li et al. (2016), namely

$$\omega_k = 1/s_k^2 = \frac{n-1}{\sum_{i=1}^{n} \left(x_{ik} - \bar{x}_k\right)^2}$$

where $\bar{x}_k = \sum_{i=1}^{n} x_{ik}/n$.

An alternative $l1$ balance metric was introduced by Li et al. (2017) as

$$B_{(l1)} = \sum_{k=1}^{K} \tilde{\omega}_k \left|\bar{x}_{Tk} - \bar{x}_{Ck}\right| \tag{2}$$

where the notations are consistent with the $l2$ metric except for the weight $\tilde{\omega}_k$, which is chosen to be the inverse of the standard deviation of the kth variable, s_k. It has been shown that the two balance metrics perform similarly in constrained randomization, that both metrics are invariant to affine transformation of baseline covariates (Li et al., 2016, 2017), and that the resulting balance scores are free of the unit used to measure the baseline covariates as long as the unit of measurement is consistent across clusters. Finally, after the randomization schemes are enumerated or simulated, we simultaneously compute the balance scores for all schemes according to either the $l1$ or $l2$ metric, using the matrix formula given in Li et al. (2017). We refer the reader to Web Appendix B of Li et al. (2017) for additional computational details.

To reflect the relative importance of different covariates, one may specify different weights in the

$l1$ and $l2$ balance metric. To do so, we can modify the $l2$ balance metric to be

$$B_{(l2)} = \sum_{k=1}^{K} d_k \omega_k \left(\bar{x}_{Tk} - \bar{x}_{Ck}\right)^2 \tag{3}$$

where d_k is the user-defined weight for the kth variable. By default, $d_k = 1$ for all variables and equation (3) reduces to equation (1). When researchers consider a certain variable to be more "important" (in terms of prognostic value) than the others, a large user-defined weight $d_k > 1$ could be assigned to that variable when assessing the balance scores. Similarly, we modify the $l1$ balance metric by allowing for user-defined weights as

$$B_{(l1)} = \sum_{k=1}^{K} d_k \widetilde{\omega}_k \left|\bar{x}_{Tk} - \bar{x}_{Ck}\right| \tag{4}$$

Another important element of constrained randomization is the cutoff value, which we denote by $q \in (0, 1]$. If we write F_B as the empirical cumulative distribution function of the balance scores calculated using a balance metric, we could define the cutoff value as the percentile such that the constrained space contains schemes with balance scores no larger than $F_B^{-1}(q)$. Intuitively, the cutoff value measures the proportion of schemes relative to the simple randomization space. When $q = 1$, there is no constraint and simple randomization is implemented. When $q < 1$, only a subset of schemes with sufficient balance will be retained and constrained randomization is implemented. In the immunization trial example, we have in total $\binom{16}{8} = 12,870$ possible randomization schemes to allocate 8 clusters each to intervention and control. If we set $q = 0.1$, the constrained randomization space contains around 1288 schemes, allowing for ties in the balance scores.

Ideally, the cutoff value q should be small and away from 1 so that only the "more balanced" randomization schemes are retained in the constrained space. In fact, the power of statistical inference on the intervention effect tends to increase as q decreases if prognostic covariates are balanced by constrained randomization. However, the cutoff value q should not be too small since this may risk deterministic allocation of clusters into arms (Moulton, 2004), and may prohibit permutation inference given a fixed type I error rate (Li et al., 2016). In addition, the relationship between q and power is not monotone since power may stabilize once $q < 0.1$, as seen in a number of simulations presented in Li et al. (2017). For this reason, we set the default cutoff value of $q = 0.1$ in `cvrall`, unless specified otherwise by the user. Finally, we note that in our `cvrall` function, one could also specify the exact number of schemes kept in the constrained randomization instead of the cutoff quantile value, through the `numschemes` argument.

In addition to constraining the randomization space via a scalar summary score, we further developed the `cvrcov` function to implement constrained randomization with baseline balance defined directly through each covariate. This covariate-by-covariate constrained randomization places separate constraints on each covariate and ensures that the final allocation scheme satisfies marginal balance of each covariate. In particular, we follow the routine developed by Greene (2017) and constrain the arm mean difference (or arm total difference) to be no larger than a pre-specified value or a certain percentage of overall mean (or mean arm total). The covariate-by-covariate balance allows user-specified constraints on different covariates and is more flexible, but simulating the constrained randomization space usually requires more computations since the balance metric does not reduce to simple forms as the $l1$ or $l2$ scores.

To better understand the constrained randomization space, we also include a check on the randomization validity (Bailey and Rowley, 1987). Constraining the randomization may induce linkage or correlation between clusters so that certain pairs of clusters may always be allocated to the same arm (cluster coincidence) or never be allocated to the same arm (cluster separation), both of which lead to loss of randomization validity. To assess the degree of loss of validity, the `cvrall` and `cvrcov` functions provide summary statistics on cluster pairs that always or never appear together in the same arm, similar to the routine by Greene (2017). Such descriptive statistics may inform the appropriate selection of a constrained space.

Finally, enumerating all possible schemes in the entire simple randomization space may be computationally demanding, when there are quite a few clusters to randomize (e.g., more than 20). In that case, the `cvrall` and `cvrcov` functions in our package will randomly simulate a large number of randomization schemes and remove duplicates if any. By default, this large number is set to be 50,000, unless specified otherwise by the user through the `size` option. With this default setting, when the total number of schemes in the simple randomization space is no greater than 50,000, the enumeration method will be used. Otherwise, 50,000 schemes will be randomly simulated from the simple randomization space and duplicates will be removed to approximate the simple randomization space.

Clustered permutation test

After using constrained randomization in the design of a CRT, a permutation test can be used to test the intervention effect. We implement the clustered permutation test used in Gail et al. (1996) and Li et al. (2016) in the `cptest` function. Specifically, we denote the outcome of the jth individual ($j = 1, \ldots, m_i$) from the ith cluster ($i = 1, \ldots, n$) as Y_{ij}. During the analysis stage, researchers may wish to adjust for baseline covariates, which we denote by a vector $\mathbf{z}_{ij}$. The choice of adjustment variables may vary from study to study, and often depends on expert knowledge. Generally, it is a good practice to adjust for variables with high prognostic values that are already balanced by constrained randomization. For the permutation test, such a recommendation is not mandatory since the test size remains valid as long as the permutation distribution is obtained from the constrained randomization space (Li et al., 2016), even though adjusting for prognostic variables improves the test power (Li et al., 2016, 2017). However, if one prefers an unadjusted test, the following permutation inference still holds by setting $\mathbf{z}_{ij}$ as the null or empty vector.

The permutation test is implemented in a two-step procedure. In the first step, an outcome regression model is fitted for response Y_{ij} with covariates $\mathbf{z}_{ij}$. This is often done by fitting a linear regression model for continuous responses and a logistic regression model for binary responses, ignoring the clustering of responses. We then compute the predicted response for each individual by $\hat{Y}_{ij}$, which could be used to calculate the individual residual $r_{ij} = Y_{ij} - \hat{Y}_{ij}$. In the second step, cluster-specific residual averages are obtained as $\bar{r}_{i\cdot} = \sum_{j=1}^{m_i} r_{ij}/m_i$. The observed test statistic is then computed as

$$U = \frac{1}{n_T} \sum_{i=1}^{n} W_i \bar{r}_{i\cdot} - \frac{1}{n_C} \sum_{i=1}^{n} (1 - W_i)\, \bar{r}_{i\cdot} \tag{5}$$

where $W_i = 1$ if the ith cluster is assigned to the treatment arm and $W_i = 0$ otherwise, and $n_T = \sum_{i=1}^{n} W_i$, $n_C = \sum_{i=1}^{n} (1 - W_i)$ are the number of treated and control clusters.

Suppose there are S randomization schemes in the constrained randomization space. To obtain the permutation distribution of the test statistic, we permute the labels of the treatment indicator according to the constrained randomization space, and recompute a value for U_s ($s = 1, \ldots, S$) based on equation (5). The collection of these values $\{U_s : s = 1, \ldots, S\}$ forms the null distribution of the permutation test statistic. The p-value is then computed by

$$\text{p-value} = \frac{1}{S} \sum_{s=1}^{S} \mathbb{I}\left(|U_s| \geq |U|\right) \tag{6}$$

where $\mathbb{I}$ is the indicator function that equals 1 when $|U_s| \geq |U|$ and 0 otherwise.

Illustrative Examples

Constrained randomization by `cvrall`

We used the `cvrall` function to perform constrained randomization based on the CRT data published in Dickinson et al. (2015). To focus ideas, we selected five variables in Table 1 to balance in the design stage. These variables include `location` (categorical), `inciis` (continuous), `uptodateonimmunizations` (continuous) and `hispanic` (continuous). We further considered `incomecat` as a categorical variable to illustrate the use of `cvrall` in the presence of a factor variable. Of note, the `cvrall` function automatically converts the categorical variables into dummy variables when implementing the constrained randomization. For instance, here we categorized the county-level covariate `incomecat` into three levels based on sample tertiles: "low", "medium", and "high". Two dummy variables are then introduced to represent these three categories. The "high" level is by default considered as the reference level by alphanumerical order of the first letter. Similarly, when the permutation test is executed in the `cptest` function, each categorical covariates will be transformed into dummy variables before performing the analysis as well. It is also important to point out that there is more than one way to define dummy variables because any one of the levels of the categorical variable could be chosen as the reference level. In the `cvrall` function, if the variable is not specified as a factor with a specific reference level, we defined the reference level to be the first level by alphanumerical order. However, if one would like to specify other reference levels, it is possible to preprocess the data to manually create dummy variables before invoking the `cvrall` routines, or to specify the variables as factors with the specific reference levels.

In this trial, we would like to randomize 8 counties into the arm with a collaborative centralized reminder approach and 8 into the other arm with a practice-based approach. So we specified

`ntotal_cluster = 16` and `ntrt_cluster = 8` for the total number of clusters and the number of clusters in the treatment arm. Since the total number of possible schemes is $\binom{16}{8} = 12,870$, which is less than the default maximum number of simulated schemes (50,000), we enumerated all 12,870 schemes. The example syntax of the function is given as the following, where the `x=` argument references the data frame of the covariates that will be used in the calculation of balance scores and hence be balanced by constrained randomization.

```
Design_result <- cvrall(clustername = Dickinson_design$county,
                        balancemetric = "l2",
                        x = data.frame(Dickinson_design[ , c("location", "inciis",
                            "uptodateonimmunizations", "hispanic", "incomecat")]),
                        ntotal_cluster = 16,
                        ntrt_cluster = 8,
                        categorical = c("location", "incomecat"),
                        savedata = "dickinson_constrained.csv",
                        bhist = TRUE,
                        cutoff = 0.1,
                        seed = 12345,
                        check_validity = TRUE)
```

Here we used the balance scores calculated by the $l2$ metric as indicated by `balancemetric = "l2"`. The cateogrical variables were specified with `categorical = c("location","incomecat")`. Location has two levels: `"rural"` and code"urban"; the level `"rural"` is the reference level. As income category is a three-level categorical variable of `"low"`, `"med"`, and `"high"`, the level `"high"` is considered as the reference level and 2 dummy variables were created. Since we specified the cutoff value as `cutoff = 0.1`, the constrained randomization space only included the schemes with $l2$ balance scores less than the 10th percentile of the balance score distribution in the simple randomization space. Finally, we randomly sampled a scheme from the constrained space.

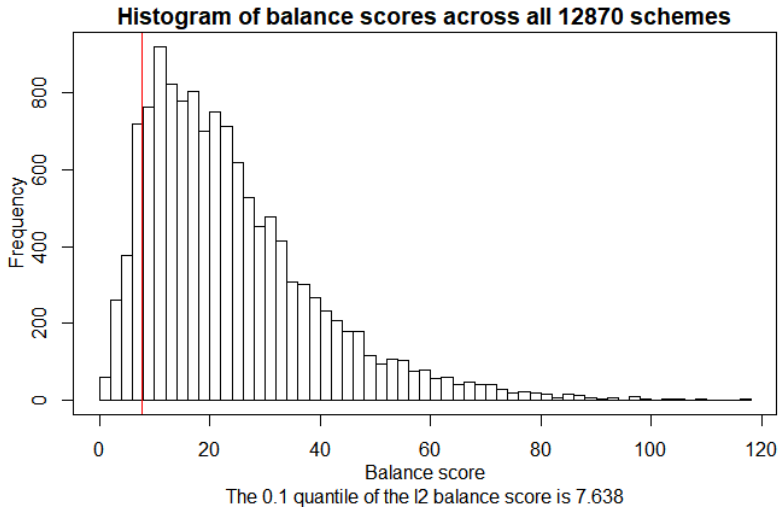


Figure 1: Histogram of balance scores across all 12870 schemes

We saved the constrained randomization space in a file named `dickinson_constrained.csv` in the current working directory. In this file, the first column is an indicator variable of whether the scheme is the final one selected by the program. The remaining columns records the constrained randomization matrix; each column of the matrix corresponds to a cluster, and each row of the matrix corresponds to an allocation scheme coded by 1's and 0's (1 if the cluster is assigned to the collaborative centralized reminder approach and 0 if assigned to the practice-based reminder approach). Furthermore, if simple randomization is used, namely `cutoff = 1`, the constrained randomization matrix has 12,870 rows and 16 columns. We provide the option to save the constrained randomization space to a local directory so that it could be used as an input for the permutation inference during the data analysis stage, which usually happens at a later calendar time.

To facilitate the understanding of the constrained randomization process, we could specify `bhist = TRUE` to generate a histogram displaying the distribution of all balance scores with a red line indicating the cutoff value (the 10th percentile). The sample histogram of balance scores is in Figure 1. The summary statistics of the balance scores are included in the `bscores` object, regardless of the `bhist =` option. As indicated below, the `bscores` object contains the cutoff value, the balance score corresponding to the selected scheme, and other quantiles of the balance score distribution.

```
> Design_result$bscores
1  score (selected scheme)   6.764
2              cutoff score   7.638
3                     Mean  24.000
4                       SD  15.775
5                      Min   1.161
6                       5%   5.826
7                      10%   7.638
8                      20%  10.849
9                      25%  12.221
10                     30%  13.840
11                     50%  20.578
12                     75%  31.621
13                     95%  55.486
14                     Max 116.656
```

In order to be transparent about the constrained randomization procedure, we also included additional summary messages in the following objects: `assignment_message`, `scheme_message`, `cutoff_message` and `choice_message`. These objects summarize the sample size and randomization ratio, the number of schemes used to calculate the balance score distribution, the balance metric and cutoff value, as well as the balance score of the selected scheme, respectively. For example, the sample size and randomization ratio are indicated in the following message:

```
> Design_result$assignment_message
[1] "You have indicated that you want to assign 8 clusters to treatment and 8 to control"
```

The final randomization scheme is included in the `allocation` object. In addition, we also provided a data frame containing the final randomization scheme in the `data_CR` element. The data frame includes the covariate values for each cluster in addition to the information on cluster allocation.

```
> Design_result$data_CR
   arm clustername location inciis uptodateonimmunizations hispanic incomecat
1    0           1    Rural     94                      37       44       Low
2    0           2    Rural     85                      39       23      High
3    0           3    Rural     85                      42       12       Low
4    1           4    Rural     93                      39       18      High
5    1           5    Rural     82                      31        6      High
6    0           6    Rural     80                      27       15       Med
7    1           7    Rural     94                      49       38       Low
8    0           8    Rural    100                      37       39       Low
9    1           9    Urban     93                      51       35       Med
10   1          10    Urban     89                      51       17       Med
11   0          11    Urban     83                      54        7      High
12   1          12    Urban     70                      29       13       Med
13   1          13    Urban     93                      50       13      High
14   0          14    Urban     85                      36       10       Med
15   1          15    Urban     82                      38       39       Low
16   0          16    Urban     84                      43       28       Med
```

To assess whether the selected constrained randomization scheme balances the baseline covariates, we provided a baseline table summarized under the selected randomization scheme. The baseline table indicates that the covariates are approximately balanced across the two arms, although more "urban" clusters are assigned to the collaborative centralized reminder approach. The baseline table is provided in the `baseline_table` element, and is illustrated below.

```
> Design_result$baseline_table
                                        arm = 0       arm = 1
n                                             8             8
location = Urban (%)                  3 (37.5)      5 (62.5)
inciis (mean (sd))                 87.00 (6.59)  87.00 (8.45)
uptodateonimmunizations (mean (sd))  39.38 (7.65)  42.25 (9.18)
hispanic (mean (sd))               22.25 (13.77) 22.38 (12.94)
incomecat (%)
   High                               2 (25.0)      3 (37.5)
   Low                                3 (37.5)      2 (25.0)
   Med                                3 (37.5)      3 (37.5)
```

Finally, we considered the validity of the randomization and used the `check_validity =` argument to summarize the cluster coincidence (cluster pairs assigned to the same arm) and cluster separation (cluster pairs assigned to different arms) within the constrained space. If `check_validity = TRUE`, we could obtain the relevant descriptive statistics in the `cluster_coin_des` object. The four rows in this object summarize the count and fraction of clusters appearing together, as well as count and fraction of clusters appearing in the different arms across the constrained randomization space. Recall that under simple randomization, no linkage or correlation is introduced between clusters and so each cluster pair has a 50% chance to appear together in the same arm and a 50% chance to appear in different arms. With the 0.1 cutoff value, the cluster pairs has a 47% chance to appear in the same arm on average, which is not too distant from the reference value 50%. However, there is a cluster pair that will appear in the same arm for only about 29% of the times (and appear in different arms for 71% of times), indicating some loss of validity. On the other hand, the constrained randomization routine offered by Greene (2017) includes default proportion values, 25% and 75%, as thresholds for loss of validity. That is to say, a reasonable constrained space should ensure each cluster pair appears in the same arm (and in different arms) for at least 25% of times and at most 75% times. Our constrained randomization space satisfies this condition.

```
> Design_result$cluster_coin_des
             Mean Std Dev Minimum 25th Pctl  Median 75th Pctl Maximum
samecount 600.600  88.807 368.000   551.750 603.000   648.500 804.000
samefrac    0.467   0.069   0.286     0.429   0.469     0.504   0.625
diffcount 686.400  88.807 483.000   638.500 684.000   735.250 919.000
difffrac    0.533   0.069   0.375     0.496   0.531     0.571   0.714
```

Stratified constrained randomization by `cvrall`

Of note, the `cvrall` function could perform constrained randomization with a stratification factor to ensure exact balance on that stratification factor. We still considered the above trial example, but now we wish to perform constrained randomization within each strata defined by the binary `location` variable. In other words, two strata of eight counties each will be defined depending on `location`, and constrained randomization is then performed based on the additional four covariates within each stratum. Motivated by the weighted $l1$ and $l2$ metrics (3), (4), we could assign a large weight (e.g., 1000) to `location` and ensure exact balance on that variable, while keeping the weights for other variables as 1 (`weights = c(1000,1,1,1,1)`). Intuitively, a large weight assigned to a covariate sharply penalizes any imbalance of that covariate, therefore the resulting randomization space approximates the one obtained by stratifying on `location`. The example syntax is provided below.

```
# Stratification on location, with constrained randomization on other
# specified covariates.
Design_stratified_result <- cvrall(clustername = Dickinson_design$county,
                                   balancemetric = "l2",
                                   x = data.frame(
                                     Dickinson_design[,
                                       c("location", "inciis",
                                         "uptodateonimmunizations", "hispanic",
                                         "incomecat")]),
                                   ntotal_cluster = 16,
                                   ntrt_cluster = 8,
                                   categorical = c("location", "incomecat"),
                                   weights = c(1000, 1, 1, 1, 1),
                                   cutoff = 0.1,
                                   seed = 12345)
```

Depending on the choice of `cutoff` value, the above syntax may not lead to a randomization space exactly the same as the one obtained after stratifying on `location`. The `cvrall` function also allows one to directly stratify on the `location` variable using the `stratify` option, as shown next. We omitted the baseline covariate table obtained from stratified constrained randomization, but just comment that final scheme ensures exact balance on the `location` variable so that each arm has now 4 urban counties and 4 rural counties.

```
# An alternative and equivalent way to stratify on location
Design_stratified_result <- cvrall(clustername = Dickinson_design$county,
                                   balancemetric = "l2",
                                   x = data.frame(
                                     Dickinson_design[ ,
```

```
    c("location", "inciis",
      "uptodateonimmunizations", "hispanic",
      "incomecat")]),
ntotal_cluster = 16,
ntrt_cluster = 8,
categorical = c("location", "incomecat"),
stratify = "location",
cutoff = 0.1,
seed = 12345)
```

Constrained randomization by `cvrcov`

We additionally provided the `cvrcov` function to perform covariate-by-covariate constrained randomization, similar to the routine provided by Greene (2017). This approach is particularly attractive for its flexibility in directly balancing each covariate. We still considered our example trial where we randomized 8 counties into the each arm for illustration. We specified `ntotal_cluster = 16` and `ntrt_cluster = 8` for the total number of clusters and the number of clusters in the treatment arm. Since the total number of possible schemes is $\binom{16}{8} = 12,870$, which is less than the default maximum number of simulated schemes (`50,000`), we enumerated all 12870 schemes.

As the covariate-by-covariate constrained randomization acts on the numeric values of each variable, we transformed the values of the `location` to be numeric with `"Rural"` being `1` and `"Urban"` being `0`. For illustrative purposes, we also used the numeric average income values rather than its categories in this example. The `x =` argument points to the data frame containing the covariates that will be balanced by constrained randomization routine.

Table 2: Example syntax of balancing constraints.

Syntax	Explanation
`any`	no constraints, any arm means or arm totals are acceptable
`s5`	arm totals must differ in absolute value by no more than 5
`sf.5`	arm totals must differ in absolute value by no more than 0.5 times the mean arm total
`m10`	arm means must differ in absolute value by no more than 10
`mf0.2`	arm means must differ in absolute value by no more than 0.2 times the overall mean
`mf.5`	arm means must differ in absolute value by no more than 0.5 times the overall mean

The `cvrcov` function works the same way as the `cvrall` function, except for that the former requires additional syntax to specify the balancing constraints for each covariate. The syntax used to balance each covariate is the same those used in Greene (2017). Specifically, if the first letter is specified as `m`, the balancing constraint acts on means, whereas if the first letter is `s`, the balancing constraint acts on sums or totals. If the second letter is `f`, the balancing constraint will be compared to a fractional of a population quantity (overall mean or mean arm total), otherwise the constraint will be compared to an actual value. A numeric constraint will follow the specified letters and indicates the tightness of the constraint. Additional examples are provided in Table 2.

```
Dickinson_design_numeric <- Dickinson_design
Dickinson_design_numeric$location = (Dickinson_design$location == "Rural") * 1

Design_cov_result <- cvrcov(clustername = Dickinson_design_numeric$county,
                            x = data.frame(Dickinson_design_numeric[ , c("location", "inciis",
                                                                         "uptodateonimmunizations",
                                                                         "hispanic", "income")]),
                            ntotal_cluster = 16,
                            ntrt_cluster = 8,
                            constraints = c("s5", "mf.5", "any", "mf0.2", "mf0.2"),
                            categorical = c("location"),
                            savedata = "dickinson_cov_constrained.csv",
                            seed = 12345,
                            check_validity = TRUE)
```

We specified `constraints = c("s5","mf.5","any","mf0.2","mf0.2")` for the five covariates respectively. As indicated above, `s5` indicates that the allocation scheme should ensure that the arm totals differ in absolute value by no more than 5. Synaxt `mf.5` indicates that the allocation scheme should ensure that the arm means differ by no more than 0.5 times the overall mean for `inciis`, among others. We saved the resulting constrained randomization space as `dickinson_cov_constrained.csv`.

Similar to `cvrall`, the `cvrcov` routine included additional summary messages in the following objects: `assignment_message` and `scheme_message`. These two objects summarize the sample size and randomization ratio, the number of schemes enumerated or simulated before applying the constraints. In addition, a data frame containing the selected final allocation scheme is saved in the `data_CR` element as follows.

```
> Design_cov_result$data_CR
   arm id location inciis uptodateonimmunizations hispanic income
1    0  1        1     94                      37       44  35988
2    1  2        1     85                      39       23  67565
3    0  3        1     85                      42       12  35879
4    0  4        1     93                      39       18  63617
5    1  5        1     82                      31        6  59118
6    0  6        1     80                      27       15  57179
7    1  7        1     94                      49       38  29738
8    1  8        1    100                      37       39  37350
9    1  9        0     93                      51       35  52923
10   0 10        0     89                      51       17  58302
11   0 11        0     83                      54        7  93819
12   0 12        0     70                      29       13  54839
13   1 13        0     93                      50       13  63857
14   1 14        0     85                      36       10  53502
15   0 15        0     82                      38       39  39570
16   1 16        0     84                      43       28  52457
```

To evaluate whether the selected constrained randomization scheme balances the baseline covariates, we provided a baseline table summarized under the that selected randomization scheme. The baseline table indicates that the covariates are well balanced across the two arms, with an equal number of “urban" clusters assigned to each reminder approach.

```
> Design_cov_result$baseline_table
                                              arm = 0             arm = 1
n                                                   8                   8
location = 1 (%)                              4 (50.0)            4 (50.0)
inciis (mean (sd))                          84.50 (7.76)        89.50 (6.35)
uptodateonimmunizations (mean (sd))         39.62 (9.44)        42.00 (7.43)
hispanic (mean (sd))                       20.62 (13.38)       24.00 (13.09)
income (mean (sd))                  54899.12 (19130.82) 52063.75 (12800.82)
```

The `cvrcov` function permits the check of randomization validity (Bailey and Rowley, 1987), and summarizes the cluster coincidence and separation statistics in the `cluster_coin_des` object. The result indicates that all cluster pairs appear together in the same arm at least 37% and at most 55% of the times across the constrained randomization space. Using the 25% and 75% threshold, the summary statistics indicate that the constrained randomization does not severely depart from validity. Finally, the `cvrcov` function summarizes the information of the constrained space in the `overall_allocations` and `overall_summary` objects, which are suppressed here due to limited space. In short, the summary information informs that the there are in total 12,870 allocations and 5,776 ($\approx$ 45%) satisfied the balancing constraints.

```
> Design_cov_result$cluster_coin_des
              Mean Std Dev  Minimum 25th Pctl   Median 75th Pctl  Maximum
samecount 2695.467 197.148 2138.000  2567.000 2720.000  2824.500 3182.000
samefrac     0.467   0.034    0.370     0.444    0.471     0.489    0.551
diffcount 3080.533 197.148 2594.000  2951.500 3056.000  3209.000 3638.000
difffrac     0.533   0.034    0.449     0.511    0.529     0.556    0.630
```

Clustered permutation test by `cptest`

Since the immunization study is an ongoing trial, we used simulated outcome data to demonstrate the clustered permutation test with the above example where constrained randomization was performed using `cvrall` based on the 5 covariates (the selected scheme had a balance score of 6.764). The same syntax applies to the constrained randomization results obtained from `cvrcov` and so is not considered further here. Suppose that the researchers were able to assess 300 children in each county, and the trial is randomized according to the selected final scheme. For illustration, we chose the covariates to be adjusted in the test $\mathbf{z}_{ij}$ as the list of covariates x_i balanced by design. This step is in line with the

recommendation of Li et al. (2017) that adjusting for prognostic factors in the analysis improves the test power.

To generate the correlated binary outcome of whether the children is eventually up-to-date on immunizations (`1`) or not (`0`), we used a generalized linear mixed model (GLMM) with a logistic link to induce correlation by including a random intercept at the county level. The intraclass correlation coefficient (ICC) is usually used to quantify the degree of association between individual outcomes in a cluster (county). We used the latent response definition of binary ICC defined by variance components in the GLMM (Eldridge et al., 2009). The ICC was set to be 0.01, which is a reasonable value for population health studies (Hannan et al., 1994). The outcome variable depends on the county-level covariates used in performing the constrained randomization, as previously mentioned, and we simulated a treatment effect so that the collaborative reminder approach increases up-to-date immunization rates compared to the practice-based reminder approach (odds ratio equals to $e^{0.5} \approx 1.649$). The binary outcome for each individual child is generated from a Bernoulli model with event probability specified by the GLMM.

We performed the clustered permutation test using the `cptest` function for the binary outcome of the status of up-to-date on immunizations. As indicated in Li et al. (2016), valid permutation test under constrained randomization should only shuffle the treatment label within the constrained space, and so it is important to save and input the constrained randomization space in the design stage (the file named `dickinson_constrained.csv`). The permutation test is performed by first regressing the outcome on the five covariates, `inciis`, `uptodateonimmunizations`, `hispanic`, `location`, and `incomecat`. As the last two covariates are categorical, the `cptest()` function creates dummy variables and set reference levels according to alphanumerical order, matching the steps in `cvrall`. Of note, had different reference levels been selected for the constrained randomization design procedure, the corresponding dummy coding should be reflected in the analysis phase when the clustered permutation test is used. We specified `outcometype` to be "binary" so that logistic regression is performed to compute the residuals. An example syntax of the function is given as follows.

```
Analysis_result <- cptest(outcome = Dickinson_outcome$outcome,
                          clustername = Dickinson_outcome$county,
                          z = data.frame(Dickinson_outcome[, c("location", "inciis",
                              "uptodateonimmunizations", "hispanic", "incomecat")]),
                          cspacedatname = "dickinson_constrained.csv",
                          outcometype = "binary",
                          categorical = c("location","incomecat"))
```

The covariates to be adjusted for in the permutation test is indicated in the `z =` option, which matches the covariate matrix used in `cvrall` for constrained randomization. If one wishes to an unadjusted permutation test, one could leave out the `z =` option as it is an optional argument. The output of `Analysis_result` includes the final scheme selected by design (`FinalScheme` object), the p-value of the test (`p-value` object) and a sentence to describe the p-value (`pvalue_statement` object). We omitted the code output here for brevity, but comment that, in this example, the p-value equals to `0.042`, indicating that there is a significant difference in the effect of the interventions on the outcome of up-to-date on immunizations, if testing is performed at the 5% significance level. Again, if the constrained randomization is performed by `cvrcov`, we could use the `cptest` function in a similar way once we provided the constrained permutation matrix obtained from `cvrcov` in the `cspacedatname =` argument.

Summary

The **cvcrand** package contains three main functions for the design and analysis of cluster randomized trials. Given that it is common for such trials to enroll a small number of clusters and that this gives rise to chance imbalance in covariates that are predictive of the outcome, the `cvrall` and `cvrcov` functions can be used to implement covariate-constrained randomization in the design phase to ensure better balance. The `cvrall` function uses a balance metric to quantify balance across multiple cluster-level covariates, whereas the `cvrcov` allows for covariate-by-covariate balance and could potentially be more flexible. For analysis of the individual-level outcome data collected in the CRT, the `cptest` function could help perform the clustered permutation test, which accommodate both continuous and binary outcomes and should be treated as a flexible alternative to model-based analysis.

There are several limitations of the **cvcrand** package. First, the `cvrall` and `cvrcov` only deal with two-arm parallel cluster randomized trials and may not be directly applied to balance covariates in other designs such as the stepped wedge designs (Hussey and Hughes, 2007; Li et al., 2018). Second, although the `cptest` function performs a valid analysis for individual-level outcome data when there is an equal number of clusters per arm, the test may be anti-conservative when there is an

unequal number of clusters per arm (Gail et al., 1996). Furthermore, our `cptest` routine does not provide a confidence interval for the intervention effect estimate, and additional programming is required to obtain a permutation confidence interval. Essentially, the permutation test will be inverted to numerically search for the interval limits, as is done in Gail et al. (1996) for an unadjusted test under simple randomization. For the adjusted test under constrained randomization, the following steps could be carried out: (i) hypothesize an treatment effect δ on the link function scale; (ii) obtain the residuals $r_{ij} = Y_{ij} - \hat{Y}_{ij}$, where $\hat{Y}_{ij}$ is estimated from regressing Y_{ij} on $\mathbf{z}_{ij}$ and the hypothesized treatment effect; (iii) perform the permutation test under the constrained randomization space and obtain a p-value; (iv) repeat steps (i)-(iii) for different values of δ and the confidence interval is the collection of δ such that the p-value is at least 0.05. We noticed that few studies were present in the CRT literature to evaluate the performance of permutation intervals that adjust for covariates under constrained randomization, and this is an avenue for future research. On the other hand, it is also important to notice that point and interval estimates could be easily obtained from model-based approaches, with caveats discussed in Li et al. (2016). In the class of model-based approaches, the most commonly-used approaches are the generalized linear mixed model (GLMM) model approach, which estimates the cluster-specific conditional effect, and the generalized estimating equations (GEE) approach, which estimates the population-averaged or marginal effect (Turner et al., 2017b). In each case, it has been demonstrated that the model-based analyses should account for the prognostic covariates used in the design (Li et al., 2016, 2017). Finally, although the `cptest` function can handle both continuous and binary outcomes, we have not yet extended the function to accommodate count outcomes. In summary, these limitations reflect the current research on constrained randomization. We plan to update the **cvcrand** package as the theory and knowledge of these procedures develop in the future.

Acknowledgements

The authors would like to thank Alyssa Platt, Joe Egger, and Ryan Simmons of the Duke Global Health Institute Research Design and Analysis Core for testing and providing feedback on the programs. This research was funded in part by National Institutes of Health grant R01 HD075875 (PI: Dr. Joanna Maselko).

Bibliography

R. Bailey and C. Rowley. Valid randomization. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 410(1838):105–124, 1987. URL https://doi.org/10.1098/rspa.1987.0030. [p194, 200]

A. W. Brown and P. Li. Best (but oft-forgotten) practices: Designing, analyzing, and reporting cluster randomized controlled trials. *The American Journal of Clinical Nutrition*, 102(2):241–248, 2015. URL https://doi.org/10.3945/ajcn.114.105072. [p191]

M. K. Campbell, G. Piaggio, D. R. Elbourne, and D. G. Altman. Consort 2010 statement: Extension to cluster randomised trials. *Bmj*, 345:e5661, 2012. URL https://doi.org/10.1136/bmj.e5661. [p191]

L. M. Dickinson, B. Beaty, C. Fox, W. Pace, W. P. Dickinson, C. Emsermann, and A. Kempe. Pragmatic cluster randomized trials using covariate constrained randomization: A method for practice-based research networks (pbrns). *The Journal of the American Board of Family Medicine*, 28(5):663–672, 2015. URL https://doi.org/10.3122/jabfm.2015.05.150001. [p192, 195]

P. Diehr, D. C. Martin, T. Koepsell, and A. Cheadle. Breaking the matches in a paired t-test for community interventions when the number of pairs is small. *Statistics in medicine*, 14(13):1491–1504, 1995. URL https://doi.org/10.1002/sim.4780141309. [p191]

A. Donner and N. Klar. Pitfalls of and controversies in cluster randomization trials. *American Journal of Public Health*, 94(3):416–422, 2004. URL https://doi.org/10.2105/AJPH.94.3.416. [p191]

S. M. Eldridge, O. C. Ukoumunne, and J. B. Carlin. The intra-cluster correlation coefficient in cluster randomized trials: A review of definitions. *International Statistical Review*, 77(3):378–394, 2009. URL https://doi.org/10.1111/j.1751-5823.2009.00092.x. [p201]

M. Fiero, S. Huang, and M. L. Bell. Statistical analysis and handling of missing data in cluster randomised trials: Protocol for a systematic review. *BMJ open*, 5(5):e007378, 2015. URL https://doi.org/10.1186/s13063-016-1201-z. [p191]

M. H. Gail, S. D. Mark, R. J. Carroll, S. B. Green, and D. Pee. On design considerations and randomization-based inference for community intervention trials. *Statistics in medicine*, 15(11): 1069–1092, 1996. URL https://doi.org/10.1002/(SICI)1097-0258(19960615)15:11<1069::AID-SIM220>3.0.CO;2-Q. [p192, 195, 202]

E. J. Greene. A sas macro for covariate-constrained randomization of general cluster-randomized and unstratified designs. *Journal of statistical software*, 77(CS1), 2017. URL https://doi.org/10.18637/jss.v077.c01. [p192, 194, 198, 199]

P. J. Hannan, D. M. Murray, D. R. Jacobs Jr, and P. G. McGovern. Parameters to aid in the design and analysis of community trials: Intraclass correlations from the minnesota heart health program. *Epidemiology*, pages 88–95, 1994. URL https://doi.org/10.1097/00001648-199401000-00013. [p201]

R. J. Hayes and L. H. Moulton. *Cluster Randomizd Trials*, chapter 1-3, pages 3–40. CRC Press, 2009. URL https://www.crcpress.com/Cluster-Randomised-Trials-Second-Edition/Hayes-Moulton/p/book/9781498728225. ISBN 9781584888178. [p191, 192]

M. A. Hussey and J. P. Hughes. Design and Analysis of Stepped Wedge Cluster Randomized Trials. *Contemporary Clinical Trials*, 28(2):182–191, 2007. URL https://doi.org/10.1016/j.cct.2006.05.007. [p201]

N. M. Ivers, I. J. Halperin, J. Barnsley, J. M. Grimshaw, B. R. Shah, K. Tu, R. Upshur, and M. Zwarenstein. Allocation techniques for balance at baseline in cluster randomized trials: a methodological review. *Trials*, 13(1):120, 2012. URL https://doi.org/10.1186/1745-6215-13-120. [p191, 192]

N. Klar and A. Donner. The merits of matching in community intervention trials: a cautionary tale. *Statistics in medicine*, 16(15):1753–1764, 1997. URL https://doi.org/10.1002/(SICI)1097-0258(19970815)16:15<1753::AID-SIM597>3.0.CO;2-E. [p191]

F. Li, Y. Lokhnygina, D. M. Murray, P. J. Heagerty, and E. R. DeLong. An evaluation of constrained randomization for the design and analysis of group-randomized trials. *Statistics in medicine*, 35(10): 1565–1579, 2016. URL https://doi.org/10.1002/sim.6813. [p192, 193, 194, 195, 201, 202]

F. Li, E. L. Turner, P. J. Heagerty, D. M. Murray, W. M. Vollmer, and E. R. DeLong. An evaluation of constrained randomization for the design and analysis of group-randomized trials with binary outcomes. *Statistics in medicine*, 36(24):3791–3806, 2017. URL https://doi.org/10.1002/sim.7410. [p192, 193, 194, 195, 201, 202]

F. Li, E. L. Turner, and J. S. Preisser. Sample Size Determination for GEE Analyses of Stepped Wedge Cluster Randomized Trials. *Biometrics*, 74(4):1450–1458, 2018. URL https://doi.org/10.1111/biom.12918. [p201]

L. H. Moulton. Covariate-based constrained randomization of group-randomized trials. *Clinical Trials*, 1(3):297–305, 2004. URL https://doi.org/10.1191/1740774504cn024oa. [p191, 194]

G. M. Raab and I. Butcher. Balance in cluster randomized trials. *Statistics in medicine*, 20(3):351–365, 2001. URL https://doi.org/10.1002/1097-0258(20010215)20:3<351::AID-SIM797>3.0.CO;2-C. [p192, 193]

E. L. Turner, F. Li, J. A. Gallis, M. Prague, and D. M. Murray. Review of recent methodological developments in group-randomized trials: Part 1—design. *American Journal of Public Health*, 107(6): 907–915, 2017a. URL https://doi.org/10.2105/AJPH.2017.303706. [p191]

E. L. Turner, M. Prague, J. A. Gallis, F. Li, and D. M. Murray. Review of recent methodological developments in group-randomized trials: Part 2—analysis. *American journal of public health*, 107(7): 1078–1086, 2017b. URL https://doi.org/10.2105/AJPH.2017.303707. [p202]

Hengshi Yu
Department of Biostatistics, School of Public Health, University of Michigan
Ann Arbor, Michigan 48109
USA
(ORCiD: 0000-0001-9850-9347)
hengshi@umich.edu

Fan Li
Department of Biosatistics, Yale School of Public Health, Yale University
New Haven, Connecticut 06510

USA
(ORCiD: 0000-0001-6183-1893)
fan.f.li@yale.edu

John A. Gallis
Department of Biosatistics and Bioinformatics, Duke University
Duke Global Health Institute, Duke University
Durham, North Carolina 27710
USA
(ORCiD: 0000-0003-1921-8424)
john.gallis@duke.edu

Elizabeth L. Turner
Department of Biosatistics and Bioinformatics, Duke University
Duke Global Health Institute, Duke University
Durham, North Carolina 27710
USA
(ORCiD: 0000-0002-7638-5942)
liz.turner@duke.edu

jomo: A Flexible Package for Two-level Joint Modelling Multiple Imputation

by Matteo Quartagno, Simon Grund and James Carpenter

Abstract Multiple imputation is a tool for parameter estimation and inference with partially observed data, which is used increasingly widely in medical and social research. When the data to be imputed are correlated or have a multilevel structure — repeated observations on patients, school children nested in classes within schools within educational districts — the imputation model needs to include this structure. Here we introduce our **jo**int **mo**delling package for multiple imputation of multilevel data, **jomo**, which uses a multivariate normal model fitted by Markov Chain Monte Carlo (MCMC). Compared to previous packages for multilevel imputation, e.g. **pan**, **jomo** adds the facility to (i) handle and impute categorical variables using a latent normal structure, (ii) impute level-2 variables, and (iii) allow for cluster-specific covariance matrices, including the option to give them an inverse-Wishart distribution at level 2. The package uses C routines to speed up the computations and has been extensively validated in simulation studies both by ourselves and others.

Introduction

Missing data are ubiquitous in clinical and social research. The most straightforward way to deal with missing data is to exclude all observations with any item missing from the analysis, i.e. a *complete records analysis*. However this strategy is at best inefficient; further unless — given covariates — the probability of a complete record does not depend on the outcome (dependent) variable, it will lead to biased results.

Rubin (1976) described different mechanisms causing missing data: Missing Completely At Random (probability of missingness unrelated to observed and unobserved values, MCAR), Missing At Random (given observed data, occurrence of missing values is independent of the actual values, MAR) and Missing Not At Random (given observed data, occurrence of missing values still depends on the actual values, MNAR).

Multiple Imputation (MI) is a very flexible, practical, tool to deal with missing data. It consists of imputing missing data several times, creating multiple imputed data sets. Then, the substantive model is directly fitted to each of the imputed data sets; the results are then combined for inference using Rubin's rules (Rubin, 1987). An appropriately specified multiple imputation model gives valid, efficient inference if data are MAR (Carpenter and Kenward, 2013, Ch. 2; Little and Rubin, 2002). Key attractions of MI are that it separates the imputation of missing data from the analysis, thus allowing (a) use of the substantive analysis model that we intended to use with fully observed data and (b) inclusion of auxiliary variables in the imputation model — which provide information about the missing values — without having to include them in the substantive analysis model.

There are several models and associated algorithms which can be used to impute missing data; Schafer (1997) proposed a joint multivariate normal model, fitted by MCMC. The main assumption of this method is that the partially observed data follow a joint multivariate normal distribution; given this, a Gibbs sampler uses the proper conditional distributions to update the parameters of the model and impute the missing data.

One of the advantages of the joint modelling approach is that it extends naturally to multilevel/hierarchical data structures. Such structures arise commonly, for example, when we have repeated observations (level 1) on individuals (level 2), or students (level 1) nested in schools (level 2).

A number of joint modelling multiple imputation packages have been written: **norm** (Novo and Schafer, 2013; Schafer and Olsen, 2000) assumes a multivariate normal model for imputation of single-level normal data, **cat** (Harding et al., 2012) a log-linear model to impute categorical data, and **mix** (Schafer., 2010) uses the general location model (Olkin and Tate, 1961) to impute a mix of continuous and categorical data (Schafer, 1997, Ch. 9). **pan** (Zhao and Schafer, 2013; Schafer and Yucel, 2002) uses a multilevel multivariate normal model for imputation of multilevel normal data.

As far as we are aware, **jomo** is the first R package to extend this to allow for a mix of multilevel (clustered) continuous and categorical data. It is derived from, but also extends the functionality of REALCOM (Carpenter et al., 2011), a standalone program written in Matlab. In REALCOM, binary and categorical variables are handled through a latent normal variables approach presented in Carpenter and Kenward (2013, Ch. 5). The aim of the **jomo** package is to provide an efficient implementation of the REALCOM imputation model in R, while both (i) speeding up the processes through the use of C sub-routines and (ii) adding the flexibility to specify level-2 specific covariance matrices and level-2 random covariance matrices, the latter following the proposal of Yucel (2011).

This paper is organized as follows: after briefly introducing joint modelling multiple imputation and the principal functions within **jomo**, we present a series of short tutorials, in which we explain how to impute (i) single-level continuous and categorical data; (ii) clustered homoscedastic data, (iii) clustered heteroscedastic data and (iv) level-2 variables. We then present functions to help check the convergence of the underlying MCMC algorithms and outline the suggested workflow for using the package. The next section introduces another R package, **mitml**, which provides both a different interface to **jomo** and a number of useful tools to manage and investigate the properties of imputed data sets. The penultimate section provides an overview of both the simulation studies and the applications where the package was used. We conclude with a short discussion.

Joint Modelling Multiple Imputation

To introduce the general ideas of joint modelling multiple imputation, we consider a data set made up of N observations on three continuous variables; we further assume that one of these variables, X, is fully observed, while the other two, Y_1 and Y_2, have missing values. The main idea behind joint modelling imputation is to define a joint multivariate model for all the variables in the data set, to be used for imputation. The simplest joint model is the multivariate normal model:

$$\begin{aligned} Y_{1,i} &= \beta_{0,1} + \beta_{1,1} X_i + \epsilon_{1,i} \\ Y_{2,i} &= \beta_{0,2} + \beta_{1,2} X_i + \epsilon_{2,i} \\ \begin{pmatrix} \epsilon_{1,i} \\ \epsilon_{2,i} \end{pmatrix} &\sim \mathcal{N}(\mathbf{0}, \mathbf{\Omega}) \end{aligned} \tag{1}$$

This model uses X, which is completely observed, as predictor while Y_1 and Y_2 are included as outcomes, being partially observed. Another possibility is a tri-variate normal model with all variables, including X, treated as outcomes. Choosing between these two models has been the matter of debate in the literature, and is not the focus of this paper. We simply note here that the imputation model chosen needs to be (at least approximately) congenial with the analysis model of substantial interest, in order for multiple imputation to lead to correct inference (Meng, 1994).

After having chosen the imputation model, Bayesian methods are used to fit it and impute the missing data. In particular, Gibbs sampling is used, dealing with missing data via a data augmentation algorithm (Tanner and Wong, 1987). This consists of repeatedly drawing new values for all the parameters in the model, one at a time, from the relevant conditional distribution. The parameters of model (1) are the fixed effects $\boldsymbol{\beta}$ and the covariance matrix $\mathbf{\Omega}$, and the missing data.

The sampler has to be run until it reaches the stationary distribution. Then, the current draw of the missing values is combined with the observed data to make the first imputed data set. Further imputed data sets are obtained by running the sampler for a sufficient number of supplementary iterations to guarantee stochastic independence between consecutive imputations.

Before running the Gibbs sampler, it is necessary to choose starting values for the parameters; the more plausible these starting values are, the faster the sampler converges. Furthermore, being a Bayesian method, prior distributions are used. By default, **jomo** uses flat priors for all the parameters in the model, except for the covariance matrices, which are given inverse-Wishart priors with the minimum possible number of degrees of freedom in order to give the greatest possible weight to the observed data.

Extending the same methodology to more complicated situations is conceptually relatively straightforward. For example, binary and categorical variables can be included in the imputation model by means of latent normal variables. Under this model, if Y_1 is binary, a latent continuous variable Y_1^* is included in the model, such that $Y_{1,i}^* > 0$ for individuals i for whom $Y_{1,i} = 0$ and $Y_{1,i}^* \leq 0$ for individuals for whom $Y_{1,i} = 1$ (Goldstein et al., 2009). Similarly, a categorical variable with K categories can be represented by $K - 1$ latent normal variables denoting the differences between categories. In contrast to models without categorical data, this strategy requires constraints on the variance-covariance matrix to guarantee identifiability of the model (for a wider discussion of model identifiability see Carpenter and Kenward (2013, Chap.5)). To sample from this constrained covariance matrix, Metropolis Hastings (MH) steps are introduced to augment the standard Gibbs Sampler.

If observations in the data set are nested in J clusters, model (1), can be readily expanded to include random intercepts as follows:

		Type of variables		
Model type	Missing data	Continuous	Binary/categorical	Mixed
Single-level	Level 1	`jomo1con`	`jomo1cat`	`jomo1mix`
Two-level	Level 1	`jomo1rancon`	`jomo1rancat`	`jomo1ranmix`
	Both levels	`jomo2com`	`jomo2com`	`jomo2com`
Two-level, heteroscedastic	Level 1	`jomo1ranconhr`	`jomo1rancathr`	`jomo1ranmixhr`
	Both levels	`jomo2hr`	`jomo2hr`	`jomo2hr`

Table 1: Summary of subfunctions used by the main "umbrella function" `jomo`, given the type of imputation model, the level at which missing data occur, and the type of the variables with missing data.

$$
\begin{aligned}
Y_{1,i,j} &= \beta_{0,1} + u_{1,j} + \beta_{1,1} X_{i,j} + \epsilon_{1,i,j} \\
Y_{2,i,j} &= \beta_{0,2} + u_{2,j} + \beta_{1,2} X_{i,j} + \epsilon_{2,i,j} \\
\begin{pmatrix} \epsilon_{1,i,j} \\ \epsilon_{2,i,j} \end{pmatrix} &\sim \mathcal{N}(\mathbf{0}, \mathbf{\Omega}) \\
\begin{pmatrix} u_{1,j} \\ u_{2,j} \end{pmatrix} &\sim \mathcal{N}(\mathbf{0}, \mathbf{\Omega}_u)
\end{aligned}
\tag{2}
$$

More general random effects structures can be modelled as well, as we will show in a later section.

It is similarly possible to consider a joint imputation model for variables at level 2, e.g. patient-level variables in longitudinal studies (Carpenter and Kenward, 2013, p. 212), or allowing for level-1 heteroscedasticity, which can be particularly appealing in individual patient data (IPD) meta-analyses among other applications.

Package structure

The **jomo** package can be used to impute missing data in continuous and categorical variables in single-level and multilevel data. The main interface to the **jomo** package is the `jomo` function, which automatically selects the correct imputation method for the data, depending on (a) the model specification (e.g., single-level vs. multilevel) and (b) the variables in the data set (e.g., continuous variables of type `numeric` vs. categorical variables as `factor` vs. a mixture of them). In addition, the sub-functions called by `jomo` can be called individually; a list of all sub-functions is given below and in Table 1.

1. `jomo1con`, `jomo1cat` and `jomo1mix`: these impute single-level continuous, categorical and mixed data sets. `jomo1con` is very similar to the `imp.norm` function of the **norm** package. However, `jomo1cat` and `jomo1mix` used the latent normal variables approach described in Carpenter and Kenward (2013, Ch. 5) to impute categorical variables and a mix of continuous and categorical variables, respectively.
2. `jomo1rancon`, `jomo1rancat` and `jomo1ranmix`: these impute clustered continuous, categorical, and mixed data, respectively. `jomo1rancon` is very similar to **pan**, whereas `jomo1rancat` and `jomo1ranmix` use the latent normal model for the categorical variables. All these functions have a fixed, common covariance matrix across all the clusters (level-2 units) in the imputation model;
3. `jomo1ranconhr`, `jomo1rancathr` and `jomo1ranmixhr`: these functions extend the above to allow for either cluster (level-2 unit) specific covariance matrices, or random covariance matrices, where the covariance matrices follow an inverse-Wishart distribution across level-2 units, as described by Yucel (2011) and Quartagno and Carpenter (2016).
4. `jomo2com` and `jomo2hr`: these functions impute missing values in level-2 variables, and can be used in the same manner as those in groups (2) and (3) above.

We next illustrate the use of `jomo` in each of the above situations. Throughout, we assume that the data are MAR.

jomo: tutorial with single-level data

We begin with single-level data sets. We first assume all variables with missing data are continuous, that the substantive model is a linear regression of one variable on some or all the others, and that each variable is approximately normally distributed conditional on the others (so that a joint multivariate normal model is an appropriate choice for imputation).

A key attraction of multiple imputation is that any variables in the data set that are not in the substantive model can still be included in the imputation model. If such *auxiliary variables* are good predictors of missing values, they will help recover missing information. If they are also predictors of the data being missing, they may correct bias. Recall that (simply speaking) the MAR assumption states that, given the observed data, the probability that a value is missing is conditionally independent of the actual value. Therefore, inclusion of judiciously chosen auxiliary variables (even ones that are not themselves complete) can improve the plausibility of the MAR assumption.

In joint modelling imputation (Carpenter and Kenward, 2013, Ch. 3), partially observed variables are dependent variables. However, as hinted above, with fully observed variables we can choose to either condition on them as predictors or include them in the (multivariate) response. The software is equally comfortable with both options, and it makes little difference in practice for single-level data. However, the choice has a bigger impact for clustered data (Quartagno and Carpenter, 2016; Grund et al., 2016b), as we will see in the multilevel imputation section.

Once we have decided on the variables to include in the imputation model, and whether to condition on any fully observed variables as covariates, imputation using the `jomo` function is straightforward.

For illustration, we use an educational data set of students' test scores (`JSPmiss`), which is a subset of the Junior School Project (Mortimore et al., 1988). The fully observed data set is freely available with the MLwiN software (Rasbash et al. (2017); or the related R package **R2MLwiN** Zhang et al. (2016)) and a partially observed version is included with this package (data are MAR). This data set has eight variables: a `school` and an individual identifier, `sex`, fluency in English language (3 categories, `fluent`), a test score at Year 1 (`ravens`) and at Year 3 (`english`) and a binary behavioral score at Year 3 (`behaviour`). Additionally a constant is provided to help the user, as we will see later in this tutorial.

First, we summarize the data:

```
library(jomo)
```

```
> summary(JSPmiss)
     school          id            sex          fluent        ravens         english
 48     : 76   280    :   1   Min.   :0.0000   0   : 32   Min.   : 4.00   Min.   : 0.00
 42     : 52   281    :   1   1st Qu.:0.0000   1   : 29   1st Qu.:22.00   1st Qu.:24.00
 31     : 44   282    :   1   Median :1.0000   2   :823   Median :26.00   Median :40.00
 8      : 43   283    :   1   Mean   :0.5103   NA's:235   Mean   :25.35   Mean   :41.36
 33     : 43   284    :   1   3rd Qu.:1.0000              3rd Qu.:30.00   3rd Qu.:56.00
 5      : 39   285    :   1   Max.   :1.0000              Max.   :36.00   Max.   :98.00
 (Other):822   (Other):1113                               NA's   :246     NA's   :236
       behaviour        cons
 lowerquarter:248   Min.   :1
 upper       :871   1st Qu.:1
                    Median :1
                    Mean   :1
                    3rd Qu.:1
                    Max.   :1
```

This shows the data set has $(248 + 871) = 1119$ observations, of which 236 have missing data on the outcome `english`, 235 have missing data on `fluent` and so on and so forth. For the purpose of this example, we ignore clustering by school and take as the substantive analysis model the following linear regression:

$$\begin{aligned} Y_{english,i} &= \alpha_0 + \alpha_1 \text{ravens}_i + \alpha_2 \text{sex}_i + \epsilon_i \\ \epsilon_i &\sim N(0, \sigma_e^2). \end{aligned} \tag{3}$$

To illustrate the use of `jomo` with continuous variables, let $i = 1, \ldots, 1119$ index observations and use the following joint imputation model:

$$\begin{aligned} Y_{english,i} &= \beta_{0,e} + \beta_{1,e} X_{sex,i} + \epsilon_{e,i} \\ Y_{ravens,i} &= \beta_{0,r} + \beta_{1,r} X_{sex,i} + \epsilon_{r,i} \\ \begin{pmatrix} \epsilon_{e,i} \\ \epsilon_{r,i} \end{pmatrix} &\sim \mathcal{N}(\mathbf{0}, \mathbf{\Omega}) \end{aligned} \tag{4}$$

Both responses in this bivariate normal model, `english` and `ravens`, are partially observed continuous variables. The binary covariate, `sex`, is fully observed. This is identical to model (1) with X being `sex`.

To impute the missing values we proceed as follows. First, in order to guarantee that the correct function is being used, we need to make sure that continuous variables are stored as numeric vectors in the data frame and binary/categorical variables as `factors`. We can easily test this as follows:

```
# Check that ravens is numeric:
class(JSPmiss$ravens)
# [1] "numeric"

# Were ravens not numeric, we convert it as follows:
JSPmiss <- within(JSPmiss, ravens <- as.numeric(ravens))

# Define the data.frame with the outcomes
Y <- JSPmiss[, c("english", "ravens")]

# The imputation model requires an intercept variable (simply a column of 1's)
# to include in the covariate matrix X. The JSPmiss data set already contains
# such a variable (cons). Were it not present, we could define it as follows:
JSPmiss$cons <- 1

# Define the data.frame with the covariates:
X <- JSPmiss[, c("cons", "sex")]

# Set the seed (so we can replicate the results exactly if desired)
set.seed(1569)

# Run jomo and store the imputed data sets in a new data frame, imp
imp <- jomo(Y = Y, X = X, nburn = 1000, nbetween = 1000, nimp = 5)
```

Running this code, with the above seed, the following output is shown on screen:

```
# No clustering, using functions for single-level imputation.
# Found  2 continuous outcomes and no categorical. Using function jomo1con.
# ------------------------------------------------------------------------
# ----------------------First imputation registered.
# ------------------------------------------------------------------------
# ----------------------Imputation number  2 registered
# ------------------------------------------------------------------------
# ----------------------Imputation number  3 registered
# ------------------------------------------------------------------------
# ----------------------Imputation number  4 registered
# ------------------------------------------------------------------------
# ----------------------Imputation number  5 registered
# The posterior mean of the fixed effects estimates is:
#             cons       sex
# english 38.15030  6.408079
# ravens  25.32372 -0.556898
#
# The posterior covariance matrix is:
#           english   ravens
# english 458.39885 64.92863
# ravens   64.92863 36.41841
```

The first sentence informs us that, as we did not pass any clustering indicator to the function, `jomo` is using single-level imputation. The second sentence is telling us which of the sub-functions was used.

As we have two numeric dependent variables, `jomo1con` has been chosen.

Then, the software prints a '.' for each 10 burn-in updates of the MCMC sampler, followed by a notification that it has created the first imputed data set. It then prints a '.' for each 10 further updates of the MCMC sampler, before imputations 2, 3, 4, and 5. The default values for the burn-in and between-imputation updates are both 1000, resulting in 100 dots printed in each case.

Finally, the software prints the estimated posterior mean of the regression coefficients for `english` ($\beta_{0,e}, \beta_{1,e}$), `ravens` ($\beta_{0,r}, \beta_{1,r}$), and the elements of the covariance matrix:

$$\begin{pmatrix} \epsilon_{e,i} \\ \epsilon_{r,i} \end{pmatrix} \sim \mathcal{N}\left[\mathbf{0}, \mathbf{\Omega} = \begin{pmatrix} 458.40 & 64.93 \\ 64.93 & 36.42 \end{pmatrix}\right].$$

The same results can be obtained by running `jomo1con` in place of `jomo` with exactly the same arguments. In general, to exactly replicate the results, you will have to (i) have the data sorted in the same order and (ii) use the same seed. Sometimes we will wish to suppress the output generated by `jomo`; the option `output = 0` controls this.

Running `jomo` creates the object `imp`, which is a data frame:

```
class(imp)
#[1] "data.frame"

# with the following names:
names(imp)
#[1] "english"    "ravens"        "cons"       "sex"        "id"         "Imputation"

# and dimension:
dim(imp)
# [1] 6714    6
```

The `names` include the original variable names, inherited from `JSPmiss`, and a new variable `Imputation`, which indexes the original data (`0`) and the imputed data sets. We have five imputed data sets. Let's look at the top of the original data, and the top of the fifth imputation:

```
# View original (partially observed) data:
head(imp)

#   english ravens cons sex id Imputation
# 1      39     NA    1   1  1          0
# 2      NA     15    1   0  2          0
# 3      65     19    1   1  3          0
# 4      NA     22    1   0  4          0
# 5      30     NA    1   1  5          0
# 6      12     NA    1   0  6          0

# View last imputation (the left most column is the row number):
head(imp[imp$Imputation == 5,])

#      english   ravens cons sex id Imputation
# 5596 39.00000 32.17927    1   1  1          5
# 5597 54.37720 15.00000    1   0  2          5
# 5598 65.00000 19.00000    1   1  3          5
# 5599 46.57598 22.00000    1   0  4          5
# 5600 30.00000 21.30211    1   1  5          5
# 5601 12.00000 24.56964    1   0  6          5
```

Starting values and prior distributions

To impute missing data, `jomo` fits a Bayesian model using MCMC. Therefore, we need to provide starting values and priors for each parameter in the model. The default starting values are a matrix of zeros for the fixed effects parameter $\boldsymbol{\beta}$, and the identity matrix for the covariance matrix. In the majority of situations, changing the starting values will not change the results materially. Nevertheless, good starting values may considerably reduce the number of iterations needed for the algorithm to converge.

In order to represent the maximum uncertainty and to give the greatest weight to the data, `jomo` assumes flat improper priors for all the parameters in the model, except the covariance matrix. For

this, an inverse-Wishart prior is used, with degrees of freedom set to the minimum possible, i.e. the dimension of the covariance matrix; this represents the greatest uncertainty (least information). Changing the scale matrix of the inverse-Wishart prior may have some impact when we have a very small number of observations. In our example, with 1119 observations and only two outcomes, the impact is immaterial.

We now illustrate how to set the starting values for all the parameters, the scale matrix of the inverse-Wishart prior for the covariance matrix as well as the number of burn-in iterations for the MCMC sampler, the number of iterations between imputations, and the number of imputations:

```
# Set starting values for fixed effect parameters beta
beta.start <- matrix(1, 2, 2)

# Set starting value for covariance matrix
l1cov.start <- diag(2, 2)

# Set scale matrix of the inverse-Wishart prior for the covariance matrix:
l1cov.prior <- diag(2, 2);

# Set seed to get results below
set.seed(1569)

# Impute:
# [Note for new R users: the inputs set above have the same names as their
# corresponding options in the function. Hence, when we set <option> = <object
# name>, we have the same string on both sides of '=']

imp2 <- jomo(Y, X = X, beta.start = beta.start, l1cov.start = l1cov.start,
             l1cov.prior = l1cov.prior, nburn = 200,
             nbetween = 200, nimp = 5)
# ....................First imputation registered.
# ....................Imputation number  2 registered
# ....................Imputation number  3 registered
# ....................Imputation number  4 registered
# ....................Imputation number  5 registered
#             cons        sex
# english 38.21636  6.4469156
# ravens  25.33982 -0.5456155
#
# The posterior covariance matrix is:
#           english   ravens
# english 460.37902 65.34086
# ravens   65.34086 36.58763
```

We see no material change from the previous results. In simple problems, the default burn-in of `nburn = 1000` is often enough for the sampler to converge. Further below, we show how to visually check whether the stationary distribution has been reached.

Analysing the imputed data

Recall our substantive linear regression model above. Once we have created our imputed data sets, we follow the usual multiple imputation rules and fit our substantive model to each imputed data set, before summarising the results for final inference using Rubin's rules, which are implemented in **mitools**:

```
library(mitools) # load if not done so above

# Use the object imp which we created with the original run above. First
# convert the data frame of results imp to a list of imputations
imp.list <- imputationList(split(imp, imp$Imputation)[-1])

# Fit model to each of the 5 imputed data sets
fit.imp <- with(data = imp.list, lm(english ~ ravens + sex))

# Extract coefficients and variances
```

```
coefs <- MIextract(fit.imp, fun = coef)
vars <- MIextract(fit.imp, fun = function(x) diag(vcov(x)))

# Pool results with Rubin's rules
results <- MIcombine(coefs, vars)
summary(results)

# Multiple imputation results:
#       MIcombine.default(coefs, vars)
#               results        se     (lower     upper) missInfo
# (Intercept) -6.549183 3.0013631 -12.618685 -0.4796807     35 %
# ravens       1.767782 0.1124461   1.540232  1.9953322     36 %
# sex          7.037727 1.3021658   4.424013  9.6514414     31 %
```

There are multiple alternative implementations of Rubin's rules in R. These include `pool` from **mice**, `runMI` from **semTools**, particularly appealing with Structural Equation Models, `MI.inference` from **BaBooN** (Meinfelder, 2011), and `testEstimates` from **mitml**, which we present more in depth in the penultimate section of this paper. Other packages with their own implementation of Rubin's rules include **Amelia**, **mi**, and **lavaan.survey**.

Categorical variables

Fully observed binary covariates can be included in the X matrix of the imputation model as type `numeric`, exactly as with `sex` in this example. To include fully observed categorical covariates with three or more categories, appropriate dummy variables have to be created. For this purpose, we might use the R package **dummies** (Brown, 2012) or the function `constrasts` in base R.

jomo also readily imputes a mix of binary, categorical and continuous variables. This is done using a latent normal model (see Goldstein et al., 2009; Carpenter and Kenward, 2013, Ch. 4). To illustrate this, we continue to use the data set `JSPmiss` but now also impute the partially observed fluency level (3 categories). The underlying joint imputation model remains a multivariate normal model, but `fluent` is represented by two latent normal variables:

$$\begin{aligned} Y_{english,i} &= \beta_{0,e} + \beta_{1,e} X_{sex,i} + \epsilon_{e,i} \\ Y_{ravens,i} &= \beta_{0,r} + \beta_{1,r} X_{sex,i} + \epsilon_{r,i} \\ Y^*_{flu,1,i} &= \beta_{0,f1} + \beta_{1,f1} X_{sex,i} + \epsilon_{f1,i} \\ Y^*_{flu,2,i} &= \beta_{0,f2} + \beta_{1,f2} X_{sex,i} + \epsilon_{f2,i} \end{aligned} \tag{5}$$

where:

$$\begin{aligned} Pr(Y_{flu,i} = 1) &= Pr\left(Y^*_{flu,1,i} = \max_{j=1,2} Y^*_{flu,j,i} \text{ and } Y^*_{flu,1,i} > 0\right) \\ Pr(Y_{flu,i} = 2) &= Pr\left(Y^*_{flu,2,i} = \max_{j=1,2} Y^*_{flu,j,i} \text{ and } Y^*_{flu,2,i} > 0\right) \\ Pr(Y_{flu,i} = 3) &= Pr\left(Y^*_{flu,j,i} < 0 \text{ for } j = 1,2\right), \end{aligned} \tag{6}$$

$$\begin{pmatrix} \epsilon_{e,i} \\ \epsilon_{r,i} \\ \epsilon_{f1,i} \\ \epsilon_{f2,i} \end{pmatrix} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Omega}) \tag{7}$$

As highlighted above, in order for this model to be estimable, we need to constrain the variance-covariance matrix of $(\epsilon_{f1,i}, \epsilon_{f2,i})^T$, (i.e. the bottom right 2×2 submatrix of $\mathbf{\Omega}$) to be

$$\begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}.$$

These constraints are automatically implemented in the software.

The code for imputation is essentially the same as before, but now we need to make sure that `fluent`,

being a categorical variable, is included in the dependent variable data frame as a `factor`:

```
# convert "fluent" to factor
JSPmiss <- within(JSPmiss, fluent <- factor(fluent))

# Define the data frame with the dependent (outcome) variables for the imputation model:
Y <- JSPmiss[, c("english", "ravens", "fluent")]

# Define the data.frame with the (fully observed) covariates of the imputation model
X <- JSPmiss[, c("cons", "sex")]


# fit the model and impute the missing data:
set.seed(1569)
imp <- jomo(Y = Y, X = X)
```

This will take a little longer than the previous examples, and returns the following output:

```
# The posterior mean of the fixed effects estimates is:
#               cons        sex
# english  38.263132  6.3636755
# ravens   25.324376 -0.5548762
# fluent.1 -1.574369 -0.2318260
# fluent.2 -1.836201  0.1829313
#
# The posterior covariance matrix is:
#             english    ravens   fluent.1  fluent.2
# english  458.164036 64.437168 -11.379005 -8.888263
# ravens    64.437168 36.743243  -2.141006 -1.500893
# fluent.1 -11.379005 -2.141006   1.000000  0.500000
# fluent.2  -8.888263 -1.500893   0.500000  1.000000
```

The output illustrates that `jomo` recognized that `fluent` was a factor variable and therefore used the function for the imputation of mixed data types.

The matrix posterior mean of the fixed effect estimates links directly to (5). Specifically, $\hat{\beta}_{0,e} = 38.26, \hat{\beta}_{1,e} = 6.36, \ldots, \hat{\beta}_{0,f2} = -1.84, \hat{\beta}_{1,f2} = 0.18$. Likewise, the posterior means of the covariance matrix terms are labelled, and correspond directly to (5). Specifically,

$$\widehat{\text{Var}} \begin{pmatrix} \epsilon_{e,i} \\ \epsilon_{r,i} \\ \epsilon_{f1,i} \\ \epsilon_{f2,i} \end{pmatrix} = \begin{pmatrix} 458.16 & 64.44 & -11.38 & -8.89 \\ 64.44 & 36.74 & -2.14 & -1.50 \\ -11.38 & -2.14 & 1 & 0.5 \\ -8.89 & -1.50 & 0.5 & 1 \end{pmatrix}.$$

Calling the relevant sub-function (`jomo1mix`) is possible again but a bit more complex, because continuous and categorical outcomes must be passed as separate arguments.

We can specify starting values explicitly if we wish. To specify all the starting values, we need to specify $n - 1$ starting β values for each n-category variable and a proper starting value for the covariance matrix. For example, in the present case, `fluent` has 3 categories, which are modelled with 2 latent normal variables. As a result, $\boldsymbol{\beta}$ is a 2×4 matrix of regression coefficients, and the covariance matrix is of size 4×4 (i.e., two predictors with two continuous and two latent dependent variables). So, continuing with `Y` and `X` defined as above:

```
# Starting value for beta
beta.start <- matrix(0, 2, 4) # Specify a 2 by 4 matrix of zeros

# Starting value for covariance matrix; the software disregards impossible values:
l1cov.start <- diag(2, 4)

set.seed(1569)
imp <- jomo(Y = Y, X = X, beta.start = beta.start, l1cov.start = l1cov.start)
```

As our starting values are different from the default, we get slightly different estimates of the posterior means.

While the software is designed for unordered categorical data, it can be used for ordinal data too. Our simulation results show that if variables are truly ordinal it gives good results with only a marginal loss in efficiency (Quartagno and Carpenter, 2019).

jomo: tutorial with multilevel data

When we wish to impute missing data with a multilevel substantive model, our imputation model should itself preserve the multilevel structure (Lüdtke et al., 2017; Andridge, 2011); one key benefit of **jomo** is that it allows us to do this.

Three different approaches for multilevel multiple imputation have been implemented in **jomo**, providing a flexible framework for the treatment of missing data in multilevel data sets. They allow:

1. imputation with a common level-1 covariance matrix across level-2 units (the default);
2. imputation with a cluster-specific level-1 covariance matrices, and
3. imputation allowing for the level-1 covariance matrix to be randomly distributed across level-2 units, following the proposal by Yucel (2011) and as developed by Quartagno and Carpenter (2016).

Option (2) requires sufficient data within each level-2 unit to estimate the covariance matrix; option (3) is a practical choice when we suspect there is heterogeneity across level-2 units, but there is insufficient information within each level-2 unit for option (2).

We illustrate the software again with the `JSPmiss` data set, distributed with the package. Let j index school and i students within schools. Our substantive model is:

$$Y_{english,i,j} = \alpha_0 + \alpha_1 \text{ravens}_{i,j} + \alpha_2 \text{sex}_{i,j} + \alpha_3 1[\text{fluent}_{i,j} == 2] + \alpha_4 1[\text{fluent}_{i,j} == 3] + u_j + \epsilon_{i,j} \tag{8}$$

$$\begin{aligned} u_j &\sim N(0, \sigma_u^2) \\ e_{i,j} &\sim N(0, \sigma_e^2). \end{aligned} \tag{9}$$

We now use multilevel imputation for the missing data. This is done by extending the joint imputation model described above to the multilevel setting. As before, the variables `english`, `ravens` and `fluent` are responses, and the fully observed variable `sex` a covariate. The imputation model has random intercepts at level 2, and a common level-1 covariance matrix (approach 1 above).

$$\begin{aligned} Y_{english,i,j} &= \beta_{0,e} + \beta_{1,e} X_{sex,i,j} + u_{e,j} + \epsilon_{e,i,j} \\ Y_{ravens,i,j} &= \beta_{0,r} + \beta_{1,r} X_{sex,i,j} + u_{r,j} + \epsilon_{r,i,j} \\ Y^*_{flu,1,i,j} &= \beta_{0,f1} + \beta_{1,f1} X_{sex,i,j} + u_{f1,j} + \epsilon_{f1,i,j} \\ Y^*_{flu,2,i,j} &= \beta_{0,f2} + \beta_{1,f2} X_{sex,i,j} + u_{f2,j} + \epsilon_{f2,i,j} \end{aligned} \tag{10}$$

where:

$$\begin{aligned} Pr(Y_{flu,i,j} = 1) &= Pr\left(Y^*_{flu,1,i,j} = \max_{k=1,2} Y^*_{flu,k,i,j} \text{ and } Y^*_{flu,1,i,j} > 0\right) \\ Pr(Y_{flu,i,j} = 2) &= Pr\left(Y^*_{flu,2,i,j} = \max_{k=1,2} Y^*_{flu,k,i,j} \text{ and } Y^*_{flu,2,i,j} > 0\right) \\ Pr(Y_{flu,i,j} = 3) &= Pr\left(Y^*_{flu,k,i,j} < 0 \text{ for } k = 1,2\right) \end{aligned} \tag{11}$$

$$\boldsymbol{\epsilon}_{i,j} = \begin{pmatrix} \epsilon_{e,i,j} \\ \epsilon_{r,i,j} \\ \epsilon_{f1,i,j} \\ \epsilon_{f2,i,j} \end{pmatrix} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega}_e) \qquad \mathbf{u}_j = \begin{pmatrix} u_{e,j} \\ u_{r,j} \\ u_{f1,j} \\ u_{f2,j} \end{pmatrix} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega}_u) \tag{12}$$

This is the same as model (5), apart from the addition of the covariance matrix $\boldsymbol{\Omega}_u$ for the vector of random intercepts, $\mathbf{u}_j$.

Apart from specifying the level-two identifier, imputing the missing values proceeds the same as before:

```
# Define cluster/group indicator
clus <- JSPmiss$school

# Define the data.frame with outcomes of the imputation model
JSPmiss <- within(JSPmiss, fluent <- factor(fluent))
Y <- JSPmiss[, c("english", "ravens", "fluent")]

# Define the data frame with covariates of the imputation model
X <- JSPmiss[, c("cons", "sex")]

# Perform multilevel imputation:
set.seed(1569)
imp <- jomo(Y = Y, X = X, clus = clus, nburn = 2000, nbetween = 1000, nimp = 5)

# Clustered data, using functions for two-level imputation.
# Found  2 continuous outcomes and  1 categorical. Using function jomo1ranmix.
# ..................................................................................
# ..................................................................................
# .........................First  imputation registered.
# ..................................................................................
# .............Imputation number  2 registered
# ..................................................................................
# .............Imputation number  3 registered
# ..................................................................................
# .............Imputation number  4 registered
# ..................................................................................
# .............Imputation number  5 registered
# The posterior mean of the fixed effects estimates is:
#               cons        sex
# english  38.145706  5.7460263
# ravens   25.207758 -0.5280992
# fluent.1 -1.889772 -0.1980197
# fluent.2 -2.133932  0.3046025
#
# The posterior mean of the random effects estimates is:
#     english.Z1   ravens.Z1 fluent.1.Z1  fluent.2.Z1
# 1   -9.3394300 -2.39167726  0.30978733  0.359526051
# 2   -0.6734767  0.09111794  0.16371750 -0.140409675
# 3    2.0366252  1.66523294 -0.52408207 -0.433563083
# 4   -0.7524001  0.15352258 -0.14147716  0.016889639
# 5    7.4086502  1.79372314 -0.46164853  0.029898809
# [...]
# 46   7.6419790  1.93470732 -0.43765561  0.099247985
# 47   0.1680530  0.56124947  1.02219296  1.083265124
# 48  -1.9106040 -0.10743067 -0.24463642 -0.101618838
# 49   1.5480441  0.23546197 -0.31258927 -0.072163795
# 50   2.8932869  1.21095460 -0.46288361 -0.351124776
#
# The posterior mean of the level 1 covariance matrices is:
#             english    ravens  fluent.1  fluent.2
# english  394.192120 53.461640 -9.709166 -7.894670
# ravens    53.461640 32.882006 -1.636866 -1.391647
# fluent.1  -9.709166 -1.636866  1.000000  0.500000
# fluent.2  -7.894670 -1.391647  0.500000  1.000000
#
# The posterior mean of the level 2 covariance matrix is:
#             english.Z1  ravens.Z1 fluent.1.Z1 fluent.2.Z1
# english*Z1   67.791552 13.5839586  -4.1394619  -2.2062157
# ravens*Z1    13.583959  4.1667944  -1.0068761  -0.5742672
# fluent.1*Z1  -4.139462 -1.0068761   0.6051437   0.3374155
# fluent.2*Z1  -2.206216 -0.5742672   0.3374155   0.4628897
```

Simply including `clus` was enough to tell `jomo` that we have a multilevel structure and to impute accordingly. The assumed model included a random intercept for each outcome variable; we will see later in this section how to specify the design matrix for the random effects differently. The output

follows the same format as that from fitting (5) above, with two additions:

1. we obtain the posterior mean of the $\mathbf{u}_j$, the random intercepts for each of the 5 responses, for each of the $j = 1, \ldots, 50$ schools. For example, for school 2, the posterior means are

$$\begin{pmatrix} u_{e,2} \\ u_{r,2} \\ u_{f1,2} \\ u_{f2,2} \end{pmatrix} = \begin{pmatrix} -0.67 \\ 0.09 \\ 0.16 \\ -0.14 \end{pmatrix}$$

2. we obtain the estimated level-2 variance covariance matrix of the random intercepts (common across all ten cities):

$$\widehat{\text{Var}} \begin{pmatrix} u_{e,j} \\ u_{r,j} \\ u_{f1,j} \\ u_{f2,j} \end{pmatrix} = \begin{pmatrix} 67.79 & 13.58 & -4.14 & -2.21 \\ 13.58 & 4.17 & -1.01 & -0.57 \\ -4.14 & -1.01 & 0.61 & 0.34 \\ -2.21 & -0.57 & 0.34 & 0.46 \end{pmatrix}.$$

Analysing the imputed data

To fit the substantive multilevel model to each imputed data set we proceed as before:

```
imp.list <- imputationList(split(imp, imp$Imputation)[-1])

# Fit model to each of the 5 imputed data sets
fit.imp <- with(data = imp.list, lmer(english ~ ravens + sex + factor(fluent) + (1|clus)))

# Extract coefficients and variances
coefs <- MIextract(fit.imp, fun = fixef)
vars <- MIextract(fit.imp, fun = function(x) diag(vcov(x)))

# Pool results with Rubin's rules
results <- MIcombine(coefs, vars)
summary(results)

# Multiple imputation results:
#       MIcombine.default(coefs, vars)
#                     results        se     (lower    upper) missInfo
# (Intercept)     -16.133067 3.7371534 -23.521995 -8.744139     18 %
# ravens            1.622946 0.1197557   1.374339  1.871554     48 %
# sex               6.837264 1.1452177   4.573386  9.101142     18 %
# factor(fluent)1   5.000691 4.2413878  -3.420624 13.422006     22 %
# factor(fluent)2  14.345070 3.1391521   8.057088 20.633051     29 %
```

In order to get multiple imputation inference for the random coefficients, we recommend using the **mitml** package, as described in the penultimate section below.

Design matrix for random effects

We now show how to specify additional random effects in the imputation model, apart from the random intercept that is included by default. This is done by specifying the design matrix of the random effects, Z. When this is not specified, it defaults to a random intercept. When it is specified by the user, the random intercept has to be included (if desired).

```
Z <- JSPmiss[, c("cons", "sex")] # intercept and sex have random effects

imp <- jomo(Y = Y, X = X, Z = Z, clus = clus)
# Output omitted
```

Starting values and prior distributions

The starting values of the sampling algorithm can again be overridden by the user, thus potentially leading to better sampling behaviour and faster convergence. In comparison with the single-level

case, we now have two additional sets of parameters: (i) the matrix of random effects, whose rows contain the random effects vectors for all level-2 units (`u.start`), and (ii) the level-2 covariance matrix (`l2cov.start`). Note that with small numbers of level-2 units, the impact of the scale matrix of the prior for the level-2 covariance matrix can be substantial. We proceed as follows:

```
beta.start <- matrix(1, 2, 4) # initialise fixed effects to zero
u.start <- matrix(0.5, nlevels(JSPmiss$school), 4) # initialise all random effects to 0.5
l1cov.start <- diag(2, 4)     # initialise diagonal covariance matrix of 2's for level 1
l2cov.start <- diag(2, 4)     # initialise diagonal covariance matrix of 2's for level 1
l2cov.prior <- diag(2, 4);    # set scale matrix of inverse-Wishart prior for level-2
                              # covariance matrix

set.seed(1569)
imp <- jomo(Y = Y, X = X, clus = clus, beta.start = beta.start, u.start = u.start,
            l1cov.start = l1cov.start, l2cov.start = l2cov.start,
            l2cov.prior = l2cov.prior, nburn = 2000, nbetween = 1000, nimp = 5)

# Output omitted; as these are not all the default values, the posterior means differ
# from the previous results by Monte Carlo error.
```

Cluster-specific covariance matrices

In some cases, it is implausible that all of the clusters share the same level-1 covariance matrix. For example, when aggregating individual patient data from different studies to perform a meta-analysis, it is often reasonable to assume that covariance matrices are different across studies.

Continuing to use (10) as an example, the only difference from before is that now the level-1 covariance matrix is not modelled as constant but as different across level-2 units. Thus instead of $\mathbf{\Omega}_e$, we have $\mathbf{\Omega}_{e,j}$, $j = 1, \ldots, 50$. We fit this model by specifying the additional argument `meth = "fixed"`:

```
# Define the data.frame with outcomes of the imputation model
Y <- JSPmiss[, c("english", "ravens", "fluent")]

# Define the data.frame with covariates of the imputation model
X <- JSPmiss[, c("cons", "sex")]

# Define cluster/group indicator
clus <- JSPmiss$school

# Fixed cluster-specific covariance matrices
imp2 <- jomo(Y = Y, X = X, clus = clus, meth = "fixed")
# Output omitted
```

Note that the output is now considerably longer, as the posterior mean for each of the level 1 covariance matrices is reported. Compared to the previous models, this model is more complex and requires estimation of a larger number of parameters.

Random cluster-specific covariance matrices

There are several reasons why we may wish to go beyond the setting above and allow the covariance matrices to be random across level-2 units. For example, we may have reason to believe that different level-2 units have different covariance matrices but that the number of observations on some of these level-2 units is insufficient to estimate level-2 specific covariance matrices reliably. In this case, sharing information across level-2 units is desirable. Another situation is when some variable is fully missing from some clusters, and therefore it is necessary to share information with clusters where it was observed through the specification of a hierarchical distribution for the covariance matrices.

Continuing to use (10) as an example, the cluster-specific covariance matrices are now assumed to follow a specific distribution. Thus instead of $\mathbf{\Omega}_{e,j}$, we have $\mathbf{\Omega}_{e,j} \sim IW(a, S)$ $j = 1, \ldots, 10$. Here, a and S are the degrees of freedom and scale matrix of the inverse-Wishart distribution, respectively.

We can simply fit this model by specifying the option `meth = "random"`:

```
imp3 <- jomo(Y = Y, X = X, clus = clus, meth = "random")
```

and then analyse the imputed data sets in the usual way. For full details on the algorithm used to fit the random covariance matrices algorithm initially proposed by Yucel (2011), see the appendix of Quartagno and Carpenter (2016).

The sub-function called by `jomo` for this type of data is `jomo1ranmixhr`, which can be called directly but with a slightly more complex syntax.

With random covariance matrices, we have one further parameter, `a`, denoting the degrees of freedom of the inverse-Wishart distribution for the cluster-specific covariance matrices. The default starting value for this parameter, `a.start`, is the minimum possible, i.e., the dimension of the level-1 covariance matrix. This is also the default for the hyperparameter `a.prior` of the chi-square prior distribution for a.

With random level-1 covariance matrices we can also specify starting values for the n_{clus} covariance matrices. Below, we show how to do this by (i) first creating the matrix for the first level 2 unit, a 4-by-4 diagonal matrix with all entries '2', using `diag(2,4)` then (ii) stacking 50 copies of this:

```
# Starting values for the 5 by 5 level-1 covariance matrix for the first level-2 unit
l1cov.start.1 <- diag(2, 4)
# Stack 10 copies of this matrix (one for each of the level-2 units)
l1cov.start <- matrix(l1cov.start.1, nrow = 4 * nlevels(JSPmiss$school), ncol = 4,
                      byrow = TRUE)

# Choose a starting value for the degrees of freedom, a (automatically >= 5)
a.start <- 7

# Run jomo
imp <- jomo(Y = Y, X = X, clus = clus, l1cov.start = l1cov.start, a = a.start,
            meth = "random")
```

Imputing level-2 variables

In many applications, we have variables describing aspects of the level-2 units, and these may also have missing values. For example, in longitudinal studies, time-independent variables related to individuals (level-2 units), such as sex or the baseline variable `ravens`, may be affected by missing data. We can impute any missing level-2 values naturally with **jomo** (Carpenter and Kenward, 2013, Ch.9). As described above, we can use either a single common or multiple cluster-specific level-1 covariance matrices.

To illustrate this, we use a new data set, `ExamScores`. The fully observed version of this data set is again available with MLwiN and **R2MLwiN**, and it represents a subset of a larger data set of examination results from six inner London Education Authorities. The partially observed version that we use here is available with **jomo**. As in the previous example, this data set contains data from pupils (level 1) clustered in schools (level 2). Some of the variables are related to students at level 1 (`normexam`, a normalised version of exam score at age 16 and `standlrt`, London Reading Test (LRT) score at age 11). The other variables describe features of the schools at level 2, for example `avslrt` (continuous), representing the average LRT score for pupils in a particular school.

The two-level multivariate normal joint model for `normexam` (n), `standlrt` (s) and `avslrt` (a) is:

$$\begin{aligned} Y_{normexam,i,j} &= \beta_{0,n} + u^{(1)}_{n,j} + \epsilon_{n,i,j} \\ Y_{standlrt,i,j} &= \beta_{0,s} + u^{(1)}_{s,j} + \epsilon_{s,i,j} \\ Y_{avslrt,j} &= \beta 0, a + u^{(2)}_{a,j} \end{aligned} \tag{13}$$

$$\boldsymbol{\epsilon}_{i,j} = \begin{pmatrix} \epsilon_{n,i,j} \\ \epsilon_{s,i,j} \end{pmatrix} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega}_e) \qquad \mathbf{u}_j = \begin{pmatrix} u^{(1)}_{n,j} \\ u^{(1)}_{s,j} \\ u^{(2)}_{a,j} \end{pmatrix} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Omega}_u), \tag{14}$$

where the superscripts (1) and (2) indicate the level-2 random effect for a level-1 and level-2 covariate, respectively (level-2 covariates have no level-1 residuals).

Fitting this model is very similar to fitting previous models; we simply have to define the level-2 variables.

```
# Define data.frame with level-1 outcomes of imputation model
```

```
Y <- ExamScores[, c("normexam", "standlrt")]

# Define data.frame with level-2 outcomes of imputation model
Y2 <- ExamScores[, "avslrt", drop = FALSE]

# Define clustering indicator
clus <- ExamScores$school

# Run jomo
set.seed(1569)
imp <- jomo(Y = Y, Y2 = Y2, clus = clus)

# 2-level data, using functions for two-level imputation.
# Found  2 level 1 continuous and  0 level 1 categorical outcomes, 1  level 2 continuous
# and  0  level 2 categorical outcomes. Using function jomo2com, assuming common
# covariance matrix across clusters

# Output partially omitted [...]

# The posterior mean of the fixed effects estimates is:
#                    X1
# normexam -0.009685026
# standlrt -0.002263212

# The posterior mean of the level 2 fixed effects estimates is:
#               X2.1
# avslrt -0.03268919

# The posterior mean of the random effects estimates is:
#     normexam.Z1  standlrt.Z1       avslrt
# 1   0.485259093  0.146952219  0.198863738
# 2   0.896523031  0.378616363  0.429914831
# 3   0.871480465  0.464155974  0.546844580
# [...]
# 63  0.586499213  0.161779179  0.188900255
# 64  0.310554391  0.391077349  0.466833301
# 65 -0.320429589 -0.086303587 -0.202661000

# The posterior mean of the level 1 covariance matrix is:
#           normexam  standlrt
# normexam 0.8536697 0.5045721
# standlrt 0.5045721 0.8876558

# The posterior mean of the level 2 covariance matrix is:
#             normexam.Z1 standlrt.Z1     avslrt
# normexam*Z1   0.2023980  0.10005877 0.10599527
# standlrt*Z1   0.1000588  0.12408893 0.09844939
# avslrt        0.1059953  0.09844939 0.13067251
```

As above, we can specify the starting values for all the parameters in the model, and in particular the parameter of the level-2 variable β_2 with the input `l2.beta.start`. As we noted above, with small cluster sizes, the scale matrix for the prior of the level 2 covariance matrix, `l2cov.prior`, may have a non-negligible impact on the results.

The sub-functions for imputation of level 2 variables are `jomo2com` and `jomo2hr`. Cluster-specific covariance matrices can be specified as before by setting `meth = "common"` or `meth = "random"`.

Checking convergence of MCMC

When using MCMC for model fitting and imputation, it is crucial to be confident of having reached the stationary distribution of the sampler before starting to register imputations. We do this by monitoring the parameter chains generated by the MCMC algorithm. To facilitate this, we introduced a `.MCMCchain` version of each function in the package, which allows convergence assessment without imputation. We illustrate this with a simple example:

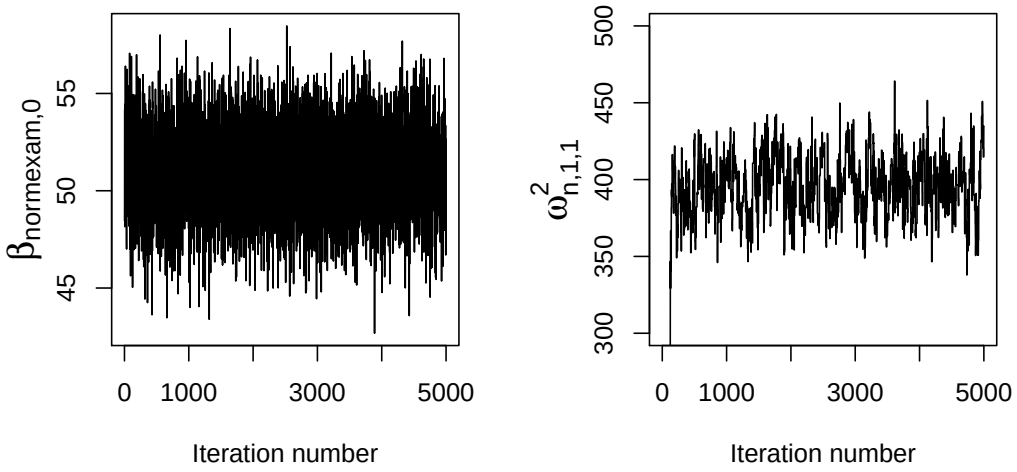


Figure 1: MCMC chain for $\beta_{e,0}$ (left panel) and $\omega^2_{e,1,1}$ (right panel).

```
# Define data.frames with outcomes and covariates of imputation model
Y <- JSPmiss[, c("english", "ravens")]
X <- JSPmiss[, c("cons", "sex")]

# Run jomo.MCMCchain
imp <- jomo.MCMCchain(Y = Y, X = X, nburn = 5000)
```

This updates the sampler `nburn` times, but does not create any imputed data sets. Instead, the output of this function is a list containing three elements:

- `finimp`: the final state of the data set, which would be the first imputation if we ran the `jomo` function with `nburn` burn-in iterations;
- `collectbeta`: a three-dimensional array containing the fixed effect parameter draws at each of the `nburn` iterations;
- `collectomega`: a three-dimensional array containing the level-1 covariance matrix draws at each of the `nburn` iterations;

When running the corresponding `.MCMCchain` functions for multilevel imputation we will also have:

- `collectu`: a three-dimensional array containing the random effects draws at each of the `nburn` iterations;
- `collectcovu`: a three-dimensional array containing the level-2 covariance matrix draws at each of the `nburn` iterations;

We can then check the convergence of the sampler by looking at the trace plot for each parameter value. For example, in Figure 1 (left panel), we can see the plot for $\beta_{e,0}$, which we obtain by running:

```
plot(imp$collectbeta[1, 1, 1:5000], type = "l", ylab = expression(beta["e,0"]),
     xlab = "Iteration number" )
```

In this case, we can see that a burn in of 100–500 is reasonable; the sampler clearly converges very quickly.

Plots for elements of the covariance matrix updated though Metropolis-Hastings steps may look different, because these chains have higher auto-correlation (as they are not guaranteed to be updated at each iteration). The right panel of Figure 1 gives an example; this was obtained by running the following commands:

```
JSPmiss <- within(JSPmiss, fluent <- factor(fluent))
Y <- JSPmiss[, c("english", "ravens", "fluent")]
```

```
X <- JSPmiss[, c("cons", "sex")]

imp2 <- jomo.MCMCchain(Y = Y, X = X, nburn = 5000)

plot(imp2$collectomega[1, 1, 1:5000], type = "l", ylab = expression(omega[e,1,1]^2),
     xlab = "Iteration number", ylim=c(300,500) )
```

Note there is little point in plotting the constrained elements of the covariance matrix — these will always give a straight line!

Using jomo in practice

The `.MCMCchain` functions only register a single imputation, but the state of the sampler at this point is captured. This provides a mechanism for combining multiple runs of `.MCMCchain` and/or `jomo` in a flexible manner, for example, to obtain the full set of posterior draws for the model parameters with multiple runs of `.MCMCchain` or to generate mildly informative prior distributions to be used with jomo. Specifically, at the end of the `.MCMCchain` run, the following objects capture the state of the MCMC sampler:

- `start.imp` for the level-1 variables with missing values;
- (where present) `l2.start.imp` for level-2 variables with missing values, and
- `finimp.latnorm`: the final state of the imputed data set using latent normals in place of categorical variables.

In practice, we typically need to use `finimp.latnorm`, together with the last value of the fixed parameters and the covariance matrix at level 1 (and at level 2 if present). The following code illustrates the approach:

```
# Define data frames for outcomes and covariates of imputation model and
# convert "fluent" to factor
JSPmiss <- within(JSPmiss, fluent <- factor(fluent))
Y <- JSPmiss[, c("english", "ravens", "fluent")]
X <- JSPmiss[, c("cons", "sex")]

# Run jomo to register 2 imputations
set.seed(1569)
imp <- jomo(Y = Y, X = X, nimp = 2)

# OR, run jomo.MCMCchain to register first imputation
set.seed(1569)
imp1 <- jomo.MCMCchain(Y = Y, X = X)

# Capture the state of the sampler as starting values for the second set of iterations:
beta.start <- imp1$collectbeta[,,1000]   # capture the fixed parameter values
l1cov.start <- imp1$collectomega[,,1000] # capture the level-1 covariance matrix values
start.imp <- imp1$finimp.latnorm         # capture the final imputed data set (with
                                         # latent normals for categorical variables)

# Run jomo.MCMCchain to register second imputation
imp2 <- jomo.MCMCchain(Y = Y, X = X, beta.start = beta.start, l1cov.start = l1cov.start,
                       start.imp = start.imp, nburn = 1000)
```

In practice, it often works well to use this function to find plausible initial values for the scale matrices of the level-1 and level-2 covariance matrix priors. This allows us to provide 'weakly informative' priors consistent with the data, and avoids imputations being unnecessarily variable.

To do this, we run `jomo.MCMCchain` first, using the default prior. We retain the last draw (or the posterior mean of the latter part of the chain) as the covariance matrix prior. We use these to assign values to `l1cov.prior` or `l2cov.prior` and then we apply `jomo` as usual. Specifically:

```
# Define data frame as usual
JSPmiss <- within(JSPmiss, fluent <- factor(fluent))
Y <- JSPmiss[, c("english", "ravens", "fluent")]
X <- JSPmiss[, c("cons", "sex")]
```

```
# Run jomo.MCMCchain with default prior
imp1 <- jomo.MCMCchain(Y = Y, X = X)

# Collect posterior mean of covariance matrix
l1cov.guess <- apply(imp1$collectomega, c(1, 2), mean)
# Multiply by degrees of freedom, i.e. dimension of the matrix (4), to get scale matrix
l1cov.prior <- l1cov.guess*4

# Run jomo
imp <- jomo(Y = Y, X = X, l1cov.prior = l1cov.prior)
```

However, if a proper prior guess for the value of all the parameters was available, this would be preferable, as it would avoid using the same data twice, to fit the model and to estimate the hyperparameters of the priors.

When using **jomo**, we recommended the following workflow:

1. Before running the imputation model (which may take some time), perform a "dry run", to check the software is fitting the model we intended. We can do this using the `.MCMCchain` function with `nburn = 2` and checking the output.
2. Re-run the same function for a larger number of iterations (e.g. 5000) and analyse the resulting trace and autocorrelation plots to choose a sensible number of burn-in and between-imputation iterations for the final imputation process.
3. Run the **jomo** function for the chosen number of iterations.
4. Fit the substantive model on the imputed data sets and apply Rubin's rules.

mitml: an alternative interface to jomo

The **mitml** package (Grund et al., 2016c) provides an alternative interface to joint modeling multiple imputation with **jomo**. Originally created as an interface to the **pan** package, **mitml** also provides access to most of the features implemented in **jomo** and includes a number of additional tools for managing, visualising, and analysing multiply imputed data sets.

Specification of the imputation model

The main interface to **jomo** is provided by the function `jomoImpute`, which offers two convenient ways of specifying the imputation model. The first option uses a `formula`-based syntax similar to packages for multilevel modelling such as **lme4**, **nlme** (Pinheiro et al., 2017), and others. The following operators can be used to define such a `formula`:

- `~` : separates the dependent variables (left-hand side) and predictor variables (right-hand side) of the imputation model;
- `+` : adds dependent and predictor variables to the model;
- `*` : adds interactions of two or more predictors to the model;
- `|` : specifies the cluster indicator and adds cluster-specific random effects to the model (e.g., `1|school`), and
- `I()` : defines additional transformation of predictor variables to be included in the model.

For example, to fit the imputation model in (10) for the substantive model in (8) with `JSPmiss`, the model `formula` can be specified as:

```
fml <- english + ravens + fluent ~ sex + (1|school)
```

The imputation is then run with `jomoImpute` by specifying the incomplete data, the imputation model, the number of imputations (`m`), and the number of iterations for burn-in (`n.burn`) and between imputations (`n.iter`). Like **jomo**, `jomoImpute` requires that categorical variables are formatted as `factors`.

```
# Convert "fluent" to factor
JSPmiss <- within(JSPmiss, fluent <- factor(fluent))

# Run imputation
imp <- jomoImpute(data = JSPmiss, formula = fml, n.burn = 1000, n.iter = 1000, m = 5,
                  seed = 1569)
```

In addition, imputation models can be run independently in subsets of the data. For this purpose, `jomoImpute` includes an optional `group` argument, denoting the name of a variable in the data set. If specified, the imputation model is run separately for each level of `group`.

As an alternative to specifying a `formula`, the imputation model can be specified with the `type` argument of `jomoImpute`. The `type` argument is an integer vector denoting the role of each variable in the imputation model. The following values are supported:

- `1` : dependent variables (fully or partially observed);
- `2` : predictor variables (fully observed) with fixed effect;
- `3` : predictor variables (fully observed) with fixed and random effect;
- `-1` : (optional) grouping variable; if specified, imputations are run separately within each group;
- `-2` : cluster indicator, and
- `0` : variables excluded from the imputation model.

In applications with missing data at both level 1 and 2, `formula` and `type` are specified as a list of two formulas or type vectors, denoting the imputation model for variables at level 1 and 2, respectively.

Analysis of the imputed data sets

The **mitml** package can also be used to manage, visualise, and analyse the imputed data. For example, the `summary` and `plot` methods display information about the imputed data object and the convergence of the MCMC algorithm.

```
# Summarize model and display convergence statistics
summary(imp)

# Call:

# jomoImpute(data = JSPmiss, formula = fml, n.burn = 1000, n.iter = 1000,
#     m = 5, seed = 1569)
#
# Cluster variable:         school
# Target variables:         english ravens fluent
# Fixed effect predictors:  (Intercept) sex
# Random effect predictors: (Intercept)
#
# Performed 1000 burn-in iterations, and generated 5 imputed data sets,
# each 1000 iterations apart.
#
# Potential scale reduction (Rhat, imputation phase):
#
#          Min   25%  Mean Median   75%   Max
# Beta:  1.000 1.002 1.009  1.011 1.013 1.021
# Psi:   1.001 1.002 1.006  1.004 1.006 1.019
# Sigma: 1.000 1.002 1.054  1.014 1.058 1.258
#
# Largest potential scale reduction:
# Beta: [1,3], Psi: [4,2], Sigma: [3,1]
#
# Missing data per variable:
#     school english ravens fluent id sex behaviour cons
# MD% 0      21.1    22.0   21.0   0  0   0         0
#
# Display convergence plots (not shown here)
plot(imp, trace = "all")
```

The function `mitmlComplete` can be used to extract a list of imputed data sets. Each data set can be transformed and analysed with the functions `with` and `within` similar to base R. For example, the following code extracts the imputed data and fits the model in (8) to each of the data sets:

```
# Extract list of completed data sets
imp.list <- mitmlComplete(imp, print = "all")
```

```
# Fit the substantive model to each of the imputed data sets
fit.imp <- with(imp.list, lmer(english ~ ravens + sex + fluent +  (1|school)))
```

Finally, **mitml** allows pooling the results obtained from the imputed data sets. For example, `testEstimates` can be used to pool the estimates of individual parameters such as fixed effects and variance components (Rubin's rules).

```
testEstimates(fit.imp, var.comp = TRUE)

# Call:
#
# testEstimates(model = fit.imp, var.comp = TRUE)
# Final parameter estimates and inferences obtained from 5 imputed data sets.
#              Estimate Std.Error   t.value        df   P(>|t|)       RIV       FMI
# (Intercept)   -16.133     3.737    -4.317   139.189     0.000     0.204     0.181
# ravens          1.623     0.120    13.552    21.626     0.000     0.755     0.476
# sex             6.837     1.145     5.970   142.006     0.000     0.202     0.179
# fluent1         5.001     4.241     1.179    94.051     0.241     0.260     0.223
# fluent2        14.345     3.139     4.570    56.201     0.000     0.364     0.292
#
#                             Estimate
# Intercept~~Intercept|school   32.231
# Residual~~Residual           291.538
# ICC|school                     0.099
# Unadjusted hypothesis test as appropriate in larger samples.
```

Many different pooling methods are supported by **mitml**, including Rubin's rules with and without correction for smaller samples (`testEstimates`), pooled Wald and likelihood-ratio tests (LRTs) for multiple parameters and model comparisons (`testModels`, `anova`), and tests of constraints on the model parameters via the "delta method" (`testConstraints`, see Casella and Berger, 2002).

Note that, in order to use **mitml** directly with **jomo**, the imputed data must be converted to the `mitml.list` format. This conversion can be achieved with the function `jomo2mitml.list`.

Simulations and applications

The **jomo** package has been extensively evaluated in simulation studies and has been used in various applications. In this section, we provide a brief overview of these studies. For continuous data, Quartagno and Carpenter (2016); Audigier et al. (2018); Grund et al. (2018c,b) showed that jomo provides accurate parameter estimates and inferences. Quartagno and Carpenter (2019); Audigier et al. (2018); Grund et al. (2018c,b) provided similar results for binary categorical data, and Quartagno and Carpenter (2019) showed that the same procedures can be used for categorical and ordinal data. Similar findings were reported by Grund et al. (2018a,b,c) for missing data at level 2 and by Quartagno and Carpenter (2016) for applications with group-specific fixed or random covariance matrices at level 1. Further, **jomo** has been used to study the performance of MI for handling missing data in clustered randomised trials (Hossain et al., 2017a,b) and matched case-control studies (Seaman and Keogh, 2015). Finally, it has been used in applications, particularly, but not exclusively, for imputation of missing data in individual patient data meta-analyses (e.g., (Bloos et al., 2017)).

Conclusions and further developments

In this article, we have introduced a flexible new package for performing joint modelling multiple imputation for multilevel data. This package provides three important contributions: (i) it handles mixed data types, including continuous, categorical and binary data in a flexible way, (ii) it allows for either a common level-1 covariance matrix across level-2 units , cluster-specific level-1 covariance matrices, or random level-1 covariance matrices, and (iii) it gives valid imputation of missing values on level-2 variables. This makes **jomo** an effective choice for treating missing data in many applications, including single-level and multilevel data, cross-sectional and longitudinal data, and meta-analyses with individual participant data.

As with all statistical techniques, multiple imputation has to be used carefully. In particular:

- We should check that the stationary distribution has been reached, before acting on our results. As described above, the package provides tools to facilitate this.

- With many (level-1) variables and relatively few observations, a careful choice of the prior for the level-1 covariance matrix is important. We recommend a weakly informative prior and, in particular, following the strategy described above, running the `.MCMCchain` functions to find a sensible choice for the scale matrices for the inverse-Wishart priors.
- Like most imputation software, ours assumes that data are MAR. If data are MNAR, the results of analyses under MAR may be biased.

In this paper we present functions for multilevel imputation, which make better and more efficient use of all the available data compared to ad-hoc strategies, like imputing including cluster as a fixed effect or imputing separately by cluster. The first approach has been discussed in various publications (Lüdtke et al., 2017; Audigier et al., 2018; Drechsler, 2015) which broadly concluded that the approach is unsatisfactory for small clusters and low intra-cluster correlation. Additionally, under this approach dealing with random slopes is problematic and it is not possible to impute systematically missing variables. Similar considerations are likely to hold for the second strategy consisting in imputing separately by cluster, with the additional complication that level 2 variables cannot be imputed with this method.

If we are imputing a variable which has a random slope in the substantive model (Grund et al., 2016a), then (i) as usual, this variable will be a response in the imputation model and (ii) we should also allow its association with the outcome to be cluster-specific in the imputation model by allowing the level-1 covariance matrix to be random across level-2 units. However, although this approach performs better than a simpler one using a common covariance matrix, it is not a perfectly compatible approach (Quartagno and Carpenter, 2018; Enders et al., 2018), and functions for substantive model compatible imputation (Goldstein et al., 2014) should be preferred to impute missing data in those settings, when possible. These have been recently added to **jomo** and they will be presented in a second paper soon. When interactions or non-linear terms are present in the model of interest, ignoring them in the imputation model may lead to bias; instead, they should be included as covariates (Carpenter and Kenward, 2013, p. 130). When these terms involve partially observed variables, the solution consists again in using substantive model compatible functions.

When using functions for random cluster-specific covariance matrices, users should note that this specifies an inverse-Wishart distribution matrix for the level-1 covariance matrices across the level-2 units. Our simulations (Quartagno and Carpenter, 2016) suggest when this assumption is not appropriate there will be some (usually immaterial) loss of efficiency. In principle, **jomo** could be extended to incorporate other distributions.

All the illustrated functions make use of either Gibbs or Metropolis-Hastings sampling; however, other sampling algorithms such as Hamiltonian Monte-Carlo may provide interesting alternatives in the future.

Future updates and additions to the package will be advertised on `www.missingdata.org.uk`, together with an up-to-date list of publications related to the package. We hope the package is useful to readers and welcome their feedback.

Sources of funding

Matteo Quartagno 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.
James Carpenter is supported by the MRC grant MC_UU_12023/21

Aknowledgements

The authors of the package would like to thank Christopher Charlton and Professor Harvey Goldstein from Bristol University for their help in creating the package. We would like to thank Alexander Robitzsch, Vincent Audigier, Anower Hossain, Manuel Gomes, Nicole Erler and all the other people that found bugs and imperfections in the code.

Bibliography

P. Mortimore, P. Sammons, L. Stoll, D. Lewis, and R. Ecob. *School Matters.* Wells: Open Books., 1988. [p208]

R. R. Andridge. Quantifying the Impact of Fixed Effects Modeling of Clusters in Multiple Imputation for Cluster Randomized Trials. *Biom J*, 53(1):57–74, 2011. URL https://doi.org/10.1002/bimj.201000140. [p214]

V. Audigier, I. R. White, S. Jolani, T. P. A. Debray, M. Quartagno, J. Carpenter, S. van Buuren, and M. Resche-Rigon. Multiple imputation for multilevel data with continuous and binary variables. *Statist. Sci.*, 33(2):160–183, 2018. URL https://doi.org/10.1214/18-sts646. [p224, 225]

F. Bloos, H. Ruddel, D. Thomas-Ruddel, D. Schwarzkopf, C. Pausch, S. Harbarth, T. Schreiber, M. Grundling, J. Marshall, P. Simon, M. M. Levy, M. Weiss, A. Weyland, H. Gerlach, T. Schurholz, C. Engel, C. Matthaus-Kramer, C. Scheer, F. Bach, R. Riessen, B. Poidinger, K. Dey, N. Weiler, A. Meier-Hellmann, H. H. Haberle, G. Wobker, U. X. Kaisers, and K. Reinhart. Effect of a Multifaceted Educational Intervention for Anti-Infectious Measures on Sepsis Mortality: a Cluster Randomized Trial. *Intensive Care Med*, 43(11):1602–1612, 2017. URL https://doi.org/10.1177/1094428117703686. [p224]

C. Brown. *Dummies: Create Dummy/Indicator Variables Flexibly and Efficiently*, 2012. URL http://CRAN.R-project.org/package=dummies. R package version 1.5.6. [p212]

J. R. Carpenter and M. G. Kenward. *Multiple Imputation and Its Application*. John Wiley & Sons, 2013. ISBN: 978-0-470-74052-1. [p205, 206, 207, 208, 212, 218, 225]

J. R. Carpenter, H. Goldstein, and M. G. Kenward. Realcom-impute software for multilevel multiple imputation with mixed response types. *Journal of Statistical Software.*, 45(5):1–14, 2011. [p205]

G. Casella and R. L. Berger. *Statistical Inference*. Duxbury Press, 2nd edition, 2002. [p224]

J. Drechsler. Multiple imputation of multilevel missing data—rigor versus simplicity. *Journal of Educational and Behavioral Statistics*, 40(1):69–95, 2015. URL https://doi.org/10.3102/1076998614563393. [p225]

C. K. Enders, T. Hayes, and H. Du. A comparison of multilevel imputation schemes for random coefficient models: Fully conditional specification and joint model imputation with random covariance matrices. *Multivariate Behavioral Research*, 53(5):695–713, 2018. URL https://doi.org/10.1080/00273171.2018.1477040. PMID: 30693802. [p225]

H. Goldstein, J. R. Carpenter, M. G. Kenward, and K. A. Levin. Multilevel models with multivariate mixed response types. *Statistical Modelling.*, 9(3):173–197, 2009. URL https://doi.org/10.1177/1471082X0800900301. [p206, 212]

H. Goldstein, J. R. Carpenter, and W. J. Browne. Fitting multilevel multivariate models with missing data in responses and covariates that may include interactions and non-linear terms. *Journal of the Royal Statistical Society A*, 177(2):553–564, 2014. URL https://doi.org/10.1111/rssa.12022. DOI: 10.1111/rssa.12022. [p225]

S. Grund, O. Lüdtke, and A. Robitzsch. Multiple Imputation of Missing Covariate Values in Multilevel Models with Random Slopes: a Cautionary Note. *Behav Res Methods*, 48(2):640–649, 2016a. URL https://doi.org/10.3758/s13428-015-0590-3. [p225]

S. Grund, O. Lüdtke, and A. Robitzsch. Multiple imputation of multilevel missing data: An introduction to the r package pan. *SAGE Open*, 6(4), 2016b. URL https://doi.org/10.1177/2158244016668220. [p208]

S. Grund, A. Robitzsch, and O. Lüdtke. *Mitml: Tools for Multiple Imputation in Multilevel Modeling*, 2016c. URL https://CRAN.R-project.org/package=mitml. R package version 0.3-0. [p222]

S. Grund, O. Lüdtke, and A. Robitzsch. Multiple imputation of missing data at level 2: A comparison of fully conditional and joint modeling in multilevel designs. *Journal of Educational and Behavioral Statistics*, 2018a. URL https://doi.org/10.3102/1076998617738087. [p224]

S. Grund, O. Lüdtke, and A. Robitzsch. Multiple imputation of missing data for multilevel models: Simulations and recommendations. *Organizational Research Methods*, 21(1):111–149, 2018b. URL https://doi.org/10.1177/1094428117703686. [p224]

S. Grund, O. Lüdtke, and A. Robitzsch. Missing data in multilevel research. In S. E. Humphrey and J. M. LeBreton, editors, *Handbook for Multilevel Theory, Measurement, and Analysis*. American Psychological Association, Washington, DC, 2018c. ISBN 978-1-4338-3001-3. [p224]

T. Harding, F. Tusell, and J. L. Schafer. *Cat: Analysis of Categorical-Variable Datasets with Missing Values*, 2012. URL http://CRAN.R-project.org/package=cat. R package version 0.0-6.5. [p205]

A. Hossain, K. Diaz-Ordaz, and J. W. Bartlett. Missing continuous outcomes under covariate dependent missingness in cluster randomised trials. *Statistical Methods in Medical Research*, 26(3):1543–1562, 2017a. URL https://doi.org/10.1177/0962280216648357. PMID: 27177885. [p224]

A. Hossain, K. DiazOrdaz, and J. W. Bartlett. Missing binary outcomes under covariate-dependent missingness in cluster randomised trials. *Statistics in Medicine*, 36(19):3092–3109, 2017b. URL https://doi.org/10.1002/sim.7334. [p224]

R. J. A. Little and D. B. Rubin. *Bayes and Multiple Imputation*, chapter 10, pages 200–220. John Wiley & Sons, 2002. ISBN 9781119013563. URL https://doi.org/10.1002/9781119013563.ch10. [p205]

O. Lüdtke, A. Robitzsch, and S. Grund. Multiple Imputation of Missing Data in Multilevel Designs: A Comparison of Different Strategies. *Psychol Methods*, 22(1):141–165, 2017. URL https://doi.org/10.1037/met0000096. [p214, 225]

F. Meinfelder. *BaBooN: Bayesian Bootstrap Predictive Mean Matching - Multiple and Single Imputation for Discrete Data*, 2011. URL http://CRAN.R-project.org/package=BaBooN. R package version 0.1-6. [p212]

X.-L. Meng. Multiple-imputation inferences with uncongenial sources of input. *Statistical Science*, 9(4): 538–558, 1994. URL http://doi.org/10.1214/ss/1177010269. [p206]

A. A. Novo and J. L. Schafer. *Norm: Analysis of Multivariate Normal Datasets with Missing Values*, 2013. URL http://CRAN.R-project.org/package=norm. R package version 1.0-9.5. [p205]

I. Olkin and R. F. Tate. Multivariate correlation models with mixed discrete and continuous variables. *The Annals of Mathematical Statistics*, 32(2):448–465, 1961. URL http://doi.org/10.1214/aoms/1177705052. [p205]

J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2017. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-131. [p222]

M. Quartagno and J. R. Carpenter. Multiple imputation for ipd meta-analysis: Allowing for heterogeneity and studies with missing covariates. *Statistics in Medicine*, 35(17):2938–2954, 2016. ISSN 1097-0258. URL https://doi.org/10.1002/sim.6837. [p207, 208, 214, 217, 224, 225]

M. Quartagno and J. R. Carpenter. Multilevel multiple imputation in presence of interactions, nonlinearities and random slopes. In *Studies in Theoretical and Applied Statistics - SIS2018 - 49th Meeting of the Italian Statistical Society, Palermo 20-22 June 2018*. Springer-Verlag, 2018. [p225]

M. Quartagno and J. R. Carpenter. Multiple imputation for discrete data: An evaluation of the joint latent normal model. *Biometrical Journal*, 61(4):1003–1019, 2019. URL https://doi.org/10.1002/bimj.201800222. [p214, 224]

J. Rasbash, F. Steele, W. J. Browne, and H. Goldstein. A user's guide to mlwin, v3.00. *Centre for Multilevel Modelling, University of Bristol.*, 2017. [p208]

D. B. Rubin. Inference and missing data. *Biometrika.*, 63(3):581–592, 1976. URL https://doi.org/10.1093/biomet/63.3.581. [p205]

D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys.* John Wiley & Sons, New York, 1987. [p205]

J. L. Schafer. *Analysis of Incomplete Multivariate Data.* Chapman & Hall., 1997. ISBN 0-412-04061-1. [p205]

J. L. Schafer. *Mix: Estimation/Multiple Imputation for Mixed Categorical and Continuous Data*, 2010. URL http://CRAN.R-project.org/package=mix. R package version 1.0-8. [p205]

J. L. Schafer and M. K. Olsen. Multiple imputation for multivariate missing-data problems: A data analyst's perspective. *Multivariate Behavioral Research*, 33(4), 2000. URL https://doi.org/10.1207/s15327906mbr3304_5. [p205]

J. L. Schafer and R. M. Yucel. Computational strategies for multivariate linear mixed-effects models with missing values. *Journal of Computational and Graphical Statistics*, 11(2):437–457, 2002. URL https://doi.org/10.1198/106186002760180608. [p205]

S. R. Seaman and R. H. Keogh. Handling missing data in matched case-control studies using multiple imputation. *Biometrics*, 71(4):1150–1159, 2015. ISSN 1541-0420. URL https://doi.org/10.1111/biom.12358. [p224]

M. A. Tanner and W. H. Wong. The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82(398):528–540, 1987. ISSN 01621459. URL http://www.jstor.org/stable/2289457. [p206]

R. M. Yucel. Random-covariances and mixed-effects models for imputing multivariate multilevel continuous data. *Statistical Modelling.*, 11(4):351–370, 2011. URL https://doi.org/10.1177/1471082X1001100404. [p205, 207, 214, 217]

Z. Zhang, R. Parker, C. Charlton, G. Leckie, and W. Browne. R2mlwin: A package to run mlwin from within r. *Journal of Statistical Software, Articles*, 72(10):1–43, 2016. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v072.i10. [p208]

J. H. Zhao and J. L. Schafer. *Pan: Multiple Imputation for Multivariate Panel or Clustered Data*. R Foundation for Statistical Computing, 2013. R Package, version 0.9. [p205]

Matteo Quartagno
MRC Clinical Trials Unit at UCL
90 High Holborn, London WC1V 6LC
United Kingdom
m.quartagno@ucl.ac.uk

Simon Grund
IPN - Leibniz Institute for Science and Mathematics Education at Kiel University
Olshausenstraße 62, D-24118 Kiel
Germany
grund@ipn.uni-kiel.de

James Carpenter
London School of Hygiene and Tropical Medicine
Keppel Street, Bloomsbury, London WC1E 7HT
United Kingdom
james.carpenter@lshtm.ac.uk

Time Series Forecasting with KNN in R: the tsfknn Package

by Francisco Martínez, María P. Frías, Francisco Charte and Antonio J. Rivera

Abstract In this paper the **tsfknn** package for time series forecasting using *k*-nearest neighbor regression is described. This package allows users to specify a KNN model and to generate its forecasts. The user can choose among different multi-step ahead strategies and among different functions to aggregate the targets of the nearest neighbors. It is also possible to assess the forecast accuracy of the KNN model.

Introduction

Time series forecasting has been performed traditionally using statistical methods such as ARIMA models or exponential smoothing (Hyndman and Athanasopoulos, 2014). However, the last decades have witnessed the use of computational intelligence techniques to forecast time series. Although artificial neural networks is the most prominent machine learning technique used in time series forecasting (Zhang et al., 1998), other approaches, such as Gaussian Processes (Andrawis et al., 2011) or KNN (Martínez et al., 2017), have also been applied. Compared with classical statistical models, computational intelligence methods exhibit interesting features, such as their nonlinearity or the lack of an underlying model, that is, they are non-parametric.

Statistical methodologies for time series forecasting are present in R as excellent packages. For example, the **forecast** package (Hyndman and Khandakar, 2008) includes implementations of ARIMA, exponential smoothing, the Theta method (Hyndman and Billah, 2003) or basic techniques that can be used as benchmark methods—such as the random walk approach. On the other hand, although a great variety of computational intelligence approaches for regression are available in R, such us the **caret** package (Kuhn, 2008), these approaches cannot be directly applied to time series forecasting. Fortunately, some new packages are filling this gap. For example, the **nnfor** package (Kourentzes, 2017) or the `nnetar` function from the **forecast** package allow users to predict time series using artificial neural networks.

KNN is a very popular algorithm used in classification and regression (Wu et al., 2007). This algorithm simply stores a collection of examples. In regression, each example consists of a vector of features describing the example and its associated numeric target value. Given a new example, KNN finds its k most similar examples, called nearest neighbors, according to a distance metric such as the Euclidean distance, and predicts its value as an aggregation of the target values associated with its nearest neighbors. In this paper we describe the **tsfknn** R package for univariate time series forecasting using KNN regression.

The rest of the paper is organized as follows. Firstly, we explain how KNN regression can be applied in a time series forecasting context using the **tsfknn** package. Next, the different multi-step ahead strategies implemented in our package are explained. Some additional features of the package related to how the KNN model is specified are also discussed. The last sections explain how to assess the forecast accuracy of a model and compare the package with other R packages based on machine learning approaches.

Time series forecasting with KNN regression

In this section we first explain how KNN regression can be applied to forecast time series. Next, we describe how the **tsfknn** package can be used to forecast a time series.

As described above, KNN regression simply holds a collection of training instances. The i-th training instance consists of a vector of n features: $(f_1^i, f_2^i, \ldots, f_n^i)$, describing the instance and an associated target vector of m attributes: $(t_1^i, t_2^i, \ldots, t_m^i)$. Given a new instance, whose features are known—$(q_1, q_2, \ldots, q_n)$—but whose target is unknown, the features of the new instance are used to find its k most similar training instances according to the vectors of features and a similarity or distance metric. For example, assuming that the similarity metric is the Euclidean distance, the distance between the new instance and the i-th training instance is computed as follows:

$$\sqrt{\sum_{x=1}^{n} \left(f_x^i - q_x\right)^2} \tag{1}$$

The k training instances that are closest to the new instance are considered their k most similar instances or k nearest neighbors. KNN is based on learning by analogy. Given a new instance, we think that the targets of its nearest neighbors are probably similar to its unknown target. This way, the targets of the nearest neighbors are aggregated to predict the target of the new instance. For example, assuming that the targets or the k nearest neighbors are the vectors: $t^1, t^2, ..., t^k$, they can be averaged to predict the target of the new instance as:

$$\sum_{i=1}^{k} \frac{t^i}{k} \tag{2}$$

In short, KNN stores a collection of training instances described by n features. Each training instance represents a point in an n-dimensional space. Given a new instance, KNN finds its k closest instances in the n-dimensional space in the hope that their targets are similar to its unknown target.

Now, let us see how KNN can be applied to time series forecasting. In this case, the target associated with a training instance is a collection of values of the time series and the features describing the instance are lagged values of the target—that is, we have an autoregressive model. For example, let us start with a monthly time series containing 132 observations, i.e., 11 years: $t = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, ..., x_{132}\}$ and assume that we want to predict its next future month. Because we are going to predict only one value the target of a training instance is a value of the time series. Let us also assume that we decide that the features describing a target are its first twelve lagged values in the time series—that we will denote as lags 1:12. Therefore, the training instances or examples associated with the time series t are shown in Table 1. Now, let us see which is the new instance used to predict the next future value of the time series. Because we are using lags 1 to 12 as feature vector, the feature vector associated with the next future point is vector $(x_{121}, x_{122}, ..., x_{132})$, which is compounded of the last twelve values of the time series. If, for example, k is equal to 2 the 2-nearest neighbors of the new instance are found and their targets will be aggregated to predict the next future month. The rationale behind the use of KNN for time series forecasting is that a time series can contain repetitive patterns. Given the last pattern of a time series we look for similar patterns in the past in the hope that their subsequent patterns will be similar to the future values of the time series.

Table 1: Training examples for time series $t = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, ..., x_{132}\}$ for one-step ahead forecasting and lags 1:12 as feature vector

Features	Target
$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}$	x_{13}
$x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}$	x_{14}
$x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}$	x_{15}
...	...
$x_{120}, x_{121}, x_{122}, x_{123}, x_{124}, x_{125}, x_{126}, x_{127}, x_{128}, x_{129}, x_{130}, x_{131}$	x_{132}

Let us see now how the **tsfknn** package can be used to forecast a time series. In our package, you can consult the training examples associated with a KNN model with the `knn_examples` function:

```
> timeS <- window(nottem, end = c(1930, 12))
> pred <- knn_forecasting(timeS, h = 1, lags = 1:12, k = 2)
> head(knn_examples(pred))
     Lag12 Lag11 Lag10 Lag9 Lag8 Lag7 Lag6 Lag5 Lag4 Lag3 Lag2 Lag1   H1
[1,]  40.6  40.8  44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8 44.2
[2,]  40.8  44.4  46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8 44.2 39.8
[3,]  44.4  46.7  54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8 44.2 39.8 45.1
[4,]  46.7  54.1  58.5 57.7 56.4 54.3 50.5 42.9 39.8 44.2 39.8 45.1 47.0
[5,]  54.1  58.5  57.7 56.4 54.3 50.5 42.9 39.8 44.2 39.8 45.1 47.0 54.1
[6,]  58.5  57.7  56.4 54.3 50.5 42.9 39.8 44.2 39.8 45.1 47.0 54.1 58.7
```

Before consulting the training examples with `knn_examples`, you have to build the model. This is done with the function `knn_forecasting` that builds a model associated with a time series and uses the

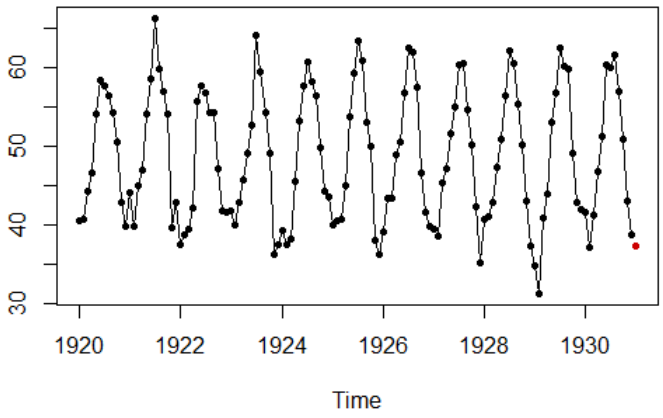


Figure 1: Result of plotting a prediction.

model to predict the future values of the time series. Let us see the main arguments of this function:

`timeS` : the time series to be forecast.

`h` : the forecasting horizon, that is, the number of future values to be predicted.

`lags` : an integer vector indicating the lagged values of the target used as its describing features in the examples—for instance, 1:12 means that lagged values 1 to 12 should be used.

`k` : the number of nearest neighbors used by the KNN model.

`knn_forecasting` is very handy because, as commented above, it builds the KNN model and then uses the model to predict the time series. This function returns a `"knnForecast"` object with information of the model and its prediction. As we have seen above, you can use the function `knn_examples` to see the training examples associated with the model. You can also consult the prediction or plot it through the `"knnForecast"` object:

```
> pred$prediction
      Jan
1931 37.4
> plot(pred)
```

Figure 1 shows the result of plotting the prediction. It is also possible to see how the prediction was made. That is, you can consult the new instance whose target was predicted and its nearest neighbors. This information is obtained with the `nearest_neighbors` function applied to a `"knnForecast"` object:

```
> nearest_neighbors(pred)
$`instance`
Lag 12 Lag 11 Lag 10  Lag 9  Lag 8  Lag 7  Lag 6  Lag 5  Lag 4  Lag 3  Lag 2  Lag 1
  41.6   37.1   41.2   46.9   51.2   60.4   60.1   61.6   57.0   50.9   43.0   38.8

$nneighbors
  Lag 12 Lag 11 Lag 10 Lag 9 Lag 8 Lag 7 Lag 6 Lag 5 Lag 4 Lag 3 Lag 2 Lag 1   H1
1   40.8   41.1   42.8  47.3  50.9  56.4  62.2  60.5  55.4  50.2  43.0  37.3 34.8
2   39.3   37.5   38.3  45.5  53.2  57.7  60.8  58.2  56.4  49.8  44.4  43.6 40.0
```

Because we have used lags 1:12 as features, the features associated with the next future value of the time series are the last twelve values of the time series. The targets of the two most similar examples or nearest neighbors are 34.8 and 40. Their average is the prediction: 37.4. A nice plot including the new instance, its nearest neighbors and the prediction can be obtained as follows:

```
> library(ggplot2)
> autoplot(pred, highlight = "neighbors", faceting = FALSE)
```

The result of executing this code snippet is shown in Figure 2. To recapitulate, in order to specify a KNN model you have to set:

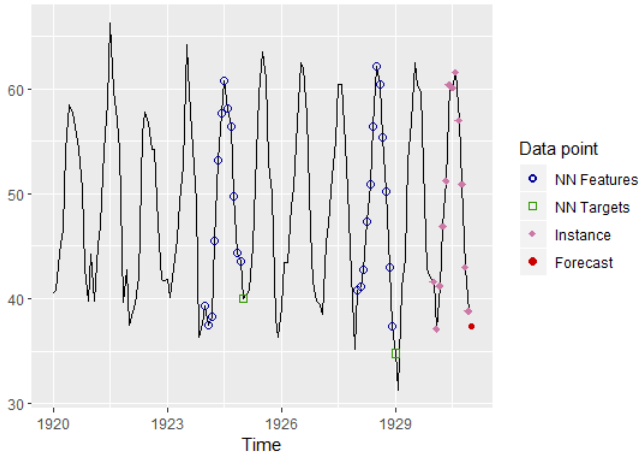


Figure 2: Plotting the instance, its nearest neighbors and the prediction.

- The lags used to build the KNN examples. They determine the lagged values used as features or autoregressive explanatory variables.
- k, that is, the number of nearest neighbors used in the prediction.
- The forecasting horizon, that is, the number of future points to predict.

Multi-step ahead strategies

In the previous section we have seen an example of one-step ahead prediction with KNN. Nonetheless, it is very common to forecast more than one value into the future. To this end, a multi-step ahead strategy has to be chosen (Ben Taieb et al., 2012). Our package implements two common strategies: the MIMO—Multiple Input Multiple Output—approach and the recursive or iterative approach. It must be noted that when only one future value is predicted both strategies are equivalent. In the next subsections these strategies are explained, together with examples of how they can be used in our package.

The MIMO strategy

The Multiple Input Multiple Output strategy is commonly applied with KNN and it is characterized by the use of a vector of target values. The length of this vector is equal to the number of periods to be forecast. For example, let us assume that we are working with a time series giving the monthly totals of car drivers killed in Great Britain and we want to forecast the number of deaths for the next 12 months. In this situation, a good choice for the lags used as features would be 1:12, i.e., the totals of car drivers killed in the previous 12 months—an explanation about why lags 1:12 are a good choice is given in the section about default parameters. If the MIMO strategy is chosen, then a training example consists of:

- a target vector with the number of deaths of 12 consecutive months and
- a feature vector with the number of deaths in the previous 12 consecutive months—just before the 12 months of the target vector.

The new instance would be the number of car drivers killed in the last 12 months of the time series. This way, we would look for the number of deaths most similar to the last 12 months in the time series and we would predict an aggregation of their subsequent 12 months. In the following example we predict the next 12 months using the MIMO strategy:

```
> timeS <- window(UKDriverDeaths, end = c(1979, 12))
> pred <- knn_forecasting(timeS, h = 12, lags = 1:12, k = 2, msas = "MIMO")
> autoplot(pred, highlight = "neighbors", faceting = FALSE)
```

The forecast for the next 12 months can be seen in Figure 3. The last 12 values of the time series are the features of the new instance whose target has to be predicted. The two sequences of 12 consecutive values most similar to this instance are found—in blue—and their subsequent 12 values—in green—are averaged to obtain the prediction—in red.

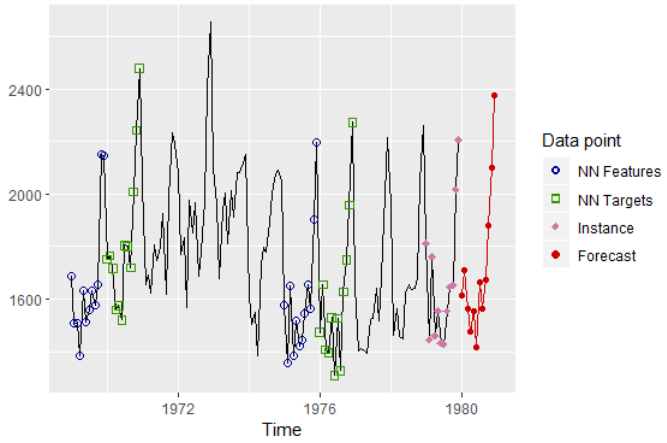


Figure 3: Applying the MIMO strategy to forecast the next 12 months.

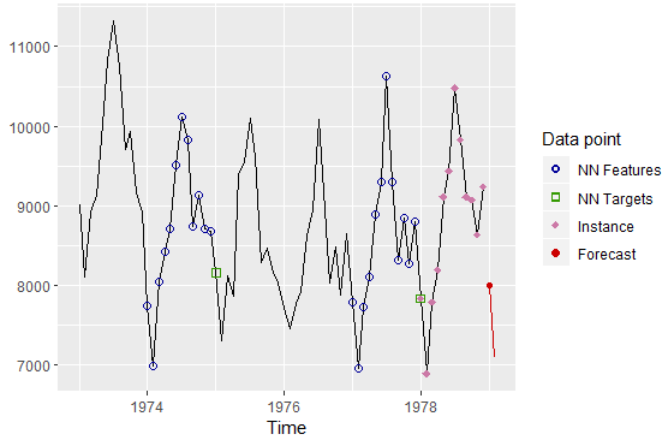


Figure 4: Applying the recursive strategy to forecast horizon 1.

The recursive strategy

The recursive or iterative strategy is the approach used by ARIMA or exponential smoothing to forecast several periods. In this strategy a model that only forecasts one-step ahead is used. Therefore, the model is applied iteratively to forecast all the future periods. When historical observations to be used as features of the new instance are unavailable, previous predictions are used instead.

Because the recursive strategy uses a one-step ahead model, this means that, in the case of KNN, the target of a training example only contains one value. For instance, let us see how the recursive strategy works with the following example in which the next two future months of a monthly time series are predicted:

```
> pred <- knn_forecasting(USAccDeaths, h = 2, lags = 1:12, k = 2, msas = "recursive")
> autoplot(pred, highlight = "neighbors")
```

In this example we have used lags 1:12 to specify the features of an example. To predict the first future point the last 12 values of the time series are used as “its features”—see Figure 4. To predict the second future point "its features" are the last eleven values of the time series and the prediction for the first future point—see Figure 5.

Setting the KNN parameters

In this section several additional features of our package related to model selection are described. In order to select a KNN model the following parameters have to be chosen:

- The distance function used to find the nearest neighbors.
- The combination function used to aggregate the targets of the nearest neighbors.

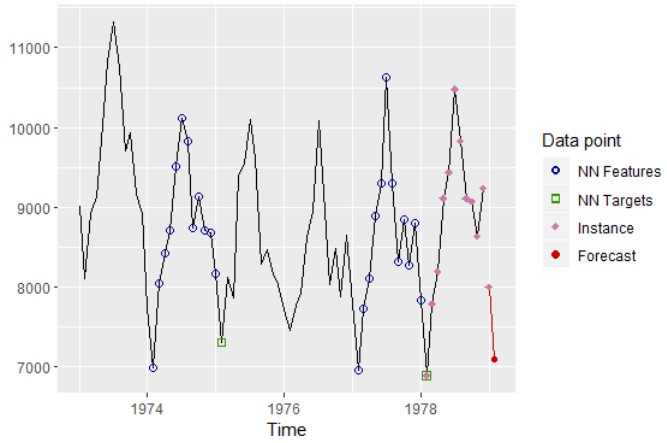


Figure 5: Applying the recursive strategy to forecast horizon 2: the previous prediction is used as feature.

- The number of nearest neighbors, that is, the k parameter.
- The lags used as autoregressive explanatory variables, that is, the features describing the training examples.
- The multi-step ahead strategy.

In the following subsections some features related to setting the KNN parameters are explained.

Distance and combination function

Our package uses the Euclidean distance to find the nearest neighbors, although we can implement other distance metrics in the future.

Regarding the combination function, the targets of the k nearest neighbors are averaged by default. However, it is possible to combine the targets using other aggregation functions. Currently, our package allows users to choose among the mean, the median and a weighted combination. In order to select the combination function the `cb` parameter of the `knn_forecasting` function has to be used.

Next, we explain how the weighted combination is computed. The goal is to give more weight to the closer neighbors. Let us denote as d_i and t_i the distance between the i-th nearest neighbor and the the new instance and the target of the i-th nearest neighbor respectively. Then, we define $w_i = 1/d_i^2$ to be the reciprocal of the squared distance to the i-th nearest neighbor and the prediction is computed as the following weighted combination of the targets:

$$\frac{\sum_{i=1}^{k} w_i t_i}{\sum_{i=1}^{k} w_i} \tag{3}$$

This scheme fails when the distance to a training example is 0. In this special case, the target of this training example, whose features are identical to the new instance, is selected as the prediction.

An ensemble of several models with different k parameters

In order to specify a KNN model the k parameter has to be chosen. If k is too small, then the prediction can more easily be affected by noise. On the other hand, if k is too large, then we are using examples far apart from the new instance. Several strategies can be used to choose the k parameter. A first, fast, straightforward solution is to use some heuristic—it is recommended setting k to the square root of the number of training examples. Another approach is to select k using an optimization tool on a validation set. k should minimize a forecast accuracy measure such as MAPE (Hyndman and Koehler, 2006). It should be noted that the optimization strategy is very time-consuming.

A third strategy explored in (Martínez et al., 2017) is to use several KNN models with different k values. Each KNN model generates its forecasts and the forecasts of the different models are averaged to produce the final forecast. This strategy is based on the success of model combination in time series

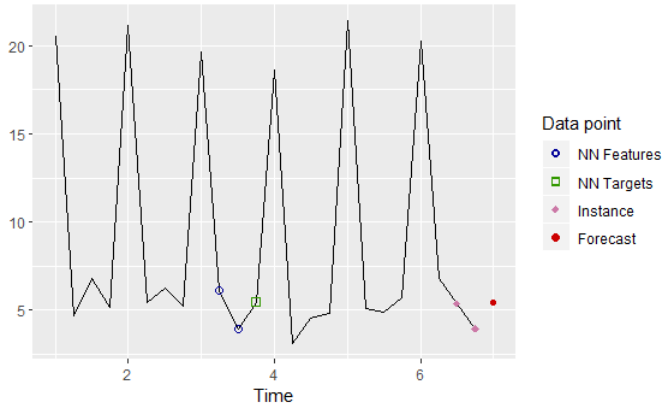


Figure 6: Quarterly time series with a strong seasonal pattern and lags 1:2.

forecasting (Hibon and Evgeniou, 2005). In this way, the use of a time-consuming optimization tool is avoided and the forecasts are not based on a unique k value. In our package you can use this strategy specifying a vector of k values:

```
> pred <- knn_forecasting(ldeaths, h = 12, lags = 1:12, k = c(2, 4))
> pred$prediction
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug      Sep
1980 2865.375 2866.250 2728.875 2189.000 1816.000 1625.875 1526.250 1404.250 1354.000
          Oct      Nov      Dec
1980 1541.250 1699.250 2198.750
```

In this example, two KNN models with k equal to 2 and 4 respectively are built. The forecast is the average of the forecasts produced by both models.

It also must be noted that if the targets are combined using weights, then the choice of the k parameter is less important because neighbors lose weight as they move away from the new instance. When the weighted combination is chosen, it even makes sense to set k to the number of training examples, as generalized regression neural networks do (Yan, 2012).

Default parameters

Sometimes a great number of time series have to be forecast. In that situation, an automatic way of generating the forecasts is very useful. In this sense, our package is able to use sensible default parameters. If the user only specifies the time series and the forecasting horizon the KNN parameters are selected as follows:

- As multi-step ahead strategy MIMO is chosen.
- The combination function used to aggregate the targets is the mean.
- k is selected as a combination of three models using 3, 5 and 7 nearest neighbors respectively.
- In order to select the autoregressive lags the following strategy is used. If the time series is seasonal and the length of the seasonal period is m, then lags 1:m are used. For example, for quarterly data lags 1:4 are selected and for monthly data lags 1:12. This way, seasonal patterns can be captured more easily. Let us see why—see also (Martínez et al., 2018). In Figure 6 an artificial quarterly time series with a strong seasonal pattern is shown. The first quarter has a mean level higher than the other quarters, which have a similar level. The autoregressive lags are lags 1:2 and we want to generate a one-step ahead forecast using only the nearest neighbor. These autoregressive lags can lead to an unsuitable forecast. As can be seen in the figure, the nearest neighbor has a fourth quarter as its target when a first quarter value is going to be predicted. In Figure 7 lags 1:4 are used and the target associated with the nearest neighbor is a first quarter value. If the time series is not seasonal, for example yearly data, then the lags with significant autocorrelation in the partial autocorrelation function (PACF) are selected. Although the PACF only tests for linear relationships, experience has shown us that this is an effective way of selecting input variables (Martínez et al., 2017). If none of the previous two conditions are met, then lags 1:5 are used. Notice that this way of selecting the autoregressive lags is quite fast.

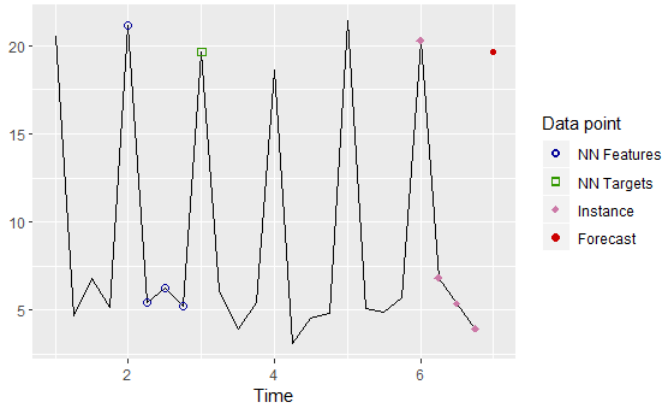


Figure 7: Quarterly time series with a strong seasonal pattern and lags 1:4.

Evaluating forecast accuracy

Once a model has been built, it is natural to want to assess its forecast accuracy. In the **tsfknn** package this is done with the `rolling_origin` function. This function uses the classical approach of dividing a dataset into training and test sets. In time series forecasting the test set consists of the last observations of the time series. For example:

```
> pred <- knn_forecasting(ldeaths, h = 12, lags = 1:12, k = 2)
> ro <- rolling_origin(pred, h = 6, rolling = FALSE)
```

As mentioned above, `knn_forecasting` builds a KNN model and returns a `"knnForecast"` object with information about the model. The `rolling_origin` function takes a `"knnForecast"` object as its first parameter. From this, information about the time series and the metaparameters of the KNN model is obtained; for example, the autoregressive lags, the number of nearest neighbors or the multi-step ahead strategy. The second parameter of `rolling_origin` is the size of the test set. In the example, the size is 6 and, therefore, the last 6 observations of the time series will be used as test set and the remaining observations as training set. `rolling_origin` returns a `"knnForecastRO"` object with information about the evaluation. For example, the test set, predictions and errors can be consulted:

```
> print(ro$test_sets)
      h=1  h=2  h=3  h=4  h=5  h=6
[1,] 1461 1354 1333 1492 1781 1915
> print(ro$predictions)
        h=1    h=2    h=3  h=4    h=5  h=6
[1,] 1513.5 1363.5 1351.5 1567 1587.5 2392
> print(ro$errors)
       h=1  h=2   h=3 h=4   h=5  h=6
[1,] -52.5 -9.5 -18.5 -75 193.5 -477
```

It is also possible to consult several forecasting accuracy measures about the predictions:

```
> ro$global_accu
      RMSE        MAE       MAPE
213.613748 137.666667   7.747168
```

The forecasting accuracy measures are: root mean square error, mean absolute error and mean absolute percentage error. A plot of the time series and the forecasts can be obtained:

```
> plot(ro)
```

The result of this plot can be seen in Figure 8. In this figure the last six observations of the time series are the test set and the forecasts are the red points.

Evaluation based on a rolling origin

A more sophisticated version of training/test sets is to use a rolling origin evaluation. The idea is as follows. The last n observations of a time series are used as test data. Then n evaluations are performed.

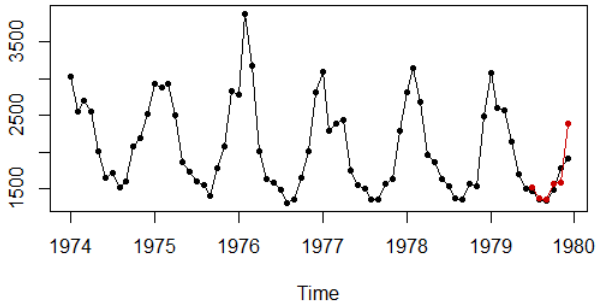


Figure 8: A time series and its forecast for the last 6 observations.

Figure 9: Training (blue) and test (red) sets for rolling origin evaluation with 6 observations.

In the first evaluation the last n observations are the test set and the previous ones the training set. A model is fitted with the training set, the predictions are generated and the errors observed. In the second evaluation the last $n-1$ observations are used as test set, and so on until the last evaluation in which the test set is the last value of the time series. Figure 9 illustrates the different training and test sets for a rolling origin evaluation of the last six values of a time series. As can be observed, the origin of the forecasts rolls forward in time. The rolling origin technique aims to get the most out of the test data in terms of evaluation. For example, with n observations in the test data, n one-step ahead forecasts are generated, $n-1$ two-steps ahead forecasts, and so on. On the other hand, the rolling origin evaluation is a time-consuming task, because several models have to be fitted.

The function `rolling_origin` uses rolling origin evaluation by default:

```
> pred <- knn_forecasting(ldeaths, h = 12, lags = 1:12, k = 2)
> ro <- rolling_origin(pred, h = 6)
> print(ro$test_sets)
      h=1  h=2  h=3  h=4  h=5  h=6
[1,] 1461 1354 1333 1492 1781 1915
[2,] 1354 1333 1492 1781 1915   NA
[3,] 1333 1492 1781 1915   NA   NA
[4,] 1492 1781 1915   NA   NA   NA
[5,] 1781 1915   NA   NA   NA   NA
[6,] 1915   NA   NA   NA   NA   NA
> print(ro$predictions)
        h=1    h=2    h=3    h=4    h=5  h=6
[1,] 1513.5 1363.5 1351.5 1567.0 1587.5 2392
[2,] 1363.5 1351.5 1567.0 1587.5 2392.0   NA
[3,] 1351.5 1567.0 1587.5 2392.0     NA   NA
[4,] 1567.0 1587.5 2392.0     NA     NA   NA
[5,] 1587.5 2392.0     NA     NA     NA   NA
[6,] 2392.0     NA     NA     NA     NA   NA
> print(ro$errors)
        h=1    h=2    h=3    h=4    h=5  h=6
[1,]  -52.5   -9.5  -18.5  -75.0  193.5 -477
[2,]   -9.5  -18.5  -75.0  193.5 -477.0   NA
[3,]  -18.5  -75.0  193.5 -477.0     NA   NA
[4,]  -75.0  193.5 -477.0     NA     NA   NA
```

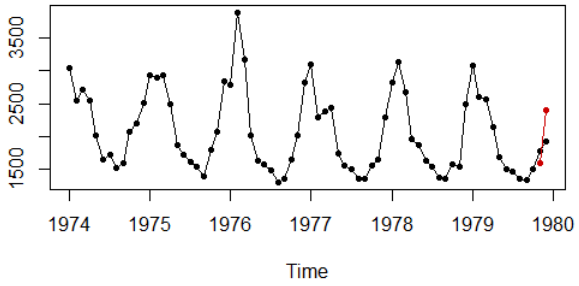


Figure 10: Forecast for the test set with the last two observations.

```
[5,]  193.5 -477.0     NA     NA     NA   NA
[6,] -477.0     NA     NA     NA     NA   NA
```

Each row in the output represents a different evaluation. Now, the `global_accu` field of the `"knnForecastRO"` object stores measures of the accuracy of all the forecasts in all the evaluations:

```
> ro$global_accu
     RMSE       MAE      MAPE
274.19569 202.69048  11.09727
```

The forecast accuracy measures for the different prediction horizons can also be consulted:

```
> ro$h_accu
            h=1        h=2       h=3       h=4       h=5       h=6
RMSE 213.613748 232.821283 260.25877 300.33107 363.98575 477.00000
MAE  137.666667 154.700000 191.00000 248.50000 335.25000 477.00000
MAPE   7.747168   8.577916  10.54699  13.60004  17.88665  24.90862
```

As expected, the errors grow with the increasing horizon. Finally, a plot of any of the different forecasts performed with the rolling origin evaluation can be obtained. For example:

```
> plot(ro, h = 2)
```

produces Figure 10.

A comparison with other time series forecasting packages

In this section our package is compared with other software for univariate time series forecasting in R. In the CRAN Task Views: Time Series Analysis, Econometrics and Finance some models for time series forecasting can be found, from GARCH models to ensembles of models, such us the **forecastHybrid** package (Shaub and Ellis, 2019). Most of the functions found in these packages use statistical models such as ARIMA or exponential smoothing. The **forecast** package is arguably the best package for time series forecasting. It implements all of the models that can be found in the **stats** package and many others such as the Theta method or multilayer perceptron. Furthermore, this package includes additional tools for plotting seasonal time series, doing Box-Cox transformations, etc.

The packages most closely related to our package are those that are based on machine learning approaches. Below, a brief description of these packages is given:

GMDH In this package the GMDH-type neural network algorithms are applied in order to perform short term forecasting for a univariate time series (Dag and Yozgatligil, 2016).

NTS This package contains a function, `NNsetting`, that allows users to create the examples needed to feed a neural network. However, the package does not allow forecasts to be generated directly.

tsDyn Allows users to predict a time series using a multi-layer perceptron with one hidden layer computed using the `nnet` function from the **nnet** package.

nnfor Uses the **neuralnet** package to build multi-layer perceptrons. It is also possible to use extreme learning machines through the `elm` function.

forecast This package contains the `nnetar` function used to forecast a time series using a multi-layer perceptron with a hidden layer.

Table 2 shows a comparison of these packages in terms of the following features:

- Is it possible to use arbitrary autoregressive lags? In some packages, for example, the lags have to be consecutive values.
- Is it possible to generate the forecasts indicating only the time series and the forecasting horizon?
- Does it include the package tools for plotting the forecasts and other information?
- Is rolling origin evaluation implemented?
- Does the package generate prediction intervals for the forecasts?
- Is it possible to include exogenous variables as predictors in the model?

Table 2: Properties of the packages using machine learning approaches.

	GMDH	**tsDyn**	**nnfor**	**forecast**	**tsfknn**
Arbitrary lags	no	no	yes	no	yes
Default parameters	yes	no	yes	yes	yes
Plotting facilities	no	yes	yes	yes	yes
Rolling origin evaluation	no	no	yes	yes	yes
Prediction intervals	yes	no	no	yes	no
Exogenous variables	no	no	yes	yes	no

We have also conducted a comparison of the methods found in these packages based on forecast accuracy and running time. For this purpose, data from the NN3 forecasting competition (Crone et al., 2011) has been used. In this competition 111 monthly time series of industry data were used. The length of the series ranges from 52 to 120 observations and there is also a balanced mix of seasonal and non-seasonal series. As in the NN3 competition, the next 18 future months of every time series have to be predicted. The MAPE has been used to assess the forecast accuracy. Given the forecast F for a NN3 time series with actual values X:

$$\text{MAPE} = \frac{100}{18} \sum_{t=1}^{18} \left| \frac{X_t - F_t}{X_t} \right|$$

Given a certain method, its MAPE is computed for the 111 time series and averaged in order to obtain a global MAPE. This global MAPE appears in the first row of Table 3 for the different methods. In the comparison the package **GMDH** has not been included because at most it allows users to forecast 5-steps ahead. `elm` and `mlp` are functions from the **nnfor** package for computing extreme learning machines and multi-layer perceptrons respectively. `auto.arima` and `ets` are functions belonging to the **forecast** package that implement ARIMA and exponential smoothing. When calling the functions we have specified as few parameters as possible, so that the function selects automatically or by default the value of the parameters. The statistical models have achieved the best results. Among the machine learning approaches our package is the winner.

In the second row of Table 3 the time in seconds needed for fitting the model and generating the forecasts is shown. There are significant differences between the methods, with our package being one of the fastest methods.

Table 3: Properties of the packages using machine learning approaches.

	tsDyn	`elm`	`mlp`	`nnetar`	`auto.arima`	`ets`	**tsfknn**
MAPE	20.73	18.76	20.95	18.38	15.64	15.52	17.06
Time	2	3332	690	15	421	105	4

Functions and methods in the tsfknn package

In this section a succinct description of all the functions and methods in the **tsfknn** package is given. Most of the functions have already been described above. For those functions not mentioned above a brief example of use is given:

`knn_forecasting` given a time series and some metaparameters this function builds a KNN model for forecasting the time series. It also uses this model to make a prediction of the future values of the time series. Information about the model and the prediction is returned in a `"knnForecast"` object.

`knn_examples` shows the examples associated with a KNN model.

`nearest_neighbors` shows the new instance used in a prediction and its nearest neighbors.

`plot and autoplot` plot a time series and its forecast.

`rolling_origin` assesses the forecast accuracy of a KNN model.

`print and summary` show information about a model and its prediction.

`predict` generates new predictions for a given KNN model.

`n_training_examples` indicates the number of examples that a KNN model would have for a given time series and some metaparameters.

Now, a quick example of how to use the functions not explained previously in the paper is given. The methods `print` and `summary` produce the expected result, i.e., they show some information about the model and its prediction:

```
> pred <- knn_forecasting(mdeaths, h = 3)
> print(pred)

Call:  knn_forecasting(timeS = mdeaths, h = 3)

Multiple-Step Ahead Strategy: MIMO
K (number of nearest neighors): 3 models with 3, 5 and 7 neighbors repectively
Autoregressive lags: 1 2 3 4 5 6 7 8 9 10 11 12
Number of examples: 58
Targets are combined using the mean function.
> summary(pred)

Call:  knn_forecasting(timeS = mdeaths, h = 3)

Multiple-Step Ahead Strategy: MIMO
K (number of nearest neighors): 3 models with 3, 5 and 7 neighbors repectively
Autoregressive lags: 1 2 3 4 5 6 7 8 9 10 11 12
Number of examples: 58
Targets are combined using the mean function.
Forecasting horizon: 3
Forecast:
          Jan      Feb      Mar
1980 1990.562 2106.390 1999.143
```

The method `predict` is used to generate new forecasts using a previously fitted model:

```
> pred <- knn_forecasting(mdeaths, h = 3, k = 2, msas = "recursive")
> new_pred <- predict(pred, h = 12)
> print(new_pred$prediction)
        Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
1980 2141.0 2052.0 1894.0 1477.0 1570.5 1216.5 1130.0 1045.5  991.5 1049.5
        Nov    Dec
1980 1144.5 1520.5
```

Finally, `n_training_examples` is a handy function for knowing how many training example would have a model:

```
> n_training_examples(mdeaths, h = 3, lags = 1:12, msas = "MIMO")
[1] 58
```

Summary

There is hardly any package in R for applying computational intelligence regression methods to time series forecasting. In this paper we have presented the **tsfknn** package that allows users to forecast a time series using KNN regression. The interface of the package is quite simple, allowing users to specify a KNN model and to predict a time series using the model. Furthermore, several plots can be generated illustrating how the prediction has been computed.

Acknowledgment

Funds: This work was partially supported by the project TIN2015-68854-R (FEDER Founds) of the Spanish Ministry of Economy and Competitiveness.

We would like to thank the anonymous reviewers whose comments helped improve and clarify this manuscript.

Bibliography

R. R. Andrawis, A. F. Atiya, and H. El-Shishiny. Forecast combinations of computational intelligence and linear models for the NN5 time series forecasting competition. *International Journal of Forecasting*, 27(3):672 – 688, 2011. URL https://doi.org/10.1016/j.ijforecast.2010.09.005. [p229]

S. Ben Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa. A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. *Expert Syst. Appl.*, 39(8):7067–7083, 2012. URL https://doi.org/10.1016/j.eswa.2012.01.039. [p232]

S. F. Crone, M. Hibon, and K. Nikolopoulos. Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting*, 27 (3):635–660, 2011. URL https://doi.org/10.1016/j.ijforecast.2011.04.001. [p239]

O. Dag and C. Yozgatligil. GMDH: An R package for short term forecasting via GMDH-type neural network algorithms. *The R Journal*, 8(1):379–386, 2016. URL https://doi.org/10.32614/RJ-2016-028. [p238]

M. Hibon and T. Evgeniou. To combine or not to combine: Selecting among forecasts and their combinations. *International Journal of Forecasting*, 21(1):15–24, 2005. URL https://doi.org/10.1016/j.ijforecast.2004.05.002. [p235]

R. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 27(1):1–22, 2008. URL https://doi.org/10.18637/jss.v027.i03. [p229]

R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2014. URL https://www.otexts.org/book/fpp. [p229]

R. J. Hyndman and B. Billah. Unmasking the Theta method. *International Journal of Forecasting*, 19(2): 287 – 290, 2003. URL https://doi.org/10.1016/S0169-2070(01)00143-1. [p229]

R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, pages 679–688, 2006. URL https://doi.org/10.1016/j.ijforecast.2006.03.001. [p234]

N. Kourentzes. *Nnfor: Time Series Forecasting with Neural Networks*, 2017. URL https://CRAN.R-project.org/package=nnfor. R package version 0.9.2. [p229]

M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5): 1–26, 2008. URL https://doi.org/10.18637/jss.v028.i05. [p229]

F. Martínez, M. P. Frías, M. D. Pérez, and A. J. Rivera. A methodology for applying k-nearest neighbor to time series forecasting. *Artificial Intelligence Review*, 2017. URL https://doi.org/10.1007/s10462-017-9593-z. [p229, 234, 235]

F. Martínez, M. P. Frías, M. D. Pérez, and A. J. Rivera. Dealing with seasonality by narrowing the training set in time series forecasting with kNN. *Expert Systems with Applications*, 103:38–48, 2018. URL https://doi.org/10.1016/j.eswa.2018.03.005. [p235]

D. Shaub and P. Ellis. *forecastHybrid: Convenient Functions for Ensemble Time Series Forecasts*, 2019. URL https://CRAN.R-project.org/package=forecastHybrid. R package version 4.2.17. [p238]

X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2007. URL https://doi.org/10.1007/s10115-007-0114-2. [p229]

W. Yan. Toward automatic time-series forecasting using neural networks. *IEEE Trans. Neural Netw. Learning Syst.*, 23(7):1028–1039, 2012. URL https://doi.org/10.1109/TNNLS.2012.2198074. [p235]

G. Zhang, B. Eddy Patuwo, and M. Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35–62, 1998. URL https://doi.org/10.1016/S0169-2070(97)00044-7. [p229]

Francisco Martínez
Computer Science Department
University of Jaén
Spain
fmartin@ujaen.es

María P. Frías
Statistics and Operations Research Department
University of Jaén
Spain
mpfrias@ujaen.es

Francisco Charte
Computer Science Department
University of Jaén
Spain
fcharte@ujaen.es

Antonio J. Rivera
Computer Science Department
University of Jaén
Spain
arivera@ujaen.es

rollmatch: An R Package for Rolling Entry Matching

by Kasey Jones, Rob Chew, Allison Witman, Yiyan Liu

Abstract The gold standard of experimental research is the randomized control trial. However, interventions are often implemented without a randomized control group for practical or ethical reasons. Propensity score matching (PSM) is a popular method for minimizing the effects of a randomized experiment from observational data by matching members of a treatment group to similar candidates that did not receive the intervention. Traditional PSM is not designed for studies that enroll participants on a rolling basis and does not provide a solution for interventions in which the baseline and intervention period are undefined in the comparison group. Rolling Entry Matching (REM) is a new matching method that addresses both issues. REM selects comparison members who are similar to intervention members with respect to both static (e.g., race) and dynamic (e.g., health conditions) characteristics. This paper will discuss the key components of REM and introduce the **rollmatch** R package.

Introduction

In experimental studies, scientists design research protocols to empirically test their hypotheses of causal relationships between one or more independent variables and an outcome variable. To isolate the effects of a treatment while mitigating confounding introduced by allocation or selection bias, researchers randomly assign treatments whenever possible. In certain scenarios, it is not always feasible to randomize who receives an intervention, due to cost, coordination, or ethical considerations (Resnik, 2008). This situation is particularly common in disciplines that study human behavior and health, including public policy, international development, medicine, and several social sciences disciplines.

To help address this methodological barrier, researchers have developed quasi-experimental designs to estimate the causal impact of an intervention where subjects are not randomly assigned a treatment (Campbell and Stanley (1996); Meyer (1995); Shadish et al. (2001)). Propensity score matching (Rosenbaum and Rubin (1983); Dehejia and Wahba (2002)) is a popular quasi-experimental method that attempts to mimic randomization by matching units that received the treatment with units having similar or identical observable covariates who did not receive treatment. This matching procedure helps create more meaningful comparisons because variables that might contribute to individuals receiving the treatment are controlled for. A propensity score, "the conditional probability of assignment to a particular treatment given a vector of observed covariates" (Rosenbaum and Rubin, 1983), is used to assess similarities between an individual receiving the treatment and potential matches. Though historically researchers have used logistic or probit regression to model propensity scores, machine learning classification methods are becoming attractive alternatives, due to their ability to deal implicitly with interactions and nonlinearities and empirical evidence supporting their ability to accurately predict outcomes (Lee et al. (2010); Westreich et al. (2010)).

PSM falls under the larger umbrella of causal inference methods and is used within the Neyman-Rubin causal modeling framework (Rubin, 1978). Under this framework, obtaining unbiased causal estimates requires two standard assumptions. First, assignment to treatment must be independent of potential outcomes. And second, we assume that all treated individuals receive the same treatment and treatment of one person does not affect the outcome of another.

Rolling Entry Matching

Traditional propensity score matching designs are cross-sectional in nature, matching on covariates before the intervention and measuring outcomes after the intervention to analyze the effect of a treatment at a specific point in time. While effective in many situations, this approach inherently assumes that covariates do not change in a time window relevant for the analysis, or if they do, that these changes will not also affect the outcome variable. In many areas such as health care and epidemiology, relevant time-varying covariates are not uncommon and cause difficulties for traditional matching approaches. Longitudinal settings can also add complexity when exposures or treatments can vary with time or when the treatment entry date is undefined for the control group (Stuart, 2010).

Rolling entry matching (Witman et al., 2018) is a propensity score matching method designed for longitudinal or panel studies where participants to be treated are enrolled on a rolling basis, a common

practice in health care interventions where delaying treatment may impact patient health. We can use rolling entry matching to retrospectively select comparison group members who are similar to intervention members with respect to both static characteristics (e.g., race) and dynamic characteristics that change over time (e.g., health conditions). Incorporating time-varying characteristics into the matching procedure is important for health care interventions because a participant's health and medicinal utilization often predict entry into an intervention.

REM is also effective when there is no intervention start date for the comparison group. For certain studies, the comparison group never actually receives an intervention. While this is not a problem for PSM methods in a pre and post setting, matching individuals based on when they *could* have started an intervention is complicated in longitudinal settings with non-uniform intervention start dates. REM address both the rolling entry and missing intervention start date issues.

Typical propensity score methods are not designed to handle rolling entry because the baseline period for potential comparison individuals needs to be different for each treatment participant. To illustrate, consider two hypothetical people: (1) Sue, who started taking a prescription (the treatment), and (2) Jan, who is similar to Sue in static characteristics but does not take this medicine. If Sue started her pills in March, we might compare Sue and Jan's data from February. If Sue started her pills in June, we might compare Sue and Jan's data from May. This is done because Jan could have started taking pills in any month. REM helps in making these comparisons by turning a single comparison individual into multiple psuedo-comparison individuals, one for each unique intervention period occurring in the dataset.

Rolling entry matching requires a quasi-panel dataset and is performed in three phases. The quasi-panel dataset should consist of all available data for both treatment and control subjects and should be longitudinal.

1. **Reduce Data:** The quasi-panel dataset is reduced based on two specifications. First, all treatment observations are filtered to observations whose current time period equals the treatments entry period minus some value. For example, if Sue was treated in May and we want to look back 1 time period, we would filter to Sue's data from April. This value is called `lookback`. And second, after filtering treatment observations, we filter the control observations to those who share a time period with any treatment individual. Continuing our example, we would keep all control data with a time value equal to April.

   The `lookback` value has a default value of 1, as researchers usually consider only the time period directly before entering the study (i.e. `lookback = 1`). In certain studies, researchers would want the `lookback` to be greater than one. For example, researchers could find participants that will begin a new diet in 4 weeks. Their health conditions may change between the announcement and the official beginning of the treatment; `lookback` would be set to four.

2. **Calculate Propensity Scores:** Propensity scores are calculated for all data left after the reduction step.

3. **Find Matches:** Individuals are matched based on their propensity scores and entry period through a matching algorithm developed specifically for REM (see Matching algorithm). When a match is created, the control observation is assigned the intervention start date of the treatment observation.

Rolling entry matching is one of several matching methods used to select a comparison group for treatments that occur on a rolling basis, including balance risk set matching (Li et al., 2011), stepwise matching (Yi, 2014), and sequential cohort matching (Seeger et al. (2005); Schneeweiss et al. (2011); Mack et al. (2013)). In addition, inverse probability propensity score weighting methods, such as marginal structural models (Robins et al., 2000), have also been suggested to deal with time-varying covariates. However, despite its importance across a number of different settings, there are few implementations of longitudinal propensity score methods for R. At the time of writing, the only packages that natively support longitudinal propensity score methods are (1) the **CBPS** package, which implements covariate balancing propensity score for longitudinal settings to be used in conjunction with marginal structural models (Imai and Ratkovic, 2015); (2) the **ipw** package, which allows users to estimate marginal structural models; and (3) the **rollmatch** package, which implements rolling entry matching. Of these three, only rollmatch provides an integrated matching approach, as both **CBPS** and **ipw** rely on propensity score weighting.

We now introduce **rollmatch**, an R package for performing rolling entry matching. In particular, we will provide an overview of the main functions in **rollmatch**, a walk-through of the rolling entry matching algorithm, and commentary on relevant parameter choices such as caliper selection.

The rollmatch package

The **rollmatch** package is comprised of three functions.

- `reduce_data()`: Step 1 of REM - Reduces the input panel dataset
- `score_data()`: Step 2 of REM - Calculates propensity scores for the reduced data. This function is not required if users want to develop their own propensity score models
- `rollmatch()`: Step 3 of REM - Performs the matching algorithm and produces output

rollmatch example:

```
library(rollmatch)
data(package = "rollmatch", "rem_synthdata_small")
reduced_data <- reduce_data(data = rem_synthdata_small, treat = "treat",
                            tm = "quarter", entry = "entry_q",
                            id = "indiv_id", lookback = 1)
fm <- as.formula(treat ~ qtr_pmt + yr_pmt + age)
vars <- all.vars(fm)
scored_data <- score_data(reduced_data = reduced_data,
                          model_type = "logistic", match_on = "logit",
                          fm = fm, treat = "treat",
                          tm = "quarter", entry = "entry_q", id = "indiv_id")
output <- rollmatch(scored_data, data=rem_synthdata_small, treat = "treat",
                    tm = "quarter", entry = "entry_q", id = "indiv_id",
                    vars = vars, lookback = 1, alpha = .2,
                    standard_deviation = "average", num_matches = 3,
                    replacement = TRUE)
```

Rolling entry matching: a walkthrough

This section describes the operations performed in **rollmatch** through an illustrative example. Though some of these operations are hidden from the user, understanding the matching algorithm will help troubleshoot potential errors and better inform the selection of parameter values. In addition to discussing steps taken to trim potential matches and calculate propensity scores, special attention is paid to the specifics of the matching algorithm.

Step 1: Trim the treatment data

We begin with a panel dataset that includes individuals who received an intervention at different time periods, as well as other individuals that are being considered for selection into the comparison group. For each individual, we have background variables (e.g., demographics, health conditions, spending habits, etc.) at each time step, an indicator variable for if the individual was treated, a variable specifying the time period of the observation, and a time period variable for when the participant entered the intervention. We let `Treat = 1` indicate an individual who had an intervention and `Treat = 0` indicate someone who did not. Finally, we will let `lookback = 1`.

ID	Treat	Time	Entry	Background Variables			
X	1	1	2	...	...	...	...
X	1	2	2	...	...	...	...
X	1	3	2	...	...	...	...
Y	1	1	2	...	...	...	...
...	...	...	...	...	...	...	...

Table 1: Example dataset of treated observations

In this example, individual *X* has 3 quarters of data and is part of the treatment group. Since participant X entered the treatment in time period 2 and `lookback = 1`, her data from time period 1 will be used for matching with control observations. As REM allows for matching on dynamic

variables that can change over time, matching individual X on observations prior to the intervention provides a clean comparison in which we do not need to worry about the influence of the intervention on the dynamic covariates.

Recall that `lookback` can be written as `entry-time`. Rows that do not match `entry-time = 1` are then dropped.

ID	Treat	Time	Entry	Background Variables			
X	1	1	2	...	...	...	...
Y	1	1	2	...	...	...	...
...	...	...	...	...	...	...	...

Table 2: Dropping treated observations based on `lookback = 1`

Step 2: Trim Control data

Let Table 3 represent the control data.

ID	Treat	Time	Background Variables			
A	0	1	...	...	...	...
A	0	2	...	...	...	...
A	0	3	...	...	...	...
B	0	1	...	...	...	...
...	...	...	...	...	...	...

Table 3: Original control observations

Since rolling entry matching requires that the entry period of any potential comparison observations be equal to the entry period of a treatment observation, we drop all comparison observations that do not share a time period with at least one treatment record. Our example treatment observation data only has individuals that enter the intervention at time periods 2 and 3. Therefore, we only look at control observations whose time is equal to 1 or 2.

ID	Treat	Time	Background Variables			
A	0	1	...	...	...	...
A	0	2	...	...	...	...
B	0	1	...	...	...	...
...	...	...	...	...	...	...

Table 4: Control data after dropping observations

Step 3: Calculate propensity scores and absolute differences for all possible matches

Users are allowed to calculate their own propensity scores to use with the **rollmatch** matching algorithm, or they can use the scoring function provided in the package. If using `score_data()`, users can specify either "logistic" or "probit" regression and the formula for the model (i.e., selecting the covariates to be used). Once a propensity score has been generated for all observations, we look at the absolute difference in scores for all possible matches between control and treatment observations. Recall that in order to be a match, the time period of a control observation must match the time period of a treatment observation. We have provided Table 5 in full so that we can go into detail about the matching algorithm.

Time	Treat ID	Treat Score	Control ID	Control Score	**Difference**
1	X	0.95	A	0.16	**0.79**
1	X	0.95	B	0.42	**0.53**
1	X	0.95	C	0.61	**0.34**
1	X	0.95	D	0.32	**0.63**
1	X	0.95	E	0.15	**0.80**
1	Y	0.03	A	0.16	**0.13**
1	Y	0.03	B	0.42	**0.39**
1	Y	0.03	C	0.61	**0.58**
1	Y	0.03	D	0.32	**0.29**
1	Y	0.03	E	0.15	**0.12**
2	Z	0.65	A	0.63	**0.02**
2	Z	0.65	B	0.26	**0.39**
2	Z	0.65	C	0.05	**0.60**
2	Z	0.65	D	0.57	**0.08**
2	Z	0.65	E	0.43	**0.22**
2	Q	0.11	A	0.63	**0.52**
2	Q	0.11	B	0.26	**0.15**
2	Q	0.11	C	0.05	**0.06**
2	Q	0.11	D	0.57	**0.46**
2	Q	0.11	E	0.43	**0.32**

Table 5: Calculated absolute differences for all matches from Table 2 and Table 4.

Step 4: Trim the Comparison Pool

Caliper

For the data in Table 5, treatment id X has been compared to control ids A, B, C, D, and E. The lowest difference value for these five comparisons is .34, which while being the best match available, may still be too different to provide a high quality match and may bias estimates of the outcome if included (Lunt, 2013). To limit the potential matches, an alpha value between 0 and 1 can be specified. The alpha value is a scaling factor that effects which propensity scores are considered. A value closer to 1 allows for a wider range of propensity scores to be considered, while a value close to 0 provides stricter requirements for matching. The alpha value is multiplied by the pooled standard deviation of the propensity scores; this final value is called the caliper and is used as a cutoff.

Consequently, if an alpha is specified, there is no guarantee that each treatment ID will receive a match. As caliper selection can play a large role in selecting potential matches, we have provided Appendix A: Theorem and Appendix B: Selecting the appropriate pooled standard deviation discussing caliper selection.

Number of Matches

When running `rollmatch()`, the user can specify the maximum number of control matches that should be assigned, when possible, to each treatment observation. If the user sets this to one, and no additional steps are taken, every single treatment observation will be assigned one control observation, regardless of the quality of their best-match (assuming there are enough control observations). However, if the user specifies an alpha value and a caliper is used, there may be some treatment observations that do not receive a match. As the value of alpha decreases, the likelihood that some treatment observations do not have a match will rise. If any treatment observations are not matched, their ids are listed in the output as `ids_not_matched`.

Currently, the user can only guarantee that each treatment observation be assigned at least one match (i.e. by not specifying an alpha). In a future version of **rollmatch**, the user will be able to specify the number of matches to attempt to create (`num_matches`), as well as a minimum number of matches to create. This would ensure that each treatment observation is matched with some number of control individuals, regardless of the alpha selected. It would also allow for other treatment observations to be matched to more observations if enough control individuals are within the caliper.

For simplicity, we will not trim the comparison pool from Table 5 for our example.

Step 5: Assign matches

After the comparison pool has been created and trimmed, treatment and control observations are matched. Rosenbaum and Rubin (1985) used the following matching rules:

1. Randomly order treatment observations
2. For the first treatment subject, based on the comparison pool find all comparison matches for the treated observation whose difference is less than the caliper. If no match exists, match treated observation to control observation with smallest difference
3. From this group, select a match based on the Mahalanobis distance for the background variables
4. Remove the treated and matched observation and repeat steps 2-4 for the next treated observation

There have already been several R packages released that make use of this original algorithm while making modifications to the algorithm to fit the specific goal of the package. Packages such as **MatchIt**, **Matching**, and **optmatch** all offer various matching algorithms for propensity scores.

Rolling entry matching takes a different approach by matching non-participants based on the entry period for which their data is most similar to their matched participant. Whereas other methods like sequential cohort matching (Seeger et al., 2005) start from specific cohorts to begin matching (allowing early cohorts to get the matches that work the best for them without consideration of later cohorts), rolling entry matching considers all periods when matching non-participants to participants. The algorithm for rollmatch must be different because control participants are treated as if they could enter the study at any time. This creates a lot more potential matches per observation. Furthermore, a control observation could best match multiple treatment observations across multiple quarters of entry and there must be logic to handle this scenario.

Matching algorithm

Each treated observation is initially assigned to its best-matching control observation based on the smallest absolute difference between their propensity scores. Recall that the comparison pool only consists of treatment/control pairs that have already been matched on their entry period. As long as the control is not the best-match for another treated observation of a different entry quarter, then the two are matched and the algorithm continues. Recall that any given control individual could have several data entries (one for each quarter they have data available). It is possible that a control observation could be matched to two different treatment observations who entered a study in different time periods. Using Table 5 we have the following matches for iteration one of the algorithm:

Time	Treat ID	Treat Score	Control ID	Control Score	Difference
1	X	0.95	C	0.61	0.34
1	Y	0.03	E	0.15	0.12
2	Q	0.11	C	0.05	0.06
1	Z	0.65	A	0.63	0.02

Table 6: Possible matches for iteration one

Notice that control individual C has been matched to X in time period 1 and matched to Q in time period 2. Since the propensity score difference between Q and C is smaller than the difference between X and C, Q will be matched to C for time period 2, and X will not be matched to any control this round. If X and Q enrolled in the same quarter, then control C would be matched to both treatments if the `replacement` parameter was set to `TRUE`, indicating matching with replacement is desired. `replacement`

Time	Treat ID	Treat Score	Control ID	Control Score	Difference
1	X	.95	C	.61	.34
1	Y	.03	E	.15	.12
1	W	.70	C	.61	.09
2	Q	.11	C	.05	.06

Table 7: Alternative possible matches for iteration one

allows for multiple treatments to be assigned the same control observation if the treatments enrolled in the same quarter. Consider the alternative set of matches in Table 7.

If `replacement` is TRUE and a control individual is matched to multiple treatment IDs for multiple quarters, we take the average difference for all treatment IDs in each quarter to make the final decision. In Table 7, control C is the best match for X and W in time period 1, and the best match for Q in time period 2. The average difference for X and W is .215 and the average difference for Q is simply .06. In this case, Control C will only be matched with treatment Q.

After each iteration of assignment, any matched treatment and control observations are removed from the pool of potential matches and the process is repeated. Once all treatment observations have been assigned the desired number of matches, or there are no more possible matches remaining, the process is complete.

An explaination of caliper selection

Rosenbaum and Rubin used the results of Cochran and Rubin (1973) to conclude that under certain conditions, specific caliper widths could remove a certain percentage of the bias of confounding variables (Rosenbaum and Rubin, 1985). Let σ_1^2 and σ_2^2 be the variances of the logit of the propensity scores (referred to as just variance going forward) for the treated and control groups, and let:

$$\sigma = \sqrt{[(\sigma_1^2 + \sigma_2^2)/2]}. \tag{1}$$

Finally, let our caliper equal $\alpha * \sigma$. According to Rosenbaum and Rubin, at different levels of α, we can remove different levels of bias. Austin (2010) conducted Monte Carlo simulations to verify these findings. We have outlined the reduction in bias in Table 8 . Note that in this case, the variance for the treatment and control groups must be equal.

Alpha	Rosenbaum and Rubin	Austin
.2	99%	At least 99.3%
.6	89%	95.2%-99.6%

Table 8: Expected bias reduction at various α levels

The likelihood that the variance of the two groups being equal is unknown, and although Rosenbaum and Rubin (1985) provided estimates for bias reduction when they are equal, the guidance on selecting a caliper is minimal. We have left the selection of the caliper width up to the user, but we will go into further detail about the two parameters effecting the caliper that are included in **rollmatch**.

The `alpha` parameter must be 0 or greater. At 0, the trimming function is ignored. For all values above 0, the dataset of potential control matches is trimmed based on if the difference between scores is less than $\alpha * \sigma$.

The second decision the user can make is on how sigma is calculated. Both Rosenbaum and Rubin, and Austin use the pooled standard deviation, which we have defined as σ above (Rosenbaum and Rubin (1983); Austin (2010)). Consider the alternative formula for pooled standard deviation for i groups:

$$\sigma = \sqrt{\frac{(n_1 - 1)\sigma_1^2 + (n_2 - 1)\sigma_2^2 + \ldots + (n_i - 1)\sigma_i^2}{(n_1 + n_2 + \ldots + n_i) - i}} \tag{2}$$

Let σ_{f1} be the average pooled standard deviation that our sources have been using so far, and let σ_{f2} be equal to the weighted pooled standard deviation for that we just introduced (for $i = 2$). These two calculations are only equal only under specific conditions (see Appendix A: Theorem). If a dataset has a much larger treatment or control group, or the variances for the two groups propensity scores are vastly different, the weighted pooled standard deviation may do a better job at selecting a cutoff.

Conclusion

We have presented **rollmatch** as an R package for performing rolling entry matching. When observational studies are conducted on a rolling entry basis or when control entry periods do not exist, **rollmatch** is an effective package for finding matches between treated and untreated subjects. The amount of bias introduced by confounding variables can often be reduced by using propensity score matching. However, rolling entry matching furthers this ability by matching treated individuals to control individuals as if they were enrolled at the same time. The parameters and options included in **rollmatch** create a robust and user friendly package. We hope to continue expanding this package as further development of rolling entry matching is completed.

Acknowledgements

rollmatch was made possible by the research conducting on rolling entry matching by Allison Witman, Yiyan Liu, Mahin Manley, and Chris Beadles. The concepts, matching algorithm, and techniques demonstrated in this paper should be credited to them. We would also like to thank Georgiy Bobashev, a Fellow at RTI International. This paper would not have been written without him.

Appendix A: Theorem

Theorem 1. *σ_{f1} is equal to σ_{f2} if and only if $n_1 = n_2$ or $\sigma_1 = \sigma_2$*

Proof. Assume σ_{f1} and σ_{f2} are equal. We will show that this is true only when $n_1 = n_2$ or $\sigma_1 = \sigma_2$.

$$\begin{aligned} \sigma_{f1} &= \sigma_{f2} \\ \sqrt{\frac{\sigma_1^2 + \sigma_2^2}{2}} &= \sqrt{\frac{(n_1-1)\sigma_1^2 + (n_2-1)\sigma_2^2}{n_1+n_2-2}} \end{aligned}$$

Variance's are positive by nature and the number of samples in each group must be greater than 0. We can remove the square root.

$$\begin{aligned} \frac{\sigma_1^2 + \sigma_2^2}{2} &= \frac{(n_1-1)\sigma_1^2 + (n_2-1)\sigma_2^2}{n_1+n_2-2} \\ (\sigma_1^2 + \sigma_2^2)(n_1+n_2-2) &= 2[(n_1-1)*\sigma_1^2 + (n_2-1)*\sigma_2^2] \\ \sigma_1^2*n_1 + \sigma_1^2*n_2 + \sigma_2^2*n_1 + \sigma_2^2*n_2 &= 2\sigma_1^2*n_1 + 2\sigma_2^2*n_2 \\ \sigma_1^2*n_2 + \sigma_2^2*n_1 &= \sigma_1^2*n_1 + \sigma_2^2*n_2 \end{aligned}$$

This can be written two ways.

$$\begin{aligned} \sigma_1^2*(n_2-n_1) &= \sigma_2^2*(n_2-n_1) \\ n_2(\sigma_1^2-\sigma_2^2) &= n_1(\sigma_1^2-\sigma_2^2) \end{aligned}$$

Let's examine these two equations. If $n_1 \neq n_2$, we can divide $(n_2 - n_1)$ from both sides of the first equation, and we end up with $\sigma_1^2 = \sigma_2^2$. This implies $\sigma_1 = \sigma_2$ because variances cannot be negative. Similarly, if $\sigma_1 \neq \sigma_2$, we can divide $(\sigma_1^2 - \sigma_2^2)$ from both sides of the second equation and we find that $n_1 = n_2$. The original equation can only hold if at least one equality holds: $n_1 = n_2$ or $\sigma_1 = \sigma_2$.

□

Appendix B: Selecting the appropriate pooled standard deviation

Selection between σ_{f1} and σ_{f2} is important when the variances of the treatment and control group are not equal. Setting $\alpha = .2$ may not reduce 99% of the bias due to confounding variables if this is true.

Let us examine how different our results are when using different options. We will use the following parameters:

```
formula <- as.formula(treat ~ qtr_pmt + yr_pmt + age)
tm = 'quarter'
entry = 'entry_q'
id = 'indiv_id'
lookback = 1
match_on = 'logit'
model_type = 'logistic'
```

For the smaller synthetic dataset, the variance (of the logit of the propensity score) of our treated group is .891. While the variance of the untreated group is 4.690. In this case, σ_{f1} is equal to 1.658 and σ_{f2} equals 2.141. The original comparison pool had 15,000 treatment and control comparison. Table 9 shows how the alpha value and choice of sigma limit the number of potential matches.

Table 9: Pooled standard deviation comparisons

Alpha	Sigma	Comparions Available
.2	σ_{f1}	414
.2	σ_{f2}	516
.6	σ_{f1}	1044
.6	σ_{f2}	1192
1.0	σ_{f1}	1350
1.0	σ_{f2}	1451

We did not do any simulations of our own to determine how much bias could be reduced when variances are not equal and when the two σ calculations are implemented. Some studies such as Wang et al. (2013) have used σ_{f2} in their calculations. However, when they used only a single treatment group they still assumed equal variances.

To summarize why this is important, if variances among the groups are not equal, the amount of bias reduced at certain levels of alpha will not be the same as what is suggested by Table 8.

Bibliography

P. Austin. Optimal caliper widths for propensity-score matching when estimating differences in means and differences in proportions in observational studies. *Pharmaceutical Statistics*, 10(2):150–161, 2010. URL https://doi.org/10.1002/pst.433. [p249]

D. Campbell and J. Stanley. *Experimental and Quasi-Experimental Designs for Research*. Houghton Mifflin Company, 1996. [p243]

W. Cochran and D. Rubin. Controlling bias in observational studies: A review. *Sankhyā: The Indian Journal of Statistics, Series A*, 35(4):417–446, 1973. URL https://www.jstor.org/stable/25049893. [p249]

R. Dehejia and S. Wahba. Propensity score-matching methods for nonexperimental causal studies. *Review of Economics and Statistics*, 84(1):151–161, 2002. URL https://doi.org/10.1162/003465302317331982. [p243]

K. Imai and M. Ratkovic. Robust estimation of inverse probability weights for marginal structural models. *Journal of the American Statistical Association*, 110, 2015. URL https://doi.org/10.1080/01621459.2014.956872. [p244]

B. Lee, J. Lessler, and E. Stuart. Improving propensity score weighting using machine learning. *Statistics in Medicine*, 29(3):337–346, 2010. URL https://doi.org/10.1002/sim.3782. [p243]

Y. Li, K. Propert, and P. Rosenbaum. Balanced risk set matching. *Journal of the American Statistical Association*, 96, 2011. URL https://doi.org/10.1198/016214501753208573. [p244]

M. Lunt. Selecting an appropriate caliper can be essential for achieving good balance with propensity score matching. *American Journal of Epidemiology*, 179(2):226–235, 2013. URL https://doi.org/10.1093/aje/kwt212. [p247]

C. Mack, R. Glynn, M. Brookhart, W. Carpenter, A. Meyer, R. Sandler, and T. Stüurmer. Calendar time-specific propensity scores and comparative effectiveness research for stage iii colon cancer chemotherapy. *Pharmacoepidemiology and Drug Safety*, 22(8):810–818, 2013. URL https://doi.org/10.1002/pds.3386. [p244]

B. Meyer. Natural and quasi-experiments in economics. *Journal of Business and Economic Statistics*, 13 (2):151–161, 1995. URL https://doi.org/10.2307/1392369. [p243]

D. Resnik. Randomized controlled trials in environmental health research: Ethical issues. *Journal of Environmental Health*, 70(6):28–30, 2008. URL https://www.ncbi.nlm.nih.gov/pubmed/18236934. [p243]

J. Robins, M. Hernán, and B. Brumback. Marginal structural models and causal inference in epidemiology. *Epidemiology*, 11(5):550–560, 2000. URL https://www.ncbi.nlm.nih.gov/pubmed/10955408. [p244]

P. Rosenbaum and D. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983. URL https://doi.org/10.1093/biomet/70.1.41. [p243, 249]

P. Rosenbaum and D. Rubin. Constructing a control group using multivariate matched sampling methods that incorporate the propensity score. *The American Statistician*, 39(1):33–38, 1985. URL https://doi.org/10.2307/2683903. [p248, 249]

D. Rubin. Bayesian inference for causal effects: The role of randomization. *The Annals of Statistics*, 6(1): 34–58, 1978. URL https://www.jstor.org/stable/2958688. [p243]

S. Schneeweiss, J. Gagne, R. Glynn, M. Ruhl, and J. Rassen. Assessing the comparative effectiveness of newly marketed medications: Methodological challenges and implications for drug development. *Clinical Pharmacology And Therapeutics*, 90(6):777–790, 2011. URL https://doi.org/10.1038/clpt.2011.235. [p244]

J. Seeger, P. Williams, and A. Walker. An application of propensity score matching using claims data. *Pharmacoepidemiology and Drug Safety*, 14(7):465–476, 2005. URL https://doi.org/10.1002/pds.1062. [p244, 248]

W. Shadish, T. Cook, and D. Campbell. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Houghton Mifflin Company, 2001. [p243]

E. Stuart. Matching methods for causal inference: A review and a look forward. *Statistical science : a review journal of the Institute of Mathematical Statistics*, 25(1):1–21, 2010. URL https://doi.org/10.1214/09-STS313. [p243]

Y. Wang, C. Hongwei, L. Chanjuan, L. Wang, S. Jiugang, and J. Xia. Optimal caliper width for propensity score matching of three treatment groups: A monte carlo study. *PLOS ONE*, 2013. URL https://doi.org/10.1371/journal.pone.0081045. [p251]

D. Westreich, J. Lessler, and M. Funk. Propensity score estimation: Machine learning and classification methods as alternatives to logistic regression. *Journal of Clinical Epidemiology*, 63(8):826–833, 2010. URL https://doi.org/10.1016/j.jclinepi.2009.11.020. [p243]

A. Witman, C. Beadles, L. Yiyan, A. Larsen, N. Kafali, S. Gandhi, P. Amico, and T. Hoerger. Comparison group selection in the presence of rolling entry for health services research: Rolling entry matching. *Health Services Research*, 2018. URL https://doi.org/10.1111/1475-6773.13086. [p243]

S. Yi. *New Matching algorithm?Outlier First Matching (OFM) and Its Performance on Propensity Score Analysis (PSA) under New Stepwise Matching Framework (SMF)*. PhD thesis, State University of New York at Albany, 2014. URL https://pqdtopen.proquest.com/doc/1610821085.html?FMT=ABS. [p244]

Kasey Jones
Division for Statistical and Data Sciences
RTI International
United States
krjones@rti.org

Rob Chew
Division for Statistical and Data Sciences
RTI International
United States
rchew@rti.org

Allison Witman
Assistant Professor of Economics
University of North Carolina Wilmington
United States
witmana@uncw.edu

Yiyan (Echo) Liu
Research Economist
RTI International
United States
yliu@rti.org

Associative Classification in R: arc, arulesCBA, and rCBA

by Michael Hahsler, Ian Johnson, Tomáš Kliegr and Jaroslav Kuchař

Abstract Several methods for creating classifiers based on rules discovered via association rule mining have been proposed in the literature. These classifiers are called associative classifiers and the best-known algorithm is Classification Based on Associations (CBA). Interestingly, only very few implementations are available and, until recently, no implementation was available for R. Now, three packages provide CBA. This paper introduces associative classification, the CBA algorithm, and how it can be used in R. A comparison of the three packages is provided to give the potential user an idea about the advantages of each of the implementations. We also show how the packages are related to the existing infrastructure for association rule mining already available in R.

Introduction

Association rule learning (Agrawal et al., 1993) was initially designed for data exploration to discover interesting patterns in very large and sparse datasets. Several years after its inception, association rule learning was also adapted to create rule-based classification models. The first algorithm called CBA (Classification Based on Associations) was introduced by Liu et al. (1998). While there were multiple follow-up algorithms providing some improvements in classification performance (e.g., CPAR (Yin and Han, 2003) and FARC-HD-OVO (Elkano et al., 2015)), these performance gains are offset by a deterioration of comprehensibility of the produced set of rules. For some practical applications, CBA still provides a very good balance between accuracy, speed, and model comprehensibility. Unlike many more recent approaches, CBA classifiers are easy to interpret and apply: the resulting ruleset is relatively small, rules are crisp (i.e., not fuzzy rules), and rules are sorted according to predictive strength. CBA uses a simple first-match strategy for classification, where the first matching rule determines the predicted class.

With the exception of fuzzy approaches such as FARC-HD, associative classification approaches require a dataset in the form of transactions, i.e., all attributes need to be binary indicators and thus numeric attributes in the input data need to be discretized. This puts additional demands on the user and may deteriorate model fit on datasets with numerical attributes. Another disadvantage relating to CBA and most other associative classification approaches is that these algorithms require the user to specify a minimum support and a minimum confidence threshold for association rule mining. The performance (accuracy and speed) is typically very sensitive to a proper selection of these threshold values. Setting these thresholds too high can result in the classifier underfitting the dataset or even an empty rule list. Too low values can lead to a combinatorial explosion with an excessive number of rules generated, leading to speed and memory issues. Another limitation that applies specifically to CBA is that even when the user specifies reasonable thresholds, CBA typically produces more rules than other related approaches (Alcala-Fdez et al., 2011). These limitations may be the reason why CBA implementations have not been available in many computational environments for machine learning and statistics. However, in the last several years, three packages with CBA implementations appeared on CRAN (listed by date of the first release): **rCBA** (Kuchar, 2018), **arc** (Kliegr, 2018) and **arulesCBA** (Johnson and Hahsler, 2019). Each of these packages offers some enhancements over the original CBA algorithm to address some shortcomings of association rule-based classification.

The goal of this paper is to introduce prospective users to the concepts used in associative classification and the CBA algorithm in particular. We provide detailed information on the three available R packages and the enhancements they provide, followed by hands-on examples.

The paper is organized as follows. We first introduce association rule mining, followed by a discussion of the CBA algorithm. We present existing CBA implementations and focus on the features and the use of the three new R implementations. We conclude with a short comparison of the features and a run-time comparison on a typical dataset.

Background: Association rule mining

Associative classifiers like CBA are based on association rules. Mining association rules was first introduced by Agrawal et al. (1993) and, following the notation used by Agrawal et al. (1993), Hahsler et al. (2005) and Tan et al. (2006), can formally be defined as:

Let $\mathbf{D} = \{t_1, t_2, \ldots, t_m\}$ be a set of transactions called the *database*, and let $I = \{i_1, i_2, \ldots, i_n\}$ be

transaction ID	**items**
1	milk, bread
2	bread, butter
3	beer
4	milk, bread, butter
5	bread, butter

Figure 1: An example supermarket database with five transactions.

the set of all *items* considered in the database. Each transaction in **D** has a unique transaction ID and contains a subset of the items in I. To illustrate the concepts, we use a small example from the supermarket domain introduced by Hahsler et al. (2005). The set of items is $I = \{\text{milk}, \text{bread}, \text{butter}, \text{beer}\}$ and a small database containing five transactions with these items is shown in Figure 1. An example rule for the supermarket could be $\{\text{milk}, \text{bread}\} \Rightarrow \{\text{butter}\}$ meaning that if milk and bread is bought, customers also may buy butter.

A *rule* is defined as an expression $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The sets of items (for short *itemsets*) X and Y are called *antecedent* (left-hand-side or LHS) and *consequent* (right-hand-side or RHS) of the rule. Often rules are restricted to only a single item in the consequent. *Association rules* are rules which meet user-specified minimum support and minimum confidence thresholds. The *support*, $\text{supp}(X)$, of an itemset X is a measure of importance defined as the proportion of transactions in the dataset which contain the itemset. The *confidence* of a rule is defined as $\text{conf}(X \Rightarrow Y) = \text{supp}(X \cup Y)/\text{supp}(X)$, measuring how likely it is to see Y in a transaction containing X.

An association rule $X \Rightarrow Y$ needs to satisfy

$$\text{supp}(X \cup Y) \geq \sigma \quad \text{and} \quad \text{conf}(X \Rightarrow Y) \geq \delta,$$

where σ and δ are the minimum support and minimum confidence thresholds, respectively. For example, the rule $\{\text{milk}, \text{bread}\} \Rightarrow \{\text{butter}\}$ has a support of $1/5 = 0.2$ and a confidence of $0.2/0.4 = 0.5$ in the database in Figure 1, which means that for 50% of the transactions containing milk and bread, the rule is correct. Confidence can be interpreted as an estimate of the probability $P(Y \mid X)$, the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS (see, e.g., Hipp et al., 2000).

Another popular measure for the importance of a rule is *lift* (Brin et al., 1997). The lift of a rule is defined as $\text{lift}(X \Rightarrow Y) = \text{supp}(X \cup Y)/(\text{supp}(X)\,\text{supp}(Y))$, and can be interpreted as the deviation of the support of the whole rule from the support expected under independence given the supports of the LHS and the RHS. Lift values greater than one indicate positive associations between the rule's LHS and RHS.

Because associative classification is based on association rules, transaction data is required as the input. Here each object (or instance) needs to be converted into a transaction containing only binary items. Discrete variables can be converted into items using a set of 0-1 dummy variables, one for each possible value. Continuous variables need to be first discretized and then converted. Typically, discretization for associative classifiers is performed using a class-based (also called supervised) discretization strategy, which identifies ranges for several intervals using information from the class variable (Yin and Han, 2003). The most popular method for class-based discretization is Minimum Description Length Principle (MDLP) discretization (Fayyad and Irani, 1993), which uses a greedy procedure to find cut points based on the entropy of the induced partition of the data with respect to the class variable. MDLP was also used in the initial paper on CBA (Liu et al., 1998). One of the advantages of MDLP is that there are no external parameters to be set; the optimal number of cut points is determined automatically using a stopping rule.

The CBA algorithm

Liu et al. (1998) proposed the first approach to associative classification called CBA. In CBA, a special type of association rules called *Class Association Rules (CARs)* are used for classification. A CAR is an association rule that conforms to the additional constraint that the consequent (RHS) of the rule is a single item that is associated with a class label for the classification problem. CBA proposed the following steps to perform associative classification (Vanhoof and Depaire, 2010):

1. Mine a set of class association rules (CARs),
2. prune and sort the rules,
3. classify new objects using the RHS of the first matching rule.

Within the original paper, the first step is handled by a modification of the popular APRIORI algorithm (Agrawal and Srikant, 1994) for mining CARs. The modification includes an optional pruning step based on the rule's pessimistic error rate with the goal to reduce the size of the set of considered CARs. According to results reported in Liu et al. (1998), the absence of pessimistic pruning does not affect classifier accuracy and a regular implementation of APRIORI can be used. The output of association rule learning algorithms is determined by two parameters, the minimum confidence and the support thresholds. In light of classification, confidence gives the proportion of objects correctly classified by the rule in the training set. Therefore it can be seen as an optimistic estimate of the accuracy of the rule.

The main obstacles for straightforward use of the discovered CARs as a classifier are the excessive number of rules discovered even on small datasets, the fact that contradicting rules are generated, and the absence of a default rule. To address these issues, CBA employs rule sorting and a *data coverage pruning* procedure to reduce the number of rules. Two variants were proposed in the original paper (Liu et al., 1998): the direct M1 version, and the M2 version which reduces data access. Accessing data fewer times is especially useful if the data is too large to be stored in main memory. The amount of available main memory has increased substantially since the original paper was published making the improvements of M2 less relevant. For pruning, the rules are first ranked in the order of their strength:

1. Rule A is ranked higher if confidence of rule A is greater than that of rule B.
2. For rules tied for 1, rule A is ranked higher if support of rule A is greater than that of rule B.
3. For rules tied for 1 and 2, rule A is ranked higher if rule A is produced before rule B in the mining process. Since APRIORI applies breath-first search, rule A is ranked higher if rule A has fewer conditions (i.e., a smaller antecedent set) than rule B.

Rules are processed in ranking order. After each rule is processed, the matching (covered) transactions are removed. If a rule does not correctly cover at least one instance, it is deleted (pruned). In CBA, data coverage pruning is combined with *default rule pruning*. A default rule is a rule added to the end of the rule set with the majority class in the uncovered transactions in the RHS and an empty LHS. This rule ensures that a query instance is always classified even if it is not matched by any other rule in the classifier. The algorithm prunes all rules below the current rule if a default rule inserted at that place reduces the total number of errors on the training set.

Other algorithms. Since CBA was introduced, several competing associative classification approaches have been proposed to improve accuracy, training time, and ruleset size. Two popular extensions of CBA are CMAR (Li et al., 2001) and CPAR (Yin and Han, 2003). A multiclass-focused approach called Multiclass Associative Classification (MAC) (Abdelhamid et al., 2012) has been proposed for expanding CBA with the goal of more accurately addressing classification problems with many different class labels. An approach related to associative classification is used by rule-induction classifiers which generate a large rulesets and then use greedy pruning strategies to reduce the size while maintaining classification accuracy. Common examples of this technique are RIPPER (Cohen, 1995) and SLIPPER (Cohen and Singer, 1999).

Recently, instead of relying on heuristics, several optimization approaches have been proposed for selecting the rules used by the classifier. Scalable Bayesian Rule Lists Model (Yang et al., 2017) tries to identify a small subset of mined CARs by optimizing the posterior of a Bayesian hierarchical model over rule lists. The method is implemented in the R package **sbrl** (Yang et al., 2019). Azmi et al. (2019) propose to learn optimal rule weights for associative classifiers that use the sum of the class weight of all matching rules instead of the first rule for classification. The authors use logistic regression with L1 regularization to learn rule weights while enforcing a small rule set. This approach is available in **arulesCBA** as function `RCAR()`.

While several alternative approaches have been introduced, CBA still acts as a strong contender in associative classification and is typically used as the benchmark against which new methods are assessed (Alcala-Fdez et al., 2011). A comparison between CBA and selected successors is performed in Kliegr (2017).

Implementations

There are only a few implementations of CBA available. Table 1 shows them ordered by the first release date and summarizes the used licenses and programming languages.

In the following, we discuss the three currently available implementations of CBA in R. We will first present each package individually and then compare the packages by providing code for the same classification problem implemented with each of the packages. We will use as the example dataset the well-known iris dataset (Fisher, 1936) and split it into 80% for training and 20% for testing.

Table 1: Review of existing CBA implementations

Software	1st release	License	Language	Notes
DM-II	2001	commercial	unknown	Original implementation by Liu et al. (1998). See http://www.comp.nus.edu.sg/~dm2/
LUCS-KDD	2004	not stated	Java	Endorsed by Bing Liu, the main CBA author. See http://cgi.csc.liv.ac.uk/~frans/KDD/Software/CBA/cba.html
KEEL	2010	GPL-3	Java	At the writing of this paper not available via **RKEEL**. See http://sci2s.ugr.es/keel/
rCBA	2015	Apache 2.0	R	See https://CRAN.R-project.org/package=rCBA
arc	2016	AGPL-3	R with Java	See https://CRAN.R-project.org/package=arc
arulesCBA	2016	GPL-3	R with C	From the authors of the **arules** R package. See https://CRAN.R-project.org/package=arulesCBA

```
data("iris")
iris <- iris[sample(seq(nrow(iris))), ]
iris_train <- iris[1:(nrow(iris)*.8), ]
iris_test <- iris[-(1:(nrow(iris)*.8)), ]
```

The data contains 150 flowers described by four quantitative variables representing different measurements and a categorical variable indicating one of three different species.

```
head(iris_train)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
18           5.1         3.5          1.4         0.3     setosa
93           5.8         2.6          4.0         1.2 versicolor
91           5.5         2.6          4.4         1.2 versicolor
92           6.1         3.0          4.6         1.4 versicolor
126          7.2         3.2          6.0         1.8  virginica
149          6.2         3.4          5.4         2.3  virginica
```

The classification problem we use for the examples is to predict a flower's species using the four measurements.

All three packages integrate with the infrastructure for association rule mining in R implemented in package **arules** (Hahsler et al., 2005) and the ecosystem of related packages (Hahsler et al., 2011). While the presented packages can perform discretization, the conversion of a dataset with continuous variable to a set of transactions with binary items, and mining class association rules (CARs) internally and transparent to the user, we will give here a short example of how the packages **arules** and **arulesCBA** can be used to perform these tasks. First, we discretize the data using supervised discretization based on the minimum description length principle (MDLP) offered by packages like **discretization** (Kim, 2012). Here we use the `discretizeDF.supervised` function provided in **arulesCBA**.

```
library("arules")
library("arulesCBA")

iris_train_disc <- discretizeDF.supervised(Species ~ ., data = iris_train,
  method = "mdlp")
head(iris_train_disc)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
18   [-Inf,5.55) [3.35, Inf]   [-Inf,2.6) [-Inf,0.75)     setosa
93   [5.55, Inf] [-Inf,2.95)   [2.6,4.75) [0.75,1.75) versicolor
91   [-Inf,5.55) [-Inf,2.95)   [2.6,4.75) [0.75,1.75) versicolor
92   [5.55, Inf] [2.95,3.35)   [2.6,4.75) [0.75,1.75) versicolor
126  [5.55, Inf] [2.95,3.35)  [5.05, Inf] [1.75, Inf]  virginica
149  [5.55, Inf] [3.35, Inf]  [5.05, Inf] [1.75, Inf]  virginica
```

Now we can convert the discretized data into transactions which automatically converts factors into binary items with labels composed of variable name and factor labels.

```
trans_train <- as(iris_train_disc, "transactions")
inspect(head(trans_train, n = 3))
```

```
    items                      transactionID
[1] {Sepal.Length=[-Inf,5.55),
     Sepal.Width=[3.35, Inf],
     Petal.Length=[-Inf,2.6),
     Petal.Width=[-Inf,0.75),
     Species=setosa}                    18
[2] {Sepal.Length=[5.55, Inf],
     Sepal.Width=[-Inf,2.95),
     Petal.Length=[2.6,4.75),
     Petal.Width=[0.75,1.75),
     Species=versicolor}                93
[3] {Sepal.Length=[-Inf,5.55),
     Sepal.Width=[-Inf,2.95),
     Petal.Length=[2.6,4.75),
     Petal.Width=[0.75,1.75),
     Species=versicolor}                91
```

Note that the class variable is translated into several items, all starting with `Species=`. From these transactions, CARs can be mined by restricting the items which can appear in the right-hand-side of the rules. This can be done with the APRIORI implementation available in **arules** by specifying appearance restrictions.

```
rules <- apriori(trans_train, parameter = list(support = 0.01, confidence = 0.8),
  appearance = list(rhs = grep("Species=",  itemLabels(trans_train), value = TRUE),
    default = "lhs"))
```

arulesCBA contains a convenience function called `mineCARs` to make setting the appropriate appearance easier using the standard formula interface.

```
rules <- mineCARs(Species ~ ., data = trans_train, support = 0.01, confidence = 0.8)
rules
```

```
set of 78 rules
```

```
inspect(head(rules, n = 3))
```

```
    lhs                           rhs                   support confidence lift count
[1] {Sepal.Width=[3.35, Inf]}  => {Species=setosa}     0.19    0.85       2.6  23
[2] {Petal.Length=[5.05, Inf]} => {Species=virginica}  0.27    1.00       3.0  32
[3] {Petal.Length=[2.6,4.75)}  => {Species=versicolor} 0.29    0.97       2.9  35
```

Test data can be discretized consistently with the training data using `discretizeDF`, which applies the discretization used in the second argument to the data in the first argument. Followed by a conversion to transactions.

```
iris_test_disc <- discretizeDF(iris_test, iris_train_disc)
trans_test <- as(iris_test_disc, "transactions")
```

While these steps are performed in most cases by the discussed packages internally, it is still helpful to understand the process. One of the advantages of associative classifiers is that the rule base can be inspected and, therefore, it is important to understand the transformations used to create items. Next, we will discuss the packages in alphabetical order.

Package arc

The R package **arc** (Kliegr, 2018) provides a pure R implementation of the rule pruning step of CBA. The association rule learning step is handled by the implementation of APRIORI in package **arules**. **arc** implements the M1 version of the CBA pruning step (Liu et al., 1998) and offers, in addition, automatic discretization and threshold tuning. A CBA model can be learned for the `iris` dataset as follows.

```
library("arc")
classifier <- arc::cba(iris_train, "Species")
```

The function `cba()` will create an instance of the S4 class `CBARuleModel` for the iris dataset using *Species* as the class variable. Note that discretization is performed and that the support and confidence thresholds are automatically found.

The resulting object holds a list of rules, a list of cut points (if discretization was automatically performed), the name of the class attribute, and a list of attribute types. The slot `rules` of the `CBARuleModel` object contains the rule base, which can be inspected by:

```
inspect(classifier@rules)
```

```
    lhs                          rhs                  support confidence lift count
[1] {Petal.Length=[-Inf;2.45],
     Petal.Width=[-Inf;0.75]}  => {Species=setosa}       0.333       1.00  3.0    40
[2] {Sepal.Length=(5.75; Inf],
     Petal.Length=(4.95; Inf],
     Petal.Width=(1.75; Inf]}  => {Species=virginica}    0.258       1.00  3.2    31
[3] {Sepal.Length=(5.75; Inf],
     Sepal.Width=[-Inf;3.15],
     Petal.Width=(1.75; Inf]}  => {Species=virginica}    0.200       1.00  3.2    24
[4] {Sepal.Length=(5.75; Inf],
     Petal.Length=(2.45;4.95],
     Petal.Width=(0.75;1.75]}  => {Species=versicolor}   0.200       1.00  2.8    24
[5] {Sepal.Length=(5.45;5.75],
     Sepal.Width=[-Inf;3.15],
     Petal.Length=(2.45;4.95],
     Petal.Width=(0.75;1.75]}  => {Species=versicolor}   0.092       1.00  2.8    11
[6] {Sepal.Length=(5.75; Inf],
     Sepal.Width=(3.15; Inf],
     Petal.Length=(2.45;4.95]} => {Species=versicolor}   0.042       1.00  2.8     5
[7] {Petal.Length=(2.45;4.95],
     Petal.Width=(0.75;1.75]}  => {Species=versicolor}   0.333       0.98  2.7    40
[8] {}                         => {Species=virginica}    0.308       0.31  1.0     0
```

Predictions for new data can be obtained using `predict()`. The new data is discretized automatically to match the rules.

```
predict(classifier, head(iris_test))
```

```
[1] virginica  setosa     versicolor virginica  setosa     versicolor
Levels: setosa versicolor virginica
```

Next, we discuss the new features of automatic discretization and threshold tuning.

Automatic discretization. Since association rule classification is a supervised task, the discretization can take advantage of using the class label. In the **arc** package, automatic discretization with MDLP is enabled by default. All numeric explanatory attributes with three or more distinct values are by default subject to discretization. The package relies on the **discretization** package (Kim, 2012). The **arc** package provides several convenience functions that allow to perform discretization of all attributes at once, addressing some of the shortcomings of the `mdlp` function from the **discretization** package, such as the inability to handle missing values, or skip non-numeric attributes. Only attributes containing at least a preset number of distinct values are discretized. The package is also capable of discretizing the target attribute if necessary. For this purpose, unsupervised discretization (clustering) is used.

Automatic threshold tuning. Association rule learning is notorious for how difficult it is to set the minimum support and minimum confidence thresholds. The necessity to set these thresholds also applies to CBA. The **arc** package contains an optional procedure for automatic setting of these thresholds detailed in (Kliegr and Kuchar, 2019) . The package contains a wrapper for the `apriori` function from the **arules** package that iterative changes mining parameters (maximum antecedent length, minimum support threshold and minimum confidence threshold) until a desired number of rules is obtained, all options are exhausted or a preset time limit is reached. The desired number of rules can be specified by the `target_rule_count` parameter.

The **arc** package also supports manual specification of thresholds:

```
classifier <- arc::cba(iris_train, "Species",
  rulelearning_options = list(minsupp = 0.05, minconf = 0.9,
```

```
    minlen = 1, maxlen = 5, maxtime = 1000, target_rule_count = 50000,
    trim=TRUE, find_conf_supp_thresholds = FALSE))

classifier@rules

set of 3 rules
```

Unlike other implementations of CBA, which also implement the M2 version of CBA described by Liu et al. (1998), the **arc** package relies solely on the M1 version. However, the implementation does not follow the originally proposed way relying on iteratively processing of rules in the sort order. Instead, the pruning steps in M1 are implemented using a more efficient multiplication of sparse matrices exposed by the **arules** package, which relies on the optimized C code from the **Matrix** package (Bates and Maechler, 2017).

Package arulesCBA

The **arulesCBA** package (Johnson and Hahsler, 2019) is an extension of the **arules** package and strives to integrate seamlessly with its association rule mining infrastructure. The packages allows the user to set a time limit for rule mining, exposed by the **arules** package. The core operations of **arulesCBA** are implemented in a mixture of R and C to speed up processing. **arulesCBA** implements both versions of the pruning step, M1 and the optimized M2 version. The code for the pruning algorithm is heavily optimized by using rule-indexed sparse matrix representation, sparse matrix operations via package **Matrix** (Bates and Maechler, 2017) and prefix trees.

The arulesCBA interface. In **arulesCBA**, classifiers are created using the `CBA()` function. An advantage of this package for R users is that it consistently uses the well-known formula interface for building classifier models and for supervised discretization. Users can provide a number of options to the function to tune discretization, rule mining, and model building. The following is the list of available parameters to the CBA function.

- `formula`: A symbolic description of the model to be fitted using a standard `formula` object of the from:

  $$\text{class} \sim \text{explanatory variables}$$

  The class is the variable name (part of the item label before `=`). Explanatory variables are separated using $+$ and the special dot symbol . for all variables is also allowed.
- `data`: A `data.frame` containing the training data. If necessary, discretization is automatically applied. Alternatively, also a transaction set can be supplied.
- `support,confidence`: Minimum support and confidence thresholds for mining CARs with APRIORI.
- `parameter,control`: Parameter and control lists passed on to the `apriori()` function from the **arules** package.
- `disc.method`: Discretization method for factorizing numeric input (default: "mdlp"). One of ('mdlp', 'caim', 'chi2', 'caac', 'ameva', 'chimerge', 'extendedchi2', 'modchi2').

A classifier for the iris dataset can be learned as follows.

```
library("arulesCBA")
classifier <- arulesCBA::CBA(Species ~ ., data = iris_train,
   supp = 0.05, confidence = 0.9)

classifier

CBA Classifier Object
Class: Species (labels: setosa, versicolor, virginica )
Default Class: Species=setosa
Number of rules: 2
Classification method: first
Description: CBA algorithm by Liu, et al. 1998 with support=0.05 and
     confidence=0.9
```

`CBA()` returns an object of class `CBA` which contains all needed information for classification. A print method shows the settings used for the classifier. Prediction follows the usual approach in R.

```
predict(classifier, head(iris_test))

[1] virginica  setosa     versicolor virginica  setosa     setosa
Levels: setosa versicolor virginica
```

The rule base is stored as a rules object from **arules** and can be extracted for inspection using the `rules()` function.

```
inspect(rules(classifier))

    lhs                           rhs                  support confidence lift count
[1] {Petal.Width=(1.75, Inf]}  => {Species=virginica}     0.29       1.00  3.1    35
[2] {Sepal.Length=(5.55, Inf],
     Petal.Width=(0.8,1.75]}   => {Species=versicolor}    0.26       0.91  2.7    31
```

Note that only two rules are shown, while **arc** above produced three rules. The reason is that **arulesCBA** stores the default class `Species=setosa` separate from the rule base while **arc** includes it.

Advanced use of arulesCBA. **arulesCBA** is implemented with flexibility and future extensions in mind. For example, to have optimal control over the discretization process, the user can discretize the data manually before learning the classifier. The discretization functions in **arules** and **arulesCBA** retain enough information so that `predict()` can later automatically discretize the new data.

Another extension implemented in `CBA_ruleset()` allows the user to create an associative classifier by providing a custom rule base in the form of a *rules* object. For example, we can easily create a classifier from a set of CARs using, for example, majority voting instead of CBA's first-match strategy for classification.

```
rules <- mineCARs(Species ~ ., trans_train,
  parameter = list(support = 0.01, confidence = 0.8))

classifier <- arulesCBA::CBA_ruleset(Species ~ ., rules, method = "majority")
classifier

CBA Classifier Object
Class: Species (labels: setosa, versicolor, virginica )
Default Class: Species=versicolor
Number of rules: 78
Classification method: majority
Description: Custom rule set
```

This gives the user the flexibility to experiment with different pruning methods and classification strategies.

Package rCBA

The **rCBA** package (Kuchar, 2018) was the first available implementation of the CBA algorithm on CRAN. The main algorithms are implemented in Java and it is the only R implementation that supports the use of multiple CPU cores during pruning. The package provides wrapper functions for pruning, prediction, and the FPGrowth association rule mining algorithm (Han et al., 2004). **rCBA** includes both, the M1 and the M2 version of the CBA algorithm. It also includes data coverage pruning and automatic threshold tuning.

Model building with automatic tuning of parameters and APRIORI is done as follows.

```
library("rCBA")
classifier <- rCBA::build(iris_train)

inspect(classifier$model)

1       {Petal.Width=0.2} => {Species=setosa}   0.183       1.00  2.9
2   {Petal.Width=1.3} => {Species=versicolor}   0.108       1.00  3.0
3      {Petal.Length=1.4} => {Species=setosa}   0.100       1.00  2.9
4      {Petal.Length=1.5} => {Species=setosa}   0.092       1.00  2.9
5    {Petal.Width=1.8} => {Species=virginica}   0.083       1.00  3.1
6    {Petal.Width=2.3} => {Species=virginica}   0.058       1.00  3.1
```

```
7       {Petal.Width=0.4} => {Species=setosa}   0.058       1.00  2.9
8    {Petal.Width=1.4} => {Species=versicolor}   0.058       1.00  3.0
9       {Sepal.Width=3.5} => {Species=setosa}   0.050       1.00  2.9
10      {Petal.Width=0.3} => {Species=setosa}   0.050       1.00  2.9
11    {Petal.Width=2.1} => {Species=virginica}   0.050       1.00  3.1
12    {Petal.Length=4} => {Species=versicolor}   0.042       1.00  3.0
13 {Petal.Length=4.7} => {Species=versicolor}   0.033       1.00  3.0
14   {Sepal.Length=7.7} => {Species=virginica}   0.033       1.00  3.1
15   {Petal.Width=1.2} => {Species=versicolor}   0.033       1.00  3.0
16      {Petal.Width=0.1} => {Species=setosa}   0.033       1.00  2.9
17    {Petal.Width=1.9} => {Species=virginica}   0.033       1.00  3.1
18     {Petal.Width=1} => {Species=versicolor}   0.033       1.00  3.0
19      {Sepal.Length=5.1} => {Species=setosa}   0.058       0.88  2.6
20 {Petal.Length=4.5} => {Species=versicolor}   0.050       0.86  2.6
21   {Petal.Length=5.1} => {Species=virginica}   0.042       0.83  2.6
22   {Petal.Width=1.5} => {Species=versicolor}   0.058       0.78  2.3
23 {Sepal.Length=5.5} => {Species=versicolor}   0.033       0.67  2.0
24                  {} => {Species=virginica}   0.325       0.33  1.0

rCBA::classification(head(iris_test), classifier$model)

[1] versicolor versicolor versicolor versicolor setosa     versicolor
Levels: setosa versicolor
```

Pruning methods. **rCBA** implements both version of the proposed pruning algorithms (Liu et al., 1998): the direct M1 version, and the optimized M2 version. It also offers the option to only use data coverage pruning, called *data coverage for business rule (dcbr)* (Kliegr et al., 2014).

Selection of algorithms for rule learning. The CBA algorithm can generally rely on any rule learning algorithm (Liu et al., 1998). By default, it uses the APRIORI implementation in **arules**, but it can also use **rCBA**'s own implementation of the FP-Growth algorithm (Han et al., 2004) for the association learning step.

```
rulebase <- rCBA::fpgrowth(iris_train, support = 0.05, confidence = 0.9,
  consequent = "Species")
rulebase <- rCBA::pruning(iris_train, rulebase, method = "m2cba")

rCBA::classification(head(iris_test), rulebase)

[1] versicolor versicolor versicolor versicolor setosa     versicolor
Levels: setosa versicolor
```

Automatic threshold tuning. Since pure random or grid search do not use any background knowledge of the algorithm, these approaches are unsuitable for optimizing the parameters of association rule learning. The implementation for the parameter optimization in **rCBA** is based on the simulated annealing (SA) algorithm, which addresses these problems. The objective criterion, which is optimized against, is the accuracy of the model. A detailed description of the approach can be found in Kliegr and Kuchar (2019).

Comparison of R implementations

In order to help the user to decide which package addresses best the particular use case, Table 2 presents a comparison of the features and limitations of the packages. Since all three packages implement the same algorithm, we did not compare classification accuracy between the implementations, but performed a small run-time comparison instead.

We compare the different implementations on some standard classification problems. The used datasets are available in the packages **mlbench**, **datasets**, **arules**, and the Lymphography dataset (Lymph) (Mickalski et al., 1986) was obtained from the UCI repository[1]. The most important dataset characteristics are summarized in Table 3. The number of transactions ranges from 101 to 48842 and the number of items (after discretization) from 15 to 147. We used for the comparison a minimum

[1] https://archive.ics.uci.edu/ml/datasets.html

Table 2: Comparison of features in CBA implementations in R

Feature	arc	arulesCBA	rCBA
CBA pruning	M1	M1/M2	M1/M2
Language	R	R + C	R + Java
Built-in discretization	MDLP	MDLP and others	No
Automatic threshold tuning	Unsupervised	No	Supervised
Recommended problem size	Small number of rules and instances	Many rules, many instances	Medium number of rules and instances

Table 3: Dataset characteristics. The dataset size ranges from 101 to more than 45,000 transactions and the datasets produce a wide variety in terms of the number of CARs and rule base size.

Dataset	Transactions	Items	Support	Confidence	CARs	Rule base	Accuracy
Zoo	101	26	0.01	0.50	3607	8	0.97
Lymph	147	63	0.10	0.50	16087	40	0.90
Iris	150	15	0.01	0.50	119	8	0.96
Ionosphere	351	147	0.40	0.50	16321	10	0.89
BreastCancer	699	91	0.01	0.50	5541	64	0.99
Pima	768	19	0.01	0.50	3536	76	0.80
Vehicle	846	77	0.08	0.50	13987	143	0.60
Adult	48842	115	0.10	0.50	932	6	0.77

confidence threshold of 0.5, a maximal rule length of 10 and set the minimum support so a reasonable number of classification association rules (CARs) was produced. CBA pruned the CARs to between 6 and 143 rules and achieves an accuracy (in sample testing) of typically around 90%. Only difficult datasets like Pima, Vehicle and Adult have worse results.

To compare run time, we conducted experiments on a standard laptop with an Intel Core i5-8250U CPU @ 1.60GHz with 4 cores and 8GB of RAM running R version 3.6.1 on Ubuntu 19.10. The package versions used for the comparison are: arc: 1.2, rCBA: 0.4.3, arulesCBA: 1.1.5. We disabled automatic threshold tuning. To remove the effect of random system load, we executed each algorithm ten times on each dataset and report the average execution time. The results are summarized in Table 4. arc produces the longest run times due to its pure R implementation. For Adult, the largest dataset arc ran out of memory. rCBA executes faster than arc. Both M2 and parallel execution using multi-core support in Java only improve the run time for the largest dataset. However, there the improvement is quite significant, reducing the run time to a third. arulesCBA's M1 implementation is on average the fastest while the M2 implementation's performance deteriorates on larger datasets.

Since many datasets of interest are typically larger then the standard datasets, we perform additional experiments to assess run time sensitivity for the number of input rules and the dataset size. For the experiments, we use the Lymph dataset. For assessing sensitivity to ruleset size, we oversample the dataset to 500 transactions and mine CARs with a minimum support of 0.05, a minimum confidence of 0.5 and a maximal rule length of 10. This results in more than 100000 rules. We then evaluate run time for building classifiers from the first 100, 1000, 10000, and 100000 mined rules. The results are shown in Figure 2(a). We see that M2 is generally slower than the corresponding M1 implementations. This might be due to the fact that the tested implementations hold all data in main memory, while M2 was designed for situations where the data does not reside in main memory. However, parallel execution helps rCBA's M2 implementation. arulesCBA's M1 implementation is the fastest.

To assess the sensitivity to dataset size, we fixed the ruleset size to 500 and increased the dataset size by oversampling every round by a factor of 2. In Figure 2(b), we see a similar result to the sensitivity to the number of rules. Parallel execution in rCBA helps both algorithms and arulesCBA's M1 implementation is the fastest. All packages are integrated with the arules infrastructure, where arulesCBA has the most consistent integration. arc and rCBA offer automatic threshold tuning, which will help users with applying associative classification for practical applications.

Conclusion

In this paper, we reviewed associative classifiers based on the CBA algorithm. While the algorithm is cited in many papers about classifiers based on association rule mining, there are only very few implementations available. This paper discussed three recent implementations in R packages. Due to the differences in implementation language (R, C, and Java) and additional implemented features,

Table 4: Comparison of the run time of the algorihtms for different datasets in milliseconds. **arc** ran out of memory (see *) and **arulesCBA**'s M1 implementation is on average the fastest.

	arc	rCBA				arulesCBA	
Dataset		M1	M1 parallel	M2	M2 parallel	M1	M2
Zoo	447.49	344.39	366.09	344.99	345.57	250.40	130.19
Lymph	3153.19	903.75	1070.12	1375.30	966.58	452.91	1539.03
Iris	117.98	282.57	332.43	273.37	283.54	147.33	102.74
Ionosphere	20350.29	13888.05	14180.38	14622.77	15185.99	4786.10	6766.72
BreastCancer	1895.51	488.06	620.62	1239.72	646.96	607.53	775.93
Pima	2508.53	837.69	854.77	847.39	837.06	707.97	982.45
Vehicle	50186.80	2741.48	2961.82	2935.02	3051.59	3966.96	12662.39
Adult	N/A*	10549.87	3395.19	9725.68	3091.15	1192.03	11737.05
Average	11237.00*	3754.48	2972.68	3920.53	3051.05	1513.90	4337.06

each of the packages has its strengths. We hope that this review and the provided examples help users to experiment with associative classifiers and that the packages will be used by the research community to develop new methods.

Acknowledgments

Tomas Kliegr was supported by long term institutional support of research activities and grant IGA 12/2019 by Faculty of Informatics and Statistics, University of Economics, Prague.

Ian Johnson was supported by the Goldwater Foundation and the President's Scholars program at Southern Methodist University, Dallas, TX, USA.

Bibliography

N. Abdelhamid, A. Ayesh, F. Thabtah, S. Ahmadi, and W. Hadi. Mac: A multiclass associative classification algorithm. *Journal of Information & Knowledge Management*, 11(02):1250011, 2012. [p256]

R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1-55860-153-8. [p256]

R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216. ACM Press, 1993. URL https://doi.org/10.1145/170035.170072. [p254]

J. Alcala-Fdez, R. Alcala, and F. Herrera. A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning. *IEEE Transactions on Fuzzy Systems*, 19(5):857–872, 2011. URL https://doi.org/10.1109/TFUZZ.2011.2147794. [p254, 256]

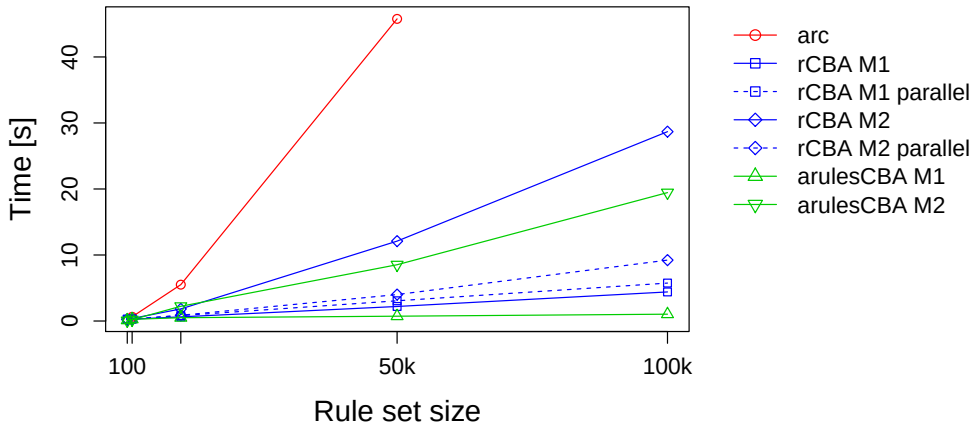
M. Azmi, G. C. Runger, and A. Berrado. Interpretable regularized class association rules algorithm for classification in a categorical data space. *Information Sciences*, 483:313–331, 2019. ISSN 0020-0255. URL https://doi.org/0.1016/j.ins.2019.01.047. [p256]

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2017. URL https://CRAN.R-project.org/package=Matrix. R package version 1.2-8. [p260]

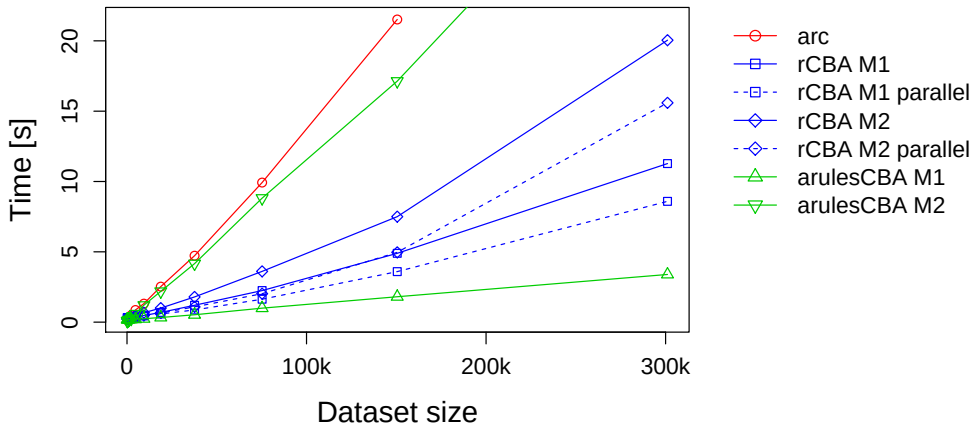
S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 255–264, Tucson, Arizona, USA, 1997. [p255]

W. W. Cohen. Fast effective rule induction. In *Machine Learning Proceedings 1995, Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Elsevier, 1995. URL https://doi.org/10.1016/B978-1-55860-377-6.50023-2. [p256]

W. W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *Publication:AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 335–342. AAAI, 1999. [p256]



(a) Sensitivity to ruleset size.



(b) Sensitivity to dataset size.

Figure 2: Comparison of the run time of different implementations on an oversampled Lymphography dataset

M. Elkano, M. Galar, J. A. Sanz, A. Fernández, E. Barrenechea, F. Herrera, and H. Bustince. Enhancing multiclass classification in farc-hd fuzzy classifier: On the synergy between n-dimensional overlap functions and decomposition strategies. *IEEE Transactions on Fuzzy Systems*, 23(5):1562–1580, 2015. ISSN 1063-6706. URL https://doi.org/10.1109/TFUZZ.2014.2370677. [p254]

U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. *Artificial intelligence, 13*, page 1022–1027, 1993. URL https://doi.org/10.1007/978-3-642-40897-7_11. [p255]

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7): 179–188, 1936. URL https://doi.org/10.1111/j.1469-1809.1936.tb02137.x. [p256]

M. Hahsler, B. Grün, and K. Hornik. Arules - a computational environment for mining association rules and frequent item sets. *Journal of Statistical Software*, 14(15):1–25, 2005. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v014.i15. [p254, 255, 257]

M. Hahsler, S. Chelluboina, K. Hornik, and C. Buchta. The arules R-package ecosystem: Analyzing interesting patterns from large transaction data sets. *Journal of Machine Learning Research*, 12(Jun): 2021–2025, 2011. [p257]

J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004. ISSN 1384-5810. URL https://doi.org/10.1023/B:DAMI.0000005258.31418.83. [p261, 262]

J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining – A general survey and comparison. *SIGKDD Explorations*, 2(2):1–58, 2000. URL https://doi.org/10.1145/360402.360421. [p255]

I. Johnson and M. Hahsler. *arulesCBA: Classification Based on Association Rules*, 2019. URL https://CRAN.R-project.org/package=arulesCBA. R package version 1.1.5. [p254, 260]

H. Kim. *Discretization: Data Preprocessing, Discretization for Classification.*, 2012. URL https://CRAN.R-project.org/package=discretization. R package version 1.0-1. [p257, 259]

T. Kliegr. QCBA: Postoptimization of quantitative attributes in classifiers based on association rules. *arXiv preprint*, 2017. URL https://arxiv.org/abs/1711.10166. [p256]

T. Kliegr. *Arc: Association Rule Classification*, 2018. URL https://CRAN.R-project.org/package=arc. R package version 1.2. [p254, 258]

T. Kliegr and J. Kuchar. Tuning hyperparameters of classification based on associations (CBA). In *Proceedings of the 19th Conference Information Technologies - Applications and Theory ITAT'19*. CEUR-WS.org, 2019. [p259, 262]

T. Kliegr, J. Kuchař, D. Sottara, and S. Vojíř. Learning business rules with association rule classifiers. In A. Bikakis, P. Fodor, and D. Roman, editors, *International Symposium on Rules and Rule Markup Languages for the Semantic Web (RuleML 2014): Rules on the Web. From Theory to Applications*, pages 236–250. Springer-Verlag, 2014. ISBN 978-3-319-09870-8. URL https://doi.org/10.1007/978-3-319-09870-8_18. [p262]

J. Kuchar. *rCBA: CBA Classifier for R*, 2018. URL https://CRAN.R-project.org/package=rCBA. R package version 0.4.3. [p254, 261]

W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 369–376, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1119-8. URL https://doi.org/10.1109/ICDM.2001.989541. [p256]

B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 80–86. AAAI Press, 1998. [p254, 255, 256, 257, 258, 260, 262]

R. S. Mickalski, I. Mozetic, H. J., and H. Lavrack. The multi purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the 5th National Conference on Artificial Intelligence*, 1986. [p262]

P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321321367. [p254]

K. Vanhoof and B. Depaire. Structure of association rule classifiers: a review. In *2010 International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 9–12, 2010. URL https://doi.org/10.1109/ISKE.2010.5680784. [p255]

H. Yang, C. Rudin, and M. Seltzer. Scalable Bayesian rule lists. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3921–3930. JMLR. org, 2017. [p256]

H. Yang, M. Chen, C. Rudin, and M. Seltzer. *sbrl: Scalable Bayesian Rule Lists Model*, 2019. URL https://CRAN.R-project.org/package=sbrl. R package version 1.2. [p256]

X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proceedings of the SIAM International Conference on Data Mining*, pages 369–376, San Franciso, 2003. SIAM Press. [p254, 255, 256]

Michael Hahsler
Office of Information Technology and Department of Engineering Management, Information, and Systems
Bobby B. Lyle School of Engineering
Southern Methodist University
P. O. Box 750123, Dallas, TX 75275, USA
mhahsler@lyle.smu.edu

Ian Johnson
Google,
Boulder, CO, USA
ianjjohnson@icloud.com

Tomáš Kliegr
Department of Information and Knowledge Engineering
Faculty of Informatics and Statistics
University of Economics, Prague
Winston Churchill Sq. 4, Prague, Czech Republic
ORCiD https://orcid.org/0000-0002-7261-0380
first.last@vse.cz

Jaroslav Kuchař
Web Intelligence Research Group
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00, Prague, Czech Republic
first.last@fit.cvut.cz

Indoor Positioning and Fingerprinting: The R Package ipft

by Emilio Sansano, Raúl Montoliu, Óscar Belmonte and Joaquín Torres-Sospedra

Abstract Methods based on Received Signal Strength Indicator (RSSI) fingerprinting are in the forefront among several techniques being proposed for indoor positioning. This paper introduces the R package **ipft**, which provides algorithms and utility functions for indoor positioning using fingerprinting techniques. These functions are designed for manipulation of RSSI fingerprint data sets, estimation of positions, comparison of the performance of different positioning models, and graphical visualization of data. Well-known machine learning algorithms are implemented in this package to perform analysis and estimations over RSSI data sets. The paper provides a description of these algorithms and functions, as well as examples of its use with real data. The **ipft** package provides a base that we hope to grow into a comprehensive library of fingerprinting-based indoor positioning methodologies.

Introduction

Intelligent spaces, as a particularity of the concept known as Ambient Intelligence (AmI) (Aarts and Wichert, 2009; Werner et al., 2005), where agents communicate and use technology in a non-intrusive way, have an interest in both open and closed environments. Since people spend 90% of time indoors (Klepeis et al., 2001), one of the most relevant aspects of AmI is indoor localization, due to the large number of potential applications: industrial and hospital applications, passenger transport, residences, assistance to emergency services and rescue, localization and support guide for the disabled, leisure applications, etc. It is expected that the global market for this type of location will grow from USD 7.11 billion in 2017 to USD 40.99 billion by 2022 (Research and markets, 2017), being among the key technologies in the future. This is a technology that has already awakened but that in a short period of time will suffer a big explosion, as happened with the systems of positioning by satellite in exteriors and its applications.

This paper introduces the R package **ipft** (Sansano, 2017), a collection of algorithms and utility functions to create models, make estimations, analyze and manipulate RSSI fingerprint data sets for indoor positioning. Given the abundance of potential applications for indoor positioning, the package may have a broad relevance in fields such as pervasive computing, Internet of Things (IoT) or healthcare, among many others.

The main progress in indoor location systems has been made during the last years. Therefore, both the research and commercial products in this area are new, and researchers and industry are currently involved in the investigation, development and improvement of these systems. We believe that the R language is a good environment for machine learning and data analysis related research, as its popularity is constantly growing [1], researchers related to indoor positioning have explicitly selected R as developing framework for their experiments (Quan et al., 2017; Harbicht et al., 2017; Popleteev et al., 2011), it is well maintained by an active community, and provides an ecosystem of good-quality packages that leverage its potential to become a standard programming platform for researchers. There are some open source applications and frameworks to build indoor positioning services, such as FIND [2], Anyplace [3] or RedPIN [4], based on fingerprinting techniques but, as far as we know, there is not any public framework or package that provides functions and algorithms to manipulate fingerprinting datasets and experiment with positioning algorithms.

RSSI (Received Signal Strength Indicator) positioning systems are based on measuring the intensities of the received radio signals of the emitting devices (beacons) that are available at a particular position, and comparing them with a previously built RSSI data set (yub Lee et al., 2013). RSSI is used to measure the relative quality of a received signal to a client device, and each chipset manufacturer is free to define their own scale for this term. The value read by a device is given on a logarithmic scale and can correspond to an instant reading or a mean of some consecutive readings.

In this scenario, a fingerprint is an RSSI feature vector composed of received signal values from different emitting devices or beacons, associated to a precise position. In the last years, this technique is becoming increasingly important for indoor localization (Liu et al., 2007; He and Chan, 2016), since Wi-Fi is generally available in indoor environments where GPS signals cannot penetrate, and the

[1] https://stackoverflow.blog/2017/10/10/impressive-growth-r/
[2] https://www.internalpositioning.com#about
[3] https://anyplace.cs.ucy.ac.cy
[4] http://redpin.org

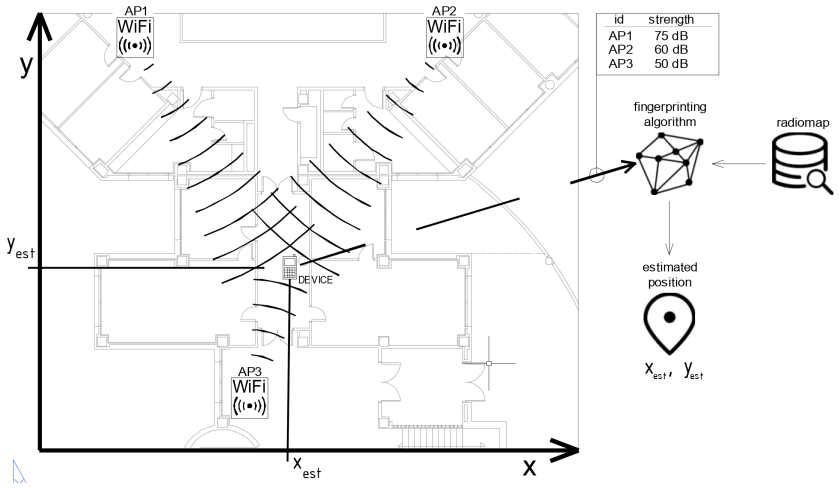


Figure 1: During the on-line phase, once the radio map has been built, the fingerprinting algorithm uses it to estimate the device's position by comparing the RSSI values heard by the device with the ones stored in the radio map.

wireless access points (WAPs) can be used as emitting devices (Li et al., 2006). Other types of indoor localization RF emitters, such as Bluetooth (Wang et al., 2013), RFID (Liu et al., 2011), or Ultra Wide Band (UWB) (Gigl et al., 2007), can be also used in combination with Wi-Fi access points or as a standalone positioning system.

The RSSI fingerprinting localization approach requires two phases of operation: a training phase, also known as off-line or survey phase, and a positioning phase, sometimes referred as on-line, runtime or tracking phase. In the training phase, multidimensional vectors of RSSI values (the fingerprints) are generated and associated with known locations. These measurements are used to build a data set (also known as radio map) that covers the area of interest. This data set can include, along with the collected RSSI values and the location coordinates, many other useful parameters, as the device type used in the measurements or its orientation. Later, during the positioning phase, an RSSI vector collected by a device is compared with the stored data to generate an estimation of its position (Figure 1).

Despite the increasing interest in RSSI positioning (Xiao et al., 2016), this topic has not been explicitly covered yet by any publicly available R package. The proposed package has been developed to provide users with a collection of fundamental algorithms and tools to manipulate RSSI radio maps and perform fingerprinting analysis. While fundamental algorithms and similarity measurement functions are implemented to provide a main framework for research and comparison purposes, these are highly customizable, to allow researchers to tailor those methods with their own parameters and functions.

This paper describes these algorithms and their implementation, and provides examples of how to use them. The remainder of the paper is structured as follows: Section Problem statement. Terminology and notation defines the fingerprinting problem statement and the nomenclature that will be used in the rest of the paper. An overview of the implemented algorithms is given in Section An overview of the implemented algorithms. Section Data wrangling outlines some data wrangling techniques included in the package. Section Positioning algorithms describes the implemented positioning algorithms. Section Beacon position estimation presents the included methods for access point position estimation. Then, Section Data clustering discuses some tools and functions included to create clusters or groups of fingerprints. Section Plotting functions illustrates the use of the plotting functions also included in the package. In all these sections, functions are described and explored using practical examples, and particular emphasis is placed on how to use them with real world examples and data sets. Finally, the paper is summarized in Section Summary.

Problem statement. Terminology and notation

This section provides a brief and general introduction to the principles of fingerprinting positioning, as well as a description of the notation and terminology that will be used in the next sections. The terms described here are related to general concepts of fingerprinting techniques, while the remaining of the paper describes the particular implementation of these concepts in the `ipft` package.

The main goal of the indoor localization techniques is to determine the position of a user in an

indoor environment, where the GPS signal might not be received. This objective might require the use of an existing infrastructure, the deployment of a new one, the use of the so-called signals-of-opportunity (Yang et al., 2014), or even a combination of some of these techniques. Many of these techniques take advantage of the radio-frequency signals emitted by devices, whose position can be known or not, to estimate the user's position from the perceived strength of these signals. There are many kinds of devices that can be used for this purpose, such as Wi-Fi access points, bluetooth beacons, RFID or UWB devices, but for all of them, the information provided for a given position, the fingerprint, can be stored as a vector of received signal strength intensities (RSSI), whose length is determined by the number of detected emitters.

A radio map, or a fingerprinting data set, is composed of a set of collected fingerprints and the associated positions where the measurements were taken, and may contain some additional variables, such as the the type of device used or a time stamp of the observation, among any other useful data. Let $\mathcal{D}$ be a fingerprinting data set. Then:

$$\mathcal{D} = \{\mathcal{F}, \mathcal{L}\}$$

where $\mathcal{F}$ is the set of collected fingerprints and $\mathcal{L}$ is the set of associated locations.

For research purposes, a fingerprinting data set is usually divided into training and test sets. The training data set is used to store the fingerprints and location data to create models of the environment that can be used to estimate the position of a new fingerprint. The test data set is used to test the models obtained from the training data, and to compute the errors from the results of the position estimation.

Let $\mathcal{D}_{train}$ be a training data set:

$$\mathcal{D}_{train} = \{\mathcal{F}_{train}, \mathcal{L}_{train}\}$$

where

$$\mathcal{F}_{train} = \left\{\lambda_1^{tr}, \lambda_2^{tr}, ..., \lambda_n^{tr}\right\}$$

$$\mathcal{L}_{train} = \left\{\tau_1^{tr}, \tau_2^{tr}, ..., \tau_n^{tr}\right\}$$

$\mathcal{D}_{train}$ is composed of n fingerprints, stored as n vectors of RSSI measurements ($\lambda_i^{tr},\ i \in [1, 2, ..., n]$), and n locations ($\tau_i^{tr},\ i \in [1, 2, ..., n]$), stored as vectors, representing the position associated with its correspondent fingerprint. Each fingerprint consists of q RSSI values ($\rho_{h,i}^{tr}, h \in [1, ..., q]$), where q is the number of beacons considered when building the training set:

$$\lambda_i^{tr} = \left\{\rho_{1,i}^{tr}, \rho_{2,i}^{tr}, ..., \rho_{q,i}^{tr}\right\},\ i \in [1, ..., n]$$

and each associated position is composed of one or more values, depending on the number of dimensions to be considered and the coordinate system used. The position can be given as a vector of values representing its coordinates, although on multi-floor and multi-building environments labels can be used to represent buildings, floors, offices, etc. Let l be the number of dimensions of a position vector. Then:

$$\tau_i^{tr} = \left\{\nu_{1,i}^{tr}, \nu_{2,i}^{tr}, ..., \nu_{l,i}^{tr}\right\},\ i \in [1, ..., n]$$

The test data set is also composed of a collection of fingerprints associated to known positions. This data set is used for testing purposes, during research or during model building adjustments, to assess the model's performance by comparing its estimation of the positions with the ground truth.

The situation is different in real applications, where the goal is to estimate the unknown position of the receiver given the RSSI values detected at a particular location, using a previously built model. In this case, the test data set is just composed of a unique fingerprint, and the objective is to estimate the actual location of the receiver. Therefore, no information about its location is provided.

The test data set is composed of m observations:

$$\mathcal{D}_{test} = \{\mathcal{F}_{test}, \mathcal{L}_{test}\}$$

where

$$\mathcal{F}_{test} = \left\{\lambda_1^{ts}, \lambda_2^{ts}, ..., \lambda_m^{ts}\right\}$$

$$\mathcal{L}_{test} = \{\tau_1^{ts}, \tau_2^{ts}, ..., \tau_m^{ts}\}$$

To be able to compare the test observations with the training fingerprints, the number of RSSI values of its respective fingerprints has to be the same, and the position in the RSSI vector must represent the same beacon in both data sets. Therefore, each one of the m observations of the test data set is composed of a fingerprint with q RSSI values:

$$\lambda_j^{ts} = \left\{\rho_{1,j}^{ts}, \rho_{2,j}^{ts}, ..., \rho_{q,j}^{ts}\right\},\ j \in [1, ..., m]$$

and a location vector with the same spatial dimensions as the training location vectors:

$$\tau_j^{ts} = \left\{\nu_{1,j}^{ts}, \nu_{2,j}^{ts}, ..., \nu_{l,j}^{ts}\right\},\ j \in [1, ..., m]$$

The notation depicted above will be used in the remaining of the paper to represent the fingerprinting data. Symbols i and j will be used to represent iterations over the training and test data sets, respectively, while h will be used to iterate over the beacons present in each fingerprint.

An overview of the implemented algorithms

This section presents an introduction to the main functions, included in the **ipft**[5] package, that implement fingerprinting-based indoor localization methods. The package also provides two data sets for training and validation purposes that are briefly described in this section.

The `ipft` package implements three algorithms to build models to estimate the position of a receiver in an indoor environment. Two of these implementations are based on the well known k-Nearest Neighbors algorithm (*knn*) (Cover and Hart, 1967) to, given an RSSI vector, select the k most similar training examples from the radio map. The similarity between the RSSI value vectors can be measured, for example, as the *euclidean* distance between them, but other distance functions may be used (Torres-Sospedra et al., 2015b). The selection of a method to compute this measure can be provided to the function in two ways, either choosing one of the already implemented distance measurements (*euclidean*, *manhattan*, etc.), or by way of a reference to a function implemented by the user that returns the distance (the lower, the more similar or 'closer') between two matrices or vectors. Once the k neighbors are selected, the location of the user is estimated as the weighted average of the neighbors positions.

The first implementation, corresponding to the function `ipfKnn`, may behave in a deterministic way, finding the k more similar neighbors using a deterministic similarity function such as the euclidean or manhattan distances, or in a probabilistic way, using similarity functions such as LDG (Logarithmic Gaussian Distance) or PLGD (Penalized Logarithmic Gaussian Distance) (Cramariuc et al., 2016b), that are based upon statistical assumptions on the RSSI measurement error. The similarity function can be chosen from the set of implemented options or provided by the user via a custom function. This implementation is discussed in the Section The `ipfKnn` function.

The other implementation of the knn algorithm assumes a probabilistic nature for the received signal distribution (Roos et al., 2002) and uses collections of many fingerprints at each particular position, acquired during the training phase. Therefore, the radio map is composed of several groups, where a group is a set of fingerprints (vectors of RSSI values) that share the same location. Assuming that the RSSI value for a specific beacon can be modeled as a random variable following a normal distribution (Haeberlen et al., 2004), any of these collections, or groups, of fingerprints can be represented by the statistical parameters of this distribution, in this case, the mean and the standard deviation. This implies that the original data set can be transformed into a new type of data structure by storing the mean and the standard deviation of every detected beacon for every group. All the original data for a group is transformed into two vectors, one storing the means and the other the standard deviations. The trustworthiness of the data in the new data set will depend on the number of measurements for every location of the original data. It is assumed that the more measurements for a particular location, the more reliable will be their inferred statistical parameters.

The implementation of this probabilistic-based method takes the original radio map and a set of group indices, and fits these groups of measurements to a normal (Gaussian) distribution for every beacon and every location, so that the signal intensity distribution is determined by the mean

[5]The **ipft** package is available at CRAN and can be installed as any other R package:

```
> install.packages("ipft")
```

The package has to be loaded into the main environment to use it for the first time in an R session:

```
> library("ipft")
```

and the standard deviation of the Gaussian fit. Then, given a test fingerprint, the algorithm estimates its position by selecting the k most probable locations, making explicit use of the statistical parameters of the data stored in the radio map to optimize the probabilities in the assignment of the estimated position by computing a similarity function based on a summatory of probabilities. This approach is implemented through the `ipfProbabilistic` function and is described in the Section The `ipfProbabilistic` function.

Finally, the third implemented algorithm is based on a scenario where the location of the beacons is known, and an estimation of the fingerprint position can be made using the log-distance path loss model (Seybold, J.S., 2005). The strength of the received signal at a particular point can be modeled as a function of the logarithmic distance between the receiver and the emitter and some parameters related to the environment properties and the devices characteristics. Therefore, as this method uses an analytical model to evaluate the position, no radio map is needed to train a model to compare fingerprints with, since the position might be estimated from the fingerprint data and the position of the beacons. This method is implemented by the function `ipfProximity` and is described in Section The `ipfProximity` function.

The previous functions `ipfKnn`, `ipfProbabilistic` and `ipfProximity` create models based on the training data and parameters provided. These models can then be evaluated using the `ipfEstimate` function, that internally detects the algorithm to apply based on the model that receives as parameter.

The package also includes data from the IPIN2016[6] Tutorial data set. In the `ipftrain` data frame there are $n = 927$ observations, including the RSSI values for $q = 168$ wireless access points, the location, expressed in Cartesian coordinates, for the observation (x, y), and some other variables, as timestamps for the measurements or an identifier for the user who took the survey. The `ipftest` data frame contains $m = 702$ observations with the same structure, for testing and validation purposes. The fingerprints included in both data sets where taken in the same building and the same floor. The `ipfpwap` data frame contains the position of 39 of the WAPs included in the `ipftrain` and `ipftest` data sets. The unknown positions of the remaining WAPs are stored as `NA`. The characteristics of these data sets attributes are:

- `RSSI values`: Columns from 1 to 168. The values represent the strength of the received signal expressed in decibels, on a scale that ranges from -30dBm to -97dBm in the training set, and from -31dBm to -99dBm in the test set. The closer the value to zero, the stronger the signal.
- `position`: Columns 169 (X) and 170 (Y). The position given in Cartesian coordinates, with its origin in the same corridor where the data was acquired.
- `user id`: A numeric value from 1 to 8 to represent each of the 8 users that acquired the train data set. The test dataset was acquired by a different user, represented by the value 0.
- `timestamp`: The UNIX time stamp of the observation, in seconds.

There are some other publicly available indoor location data sets that have been used to develop and test this package and that are not included for size reasons, as the UJIIndoorLoc Data Set (Torres-Sospedra et al., 2015a) or the Tampere University data set (Cramariuc et al., 2016a).

The theoretical foundations of the algorithms and its uses are discussed in detail in Section Positioning algorithms. A description of the functions `ipfKnn`, `ipfProximity`, `ipfProbabilistic` and `ipfEstimate` is given while presenting some simulations to show how these algorithms can be useful in practice.

Data wrangling

An RSSI fingerprint is a vector composed of signal strength measurements from all the emitters received by a client device at a particular point, and can be measured in any unit of power. It is often expressed in decibels (dBm), or as percentage values between 1-100, and can be a negative or a positive value. Typically this values are stored as negative figures, where the strongest signals are closer to zero.

Some algorithms are sensitive to the scale of the data. For example, Neural Networks generally work better (**?**) with data scaled to a range between $[0, 1]$ or $[-1, 1]$, since unscaled data may slow down the learning process and the convergence of the network parameters and, in some cases, prevent the network from effectively learning the problem. Thus, the first step before the data can be fed to a positioning algorithm may involve some kind of transformation, depending on the characteristics of the original data.

The data sets included in this package represent the RSSI data from a set of wireless access points as negative integer numbers from -99 (weakest detected signal) to -30 (strongest detected signal).

[6] http://www3.uah.es/ipin2016/

When the RSSI of a WAP is not available, the value used is NA. This convention may be inconvenient for some calculations. For example, a similarity measure between two fingerprints as the euclidean distance will only take into account those WAPs that have been detected in both observations, causing a loss of information that otherwise could be utilized.

The **ipft** package contains some functions to manipulate and wrangle raw fingerprint data. The `ipfTransform` function mutates the given fingerprint data into a new data set with a specified range for the RSSI signals. The signature of the function is:

```
ipfTransform <- function(data, outRange = c(0, 1), outNoRSSI = 0, inRange = NULL,
                         inNoRSSI = 0, trans = "scale", alpha = 24)
```

where:

- `data`: The input data set with the original RSSI fingerprints.
- `outRange`: A numeric vector with two values indicating the desired range of the output data.
- `outNoRSSI`: The desired value for not detected beacons in the output data.
- `inRange`: A numeric vector with two values indicating the range of signal strength values in the input data. If this parameter is not provided, the function will infer it from the provided data.
- `inNoRSSI`: The value given to a not detected beacon in the original data.
- `trans`: The transformation to perform over the RSSI data, either 'scale' or 'exponential'.
- `alpha`: The α parameter for the exponential transformation.

The *scale* transformation scales the input data values to a range specified by the user. The feature scaling is performed according to Equation 1:

$$\rho_{h,i}^{out} = \begin{cases} a + b \cdot \rho_{h,i}^{in}, & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ outNoRSSI, & otherwise \end{cases} \tag{1}$$

$$b = \frac{outMin - outMax}{inMin - inMax}$$
$$a = outMin - inMin \cdot b$$

where:

- $\rho_{h,i}^{out}$ and $\rho_{h,i}^{in}$ are the output and input RSSI values, respectively, for the h^{th} beacon from the i^{th} observation
- $outMax$ and $outMin$ are the maximum and minimum values, respectively, specified for the output by the `outRange` parameter.
- $inMax$ and $inMin$ are the maximum and minimum values, respectively, of the input data.
- $outNoRSSI$ and $inNoRSSI$ are the values assigned in the fingerprint to represent a not detected beacon for the output and input data, respectively, specified by the parameters `outNoRSSI` and `inNoRSSI`.

The *exponential* transformation (Torres-Sospedra et al., 2015b) changes the data according to the next equation:

$$\rho_{h,i}^{out} = \begin{cases} \exp(\frac{pos(\rho_{h,i}^{in})}{\alpha}), & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ outNoRSSI, & otherwise \end{cases}$$

$$pos(\rho_{h,i}^{in}) = \begin{cases} \rho_{h,i}^{in} - inMin, & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ 0, & otherwise \end{cases}$$

where α is a parameter for the exponential transformation. The authors establish α as a case-based parameter, and find that 24 is a good value for RSSI fingerprinting data, but they did not study the effects of α in the transformed data.

The following code scales the `ipftrain` and `ipftest` data sets RSSI data, stored in the columns 1:168, to a positive range of values, from 0 to 1, with NA representing a not detected WAP. As a not detected WAP is represented by a NA value in the original data, this has to be indicated to the function so it can transform these values to the desired output:

```
trainRSSI <- ipfTransform(ipftrain[, 1:168], outRange = c(0.1, 1), inNoRSSI = NA,
                          outNoRSSI = NA)
testRSSI <- ipfTransform(ipftest[, 1:168], outRange = c(0.1, 1), inNoRSSI = NA,
                         outNoRSSI = NA)
```

The `ipfTransform` function returns a new data set with the same structure (vector, matrix or data frame) as the input.

Positioning algorithms

This section describes three positioning algorithms implemented in the `ipft` package. The examples illustrating each description are based on the data previously scaled in Section Data wrangling .

The `ipfKnn` function.

The `ipfKnn` and `ipfEstimate` functions implement a version of the knn algorithm to select the k nearest neighbors (the k more similar vectors from the training set) to a given RSSI vector. Many different distance metrics (Torres-Sospedra et al., 2015b) can be used to compare two RSSI vectors and measure how 'near' or similar they are.

The distance metrics implemented in the package include some typical functions, as the L^1 norm, or manhattan distance, or the L^2, or euclidean distance. The L^u norm between two fingerprints with indices a and b is defined as follows:

$$L^u = \left(\sum_{h=1}^{q} |(\rho_{h,a} - \rho_{h,b}|^u \right)^{1/u}$$

The package also implements some fingerprinting specific distance estimation functions such as LDG and PLGD. The LGD between two RSSI vectors λ_i^{tr} and λ_j^{ts} of longitude q is given by:

$$LGD(\lambda_i^{tr}, \lambda_j^{ts}) = -\sum_{h=1}^{q} \log\ \max(G(\rho_{h,i}^{tr},\ \rho_{h,j}^{ts}),\ \epsilon)$$

where ϵ is a parameter to avoid logarithm of zero, as well as having one beacon RSSI value influence the LGD only above a certain threshold. $G(\rho_{h,i}^{tr},\ \rho_{h,j}^{ts})$ represents the Gaussian similarity between $\rho_{h,i}^{tr}$ and $\rho_{h,j}^{ts}$, defined as

$$\mathrm{G}(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\rho_{h,i}^{tr} - \rho_{h,j}^{ts})^2}{2\sigma^2} \right), & \text{if } \rho_{h,i}^{tr} \neq 0 \text{ and } \rho_{h,j}^{ts} \neq 0 \\ 0, & otherwise \end{cases}$$

The σ^2 parameter represents the shadowing variance (Shrestha et al., 2013). Values for σ in the range between 4 and 10 dBm are usually good for indoor scenarios (Lohan et al., 2014).

The PLGD between two RSSI vectors λ_i^{tr} and λ_j^{ts} of longitude q is given as:

$$PLGD(\lambda_i^{tr}, \lambda_j^{ts}) = LGD(\lambda_i^{tr}, \lambda_j^{ts}) + \alpha(\phi(\lambda_i^{tr}, \lambda_j^{ts}) + \phi(\lambda_j^{ts}, \lambda_i^{tr}))$$

where $\phi(\lambda_i^{tr}, \lambda_j^{ts})$ is a penalty function for the beacons that are visible in the i^{th} training fingerprint but not in the j^{th} test fingerprint, $\phi(\lambda_j^{ts}, \lambda_i^{tr})$ is a penalty function for the beacons that are visible in the j^{th} test fingerprint but not in the i^{th} training fingerprint, and are defined as follows:

$$\phi(\lambda_i^{tr}, \lambda_j^{ts}) = \sum_{h=1}^{q} T_{max} - \rho_{h,i}^{tr},\ \text{for}\ 0 < \rho_{h,i}^{tr} \leq T_{max} \text{ and } r_i = 0)$$

$$\phi(\lambda_j^{ts}, \lambda_i^{tr}) = \sum_{h=1}^{q} T_{max} - \rho_{h,j}^{ts},\ \text{for}\ 0 < \rho_{h,j}^{ts} \leq T_{max} \text{ and } r_j = 0)$$

T_{max} is an upper threshold for the strength of the signal, and α is a scaling factor.

The similarity measurement method can be chosen by means of the parameter `method`, or by providing a custom function (parameters `FUN` and `...`). The signature of the `ipfKnn` function is:

```
ipfKnn <- function(train_fgp, train_pos, k = 3, method = 'euclidean',
```

```
weights = 'distance', norm = 2, sd = 5, epsilon = 1e-3,
alpha = 1, threshold = 20, FUN = NULL, ...)
```

where:

- `train_fgp`: A data frame of n rows and q columns containing the fingerprint vectors of the training set.
- `train_pos`: A data frame of n rows and l columns containing the positions of the training observations.
- `k`: The k parameter of the *knn* algorithm, the number of nearest neighbors to consider.
- `method`: The distance metric to be used by the algorithm. The implemented options are 'euclidean', 'manhatan', 'norm', 'LGD' and 'PLGD'
- `weights`: The weight function to be used by the algorithm. The implemented options are 'distance' and 'uniform'. The default 'distance' function calculate the weights from the distances as:

  $$w_{j,t} = \frac{1}{(1 + d_{j,t})\mathcal{W}_j}$$

  where $w_{j,t}$ is the weight assigned to the t^{th} ($t \in [1..k]$) neighbor of the j^{th} ($j \in [1..m]$) test observation, $d_{j,t}$ is the distance in the feature (RSSI) space between the t^{th} neighbor and the j^{th} test fingerprint, and $\mathcal{W}_j$ is a term used to normalize the values so that the total sum of the k weights is 1.

  The 'uniform' function assigns the same weight value to each neighbor:

  $$w_{j,t} = \frac{1}{k}$$

- `norm`, `sd`, `epsilon`, `alpha`, `threshold`: Parameters for the 'norm', 'LGD' and 'PLGD' methods.
- `FUN`: An alternative function provided by the user to compute the distance.
- `...`: Additional parameters for the function FUN.

For a training data set of n RSSI vectors (a data frame or a matrix named `tr_fingerprints`) and a data set of n position vectors (a data frame or a matrix named `tr_positions`), the code for fitting a knn model with a k value of 4 and the manhattan distance as the distance measurement method is:

```
knnModel <- ipfKnn(tr_fingerprints, tr_positions, k = 4, method = 'manhattan')
```

This function returns an S3 object of class `ipftModel` containing the following properties:

- `params`: A list with the parameters passed to the function.
- `data`: A list with the fingerprints and the location data of the radio map.

To estimate the position of a new fingerprint, the `ipfEstimate` function makes use of the previously obtained model. An `ipfModel` object holds the data model needed by the `ipfEstimate` function to apply the selected algorithm and returns an estimation of the test fingerprints positions. The signature of `ipfEstimate` is:

```
ipfEstimate <- function(ipfmodel, test_fgp, test_pos = NULL)
```

where:

- `ipfmodel`: An S3 object of class `ipfModel`.
- `test_fgp`: A data frame of m rows and q columns containing the fingerprints of the test set.
- `test_pos`: An optional parameter containing a data frame of m rows and l columns with the position of the test observations.

The `ipfEstimate` function returns an S3 object of the class `ipfEstimation` with the following elements:

- `location`: A $m \times l$ matrix with the predicted position for each observation in the `test` data set.
- `errors`: If the actual location of the test observations is passed in parameter `test_pos`, and the data that represents the position is numeric, this property returns a numeric vector of length n with the errors, calculated as the *euclidean* distances between the actual and the predicted locations.

- `confusion`: If the actual location of the test observations is passed in parameter `test_pos`, and the data that represents the position is a factor, the estimation of the actual position is performed as a classification task, and this property returns a confusion matrix summarizing the results of this classification.
- `neighbors`: A $m \times k$ matrix with the indices of the k selected neighbors for each observation in the `test` data set.
- `weights`: A $m \times k$ matrix containing the weights assigned by the algorithm to the selected neighbors.

The following `R` code shows an example of the usage of the `ipfKnn` function with the data set included in the package. This example takes the data previously scaled and generates a positioning model from the input data `trainRSSI` (the radio map) that is stored in `knnModel`. Then, the model is passed to the `ipfEstimate` function, along with the test data, to get an estimation of the position of the 702 test observations:

```
tr_fingerprints <- trainRSSI[, 1:168]
tr_positions    <- ipftrain[, 169:170]
knnModel        <- ipfKnn(tr_fingerprints, tr_positions, k = 7, method = "euclidean")
ts_fingerprints <- testRSSI[, 1:168]
ts_positions    <- ipftest[, 169:170]
knnEstimation   <- ipfEstimate(knnModel, ts_fingerprints, ts_positions)
```

Since the position of the test observations is known, the mean error for the 702 test observations can be calculated as follows:

```
> mean(knnEstimation$errors)
[1] 3.302739
```

The mean positioning error is one of the most common evaluation metrics used in indoor positioning (Liu et al., 2007) to assess the system's accuracy. This metric corresponds to the average Euclidean distance between the estimated locations and the true locations. As positions in the `ipftrain` and `ipftest` are expressed in meters, this metric represents the average error in meters for this scenario.

The neighbors selected from the training data set for the 6 first test fingerprints are:

```
> head(knnEstimation$neighbors)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   71  176  126  125  127  771  130
[2,]   71  176  126  125  127  771  130
[3,]  465  914  915  913  217   77  218
[4,]  465  914  915  176  913  461  217
[5,]  176  126  125  771  130  127  914
[6,]   77  914  915  217  176  465  218
```

where each row of the output corresponds to the indices of the $k = 7$ more similar vectors from the training data set to the i^{th} vector of the `test` data set.

As an example of how to use `ipfKnn` with a custom function, the next code shows the definition of a C++ function that implements a modified version of the manhattan distance. The function needs at least two parameters, the two matrices representing the training and test data sets. A third parameter is here introduced to represent a penalization value. This function penalizes the computed distance between two RSSI measurements when one of the beacons is not detected (represented by the value $\emptyset$), by multiplying the resulting distance by a factor F. Given two fingerprints λ_i^{tr} and λ_j^{ts} of length q, the *myD* distance is:

$$\text{myD}(\lambda_i^{tr}, \lambda_j^{ts}) = \sum_{h=1}^{q} \text{myd}(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}),$$

where

$$\text{myd}(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}) = \begin{cases} |\rho_{h,i}^{tr} - \rho_{h,j}^{ts}|, & \text{if } \rho_{h,i}^{tr} \neq \emptyset \text{ and } \rho_{h,j}^{ts} \neq \emptyset \\ |\rho_{h,i}^{tr} - \rho_{h,j}^{ts}|F, & otherwise \end{cases}$$

The following code implements the `myD` function and shows an example of its usage with `ipfKnn`, as well as the results obtained. The function is coded in C++ to improve its performance when using

large data sets, although the method also accepts custom plain R functions. The `myD` function assumes that the fingerprints are in a positive range:

```
library('ipft')
library('Rcpp')
cppFunction('
  NumericMatrix myD(NumericMatrix train, NumericMatrix test, double F = 2.0) {
    NumericMatrix distanceMatrix(test.nrow(), train.nrow());
    double d = 0, pv = 0, rssi1 = 0, rssi2 = 0;
    for (int itrain = 0; itrain < train.nrow(); itrain++) {
      for (int itest = 0; itest < test.nrow(); itest++) {
        d = 0;
        for (int i = 0; i < train.ncol(); i++) {
          rssi1 = R_IsNA(train(itrain, i))? 0 : train(itrain, i);
          rssi2 = R_IsNA(test(itest, i))? 0 : test(itest, i);
          pv = (rssi1 != 0 && rssi2 != 0)? 1 : F;
          d = d + std::abs(rssi1 - rssi2) * pv;
        }
        distanceMatrix(itest, itrain) = d;
      }
    }
    return distanceMatrix;
  }'
)
customModel      <- ipfKnn(tr_fingerprints, tr_positions, k = 1, FUN = myD, F = 0.25)
customEstimation <- ipfEstimate(customModel, ts_fingerprints, ts_positions)

> head(customEstimation$neighbors)
     [,1]
[1,]  773
[2,]  773
[3,]  776
[4,]  773
[5,]  130
[6,]  130
```

The previous code outputs the selected neighbors for the first 6 observations in the test data set. As the `ts_positions` data frame contains the actual location of the observations, the absolute error committed by the model is returned in the `ipfEstimation` object:

```
> head(customEstimation$errors)
[1] 5.708275 5.708275 5.708275 5.708275 3.380000 3.380000

And the mean error with this custom similarity function is:

> mean(customEstimation$errors)
[1] 3.297342
```

An `ipfEstimation` object can be used directly to plot the Empirical cumulative distribution function of the error (function `ipfPlotEcdf()`) and the Probability density function (function `ipfPlotPdf()`). Figures 1 and 2 show the plots obtained from the following code:

```
> ipfPlotEcdf(customEstimation)
> ipfPlotPdf(customEstimation)
```

The plotting functions included in the package are described in detail in Section Plotting functions.

The `ipfProbabilistic` function.

Given the limitations of sensors accuracy (Luo and Zhan, 2014) and the irregular character of signal propagation (Ali et al., 2010), the RSSI vector stored for a particular position cannot have completely reliable and accurate information about the emitters signal strength. This uncertainty is generally modeled by a normal distribution (Haeberlen et al., 2004), but to do so many readings of the signals at the same position are needed to obtain a representative set of statistical parameters to model each RSSI present at that position.

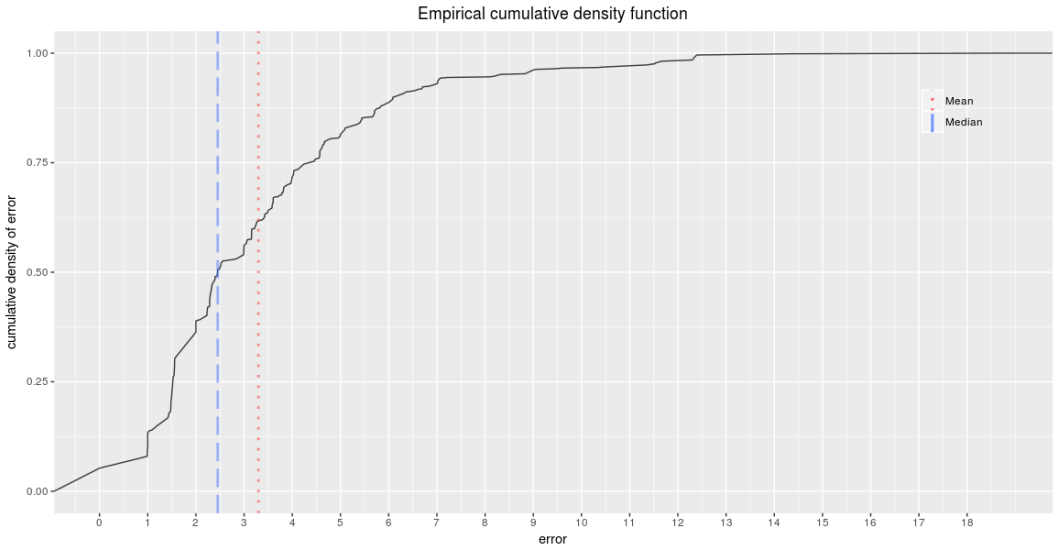


Figure 2: Funtion `ipfPlotEcdf`. Empirical cumulative distribution function of the error. The plot also shows the mean (red dotted line) and the median (blue dashed line) of the errors.

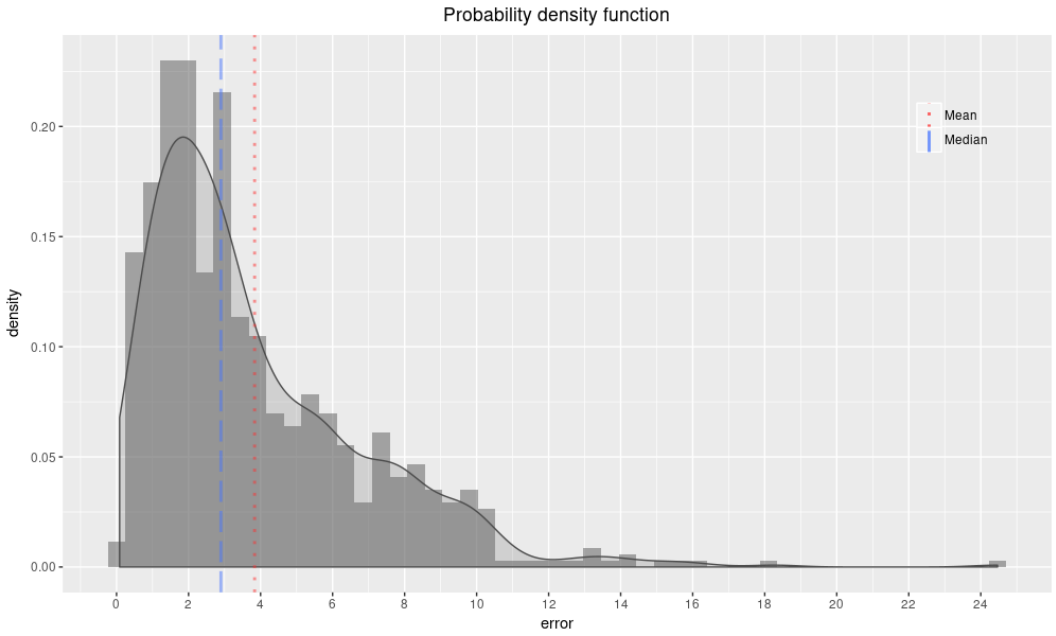


Figure 3: Funtion `ipfPlotPdf`. Probability density function. The plot shows the normalized histogram of the errors and its density function. The plot also shows the mean (red dotted line) and the median (blue dashed line) of the errors.

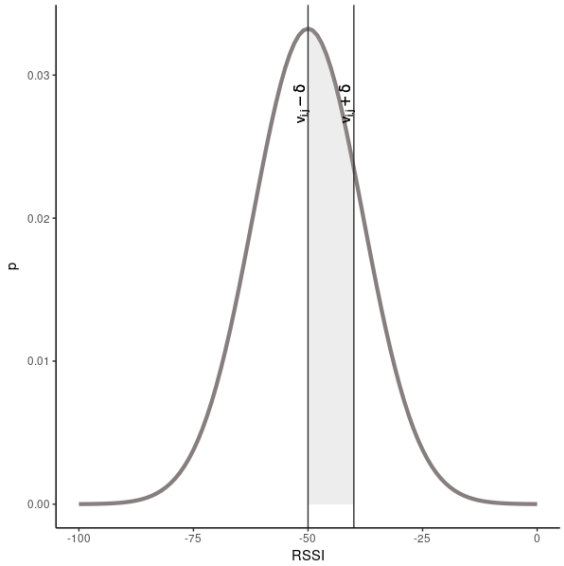


Figure 4: δ parameter for the probabilistic approach. This parameter sets the width of the discretization steps.

Thus, the initial collection of RSSI observations associated to a particular point is transformed into a pair of vectors containing the means and the standard deviations of the RSSI for each beacon, and then the complete training data is stored as a set of statistical parameters that can be used to infer the location of a test observation as the one that maximizes a probability function.

Let $\widehat{\mathcal{D}}_{train}$ be the new training set obtained from the previous procedure:

$$\widehat{\mathcal{D}}_{train} = \left\{ \widehat{\mathcal{F}}_{train},\ \widehat{\mathcal{L}}_{train} \right\}$$

$$\widehat{\mathcal{F}}_{train} = \left\{ \widehat{\lambda_1^{tr}}, \widehat{\lambda_2^{tr}}, ..., \widehat{\lambda_g^{tr}} \right\}$$

$$\widehat{\mathcal{L}}_{train} = \left\{ \widehat{\tau_1^{tr}}, \widehat{\tau_2^{tr}}, ..., \widehat{\tau_g^{tr}} \right\}$$

where $\widehat{\mathcal{F}}_{train}$ is the set of statistical parameters obtained from the fingerprints of the training set, g is the number of groups of fingerprints with the same associated position, and $\widehat{\mathcal{L}}_{train}$ is the set of positions associated to each group. Each one of the g observations of the training data set is now composed of a fingerprint with q values:

$$\widehat{\lambda_i^{tr}} = \left\{ \theta_{1,i}^{tr}, \theta_{2,i}^{tr}, ..., \theta_{q,i}^{tr} \right\},\ i \in [1, ..., g]$$

$$\theta_{h,i}^{tr} \sim \mathcal{N}(\mu_{h,i}, \sigma_{h,i}^2)$$

where $\mu_{h,i}$ and $\sigma_{h,i}^2$ are the mean and the variance, respectively, of the h^{th} RSSI of the i^{th} group of original fingerprints.

Let $\rho_{h,j}^{ts}$ be the h^{th} RSSI measurement of the j^{th} test fingerprint (λ_j^{ts}), and let $\mu_{h,i}$ and $\sigma_{h,i}^2$ be the mean and the standard deviation of the h^{th} beacon distribution obtained for the i^{th} position from the training set. The probability $p_{h,j}^{(i)}$, of observing $\rho_{h,j}^{ts}$ at the i^{th} position is:

$$p_{h,j}^{(i)} = \int_{\rho_{h,j}^{ts}-\delta}^{\rho_{h,j}^{ts}+\delta} \frac{1}{\sigma_{h,i}\sqrt{2\pi}}\, \mathrm{e}^{-\frac{x-\mu_{h,i}}{2\sigma_{h,i}^2}}\, dx$$

where δ is a parameter to allow the discretization of the normal distribution (Figure 4).

The set of all probabilities $p_{h,j}^{(i)},\ h \in [1, ..., q]$ obtained for a given test observation j, expresses the similarity between the observation measurement and the training data for a particular location. An evaluation of the total similarity for every location can be computed as a function of these individual probabilities, like its sum or its product. In the **ipft** package, this algorithm is implemented by the

`ipfProbabilistic` and `ipfEstimate` functions, and by default uses the sum of probabilities as default operator to evaluate the similarity:

$$\psi_j^{(i)} = \sum_{h=1}^{p} p_{h,j}^{(i)}$$

where $\psi_j^{(i)}$ is the similarity between the j^{th} test observation and the i^{th} distribution from the training data set. The function to evaluate the similarity can be passed to `ipfProbabilistic` as a parameter.

As well as the `ipfKnn` and `ipfProximity` funtions, `ipfProbabilistic` returns a `ipfModel` object with the same data structure seen in Section The `ipfKnn` function, but with the difference that now the `data` property returns the probabilistic parameters that define the fitted distributions for every group of fingerprints on the training set. The clustering or grouping of the training data is performed by default over the location data provided by the user, but this behavior can be customized by passing a parameter with the columns over which to group the data, or by passing the group indices directly. The `ipft` package implements two functions (`ipfGroup()` and `ipfCluster()`) to perform clustering tasks. These functions are described in Section Data clustering.

The signature of the `ipfProbabilistic` function is:

```
ipfProbabilistic <- function(train_fgp, train_pos, group_cols = NULL, groups = NULL,
                             k = 3, FUN = sum, delta = 1, ...)
```

where `train_fgp`, `train_pos` and `k` have the same meaning and structure as described in Section The `ipfKnn` function, and, given n observations in the training set:

- `groups`: is a numeric vector of length n, containing the index of the group assigned to each observation of the training set. This parameter is optional.
- `group_cols`: is a character vector with the names of the columns to use as criteria to form groups of fingerprints. This parameter is optional.
- `FUN`: is a function to estimate a similarity measure from the calculated probabilities.
- `delta`: is a parameter to specify the interval around the test RSSI value to take into account when determining the probability.
- `...`: are additional parameters for `FUN`.

The following code shows how to use the `ipfProbabilistic` function to obtain a probabilistic model from the `ipftrain` and `ipftest` data sets. The default behavior of `ipfProbabilistic` groups the training data attending at the position of each observation, in this case, its x and y coordinates:

```
> probModel <- ipfProbabilistic(tr_fingerprints, tr_positions, k = 7, delta = 10)
> head(probModel$data$positions)
     X     Y
1 -0.6 24.42
2 -0.6 27.42
3  0.0  0.00
4  0.4  0.00
5  0.4  3.38
6  0.4  6.81
```

Now the `ipfModel$data` property returns a list with 3 elements:

- `means`: a data frame with the means for every beacon and every group of fingerprints.
- `sds`: a data frame with the standard deviations for every beacon and every group of fingerprints.
- `positions`: a data frame with the position of each group of fingerprints.

To obtain an estimation from this model, the same code used in section The `ipfKnn` function can be used to produce the estimated locations:

```
> ts_fingerprints <- ipftest[, 1:168]
> ts_positions    <- ipftest[, 169:170]
> probEstimation  <- ipfEstimate(probModel, ts_fingerprints, ts_positions)
```

and their errors and its mean value:

```
> mean(probEstimation$errors)
[1] 6.069336
```

An alternative function can be passed to `ipfProbabilistic`. The following code uses the maximum value of the probabilities as the similarity measure, and passes a parameter to remove NAs from the data[7]:

```
> probModel      <- ipfProbabilistic(tr_fingerprints, tr_positions, k = 9, delta = 10,
+                                    FUN = max, na.rm = TRUE)
> probEstimation <- ipfEstimate(probModel, ts_fingerprints, ts_positions)
> mean(probEstimation$errors)
[1] 8.652321
```

The `ipfProximity` function.

When the location of the access points is known, it's possible to estimate the position of a fingerprint using the log-distance path loss model (Seybold, J.S., 2005). Given a set of q beacons, and a fingerprint vector $\lambda = \{\rho_1, \rho_2, ..., \rho_q\}$ of length q, this model is expressed as:

$$\rho_h = P_{1m,h} - 10\alpha \log_{10} d_h - \gamma, \;\; h \in [1, 2, ..., q]$$

where ρ_h is the value of the received signal from the h^{th} beacon, d_h is the distance from the observation to the beacon, $P_{1m,h}$ is the received power at 1 meter from the emitter, α is the path loss exponent, and $\gamma \sim \mathcal{N}(0, \sigma_\gamma^2)$ represents a zero mean Gaussian noise that models the random shadowing effects of the environment.

The estimator of the distance between the emitting beacon and the position where the signal is received is:

$$\hat{d}_h = 10^{\frac{\rho_h - P_{1m,h}}{10\alpha}}$$

This estimation follows a log-normal distribution that is:

$$\ln \hat{d}_h \sim \mathcal{N}(\ln d_h, \sigma_d^2)$$

where $\sigma_d = (\sigma_\gamma ln10)/(10\alpha)$.

The mean and the variance of the distribution are:

$$E[\hat{d}_h] = d_h \, e^{\sigma_d^2/2}$$
$$Var[\hat{d}_h] = d_h^2 \, e^{\sigma_d^2} \, (e^{\sigma_d^2} - 1)$$

Note that the variance grows quadratically with the distance, making the estimation less reliable as the distance becomes larger. Therefore, the distances estimated from different beacons will have different accuracies. To take this into account, the algorithm estimates the position of a fingerprint as a minimization problem of the overall squared error of the estimated distances. The objective function to minimize is:

$$\min_{\tau} J = \sum_{h=1}^{p} \omega_h (\hat{d}_h - \|s_h - \tau\|)^2$$

where τ is the position that minimizes the function, that is, the estimated position, q is the number of beacons present in the fingerprint, and $\omega_h = 1/Var[\hat{d}_h]$ are the weights.

The functions `ipfProximity` and `ipfEstimate` implement this design, and uses the Broyden-Fletcher-Goldfard-Shano algorithm (BFGS) (Broyden, 1969; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970), a quasi-Newton method, to minimize the previous function to make an estimation of the fingerprint position. The accuracy of the estimation is strongly dependent on the reliability of the emitters positions. When these positions are unknown, they can be estimated with the function `ipfEstimateBeaconPositions`. Section Beacon position estimation details the implementation and usage of this function. The `ipfProximity` function returns an `ipfModel` object with the data needed by the `ipfEstimate` function to estimate a fingerprint position.

The signature of the `ipfProximity` function is:

```
ipfProximity <- function(bpos, rssirange = c(-100, 0), norssi = NA, alpha = 5,
```

[7]The `ipfProbabilistic` function takes into account the NAs contained in the data when using the default function (sum), but the user needs to manage this situation when a custom function is provided. In this example, the data is not previously transformed, is passed as it is, with NAs for not detected WAPs, to illustrate this situation.

```
                                wapPow1 = -30)
```

where:

- `bpos`: a matrix or a data frame containing the position of the beacons, in the same order as they appear in fingerprints.
- `rssirange`: the range of the RSSI data in the fingerprints.
- `norssi`: the value used to represent a not detected beacon.
- `alpha`: the path loss exponent (α).
- `wapPow1`: a numeric vector with the received power at one meter distance from the beacon ($P_{1m,h}$). If only one value is supplied, it will be assigned to all beacons.

In the following example, the goal is to estimate the position of the 702 fingerprints included in the test set, using the known position of the WAPs and the log-distance path loss model. The `ipfpwap` dataset contains the location of 39 of the 168 wireless access points of the `ipftrain` and `ipftest` data sets. The `ipfProximity` function returns a model that is used to estimate the position of the fingerprints. As the real position of the test fingerprints is known, this information can be also passed to the `ipfEstimate` function. Thus, the returned `ipfEstimation` object will contain, along with the estimated positions, the associated errors:

```
> proxModel      <- ipfProximity(ipfpwap, alpha = 4, rssirange = c(-100, 0),
+                            norssi = NA, wapPow1 = -32)
> fingerprints   <- ipftest[, 1:168]
> positions      <- ipftest[, 169:170]
> proxEstimation <- ipfEstimate(proxModel, ipftest[, 1:168], ipftest[, 169:170])
> mean(proxEstimation$errors)
[1] 8.0444
```

Positioning algorithms comparison

In a classical fingerprint-based positioning system, the radio map is constructed in accordance to the positioning algorithm to be used in the online phase. The knn algorithm follows a deterministic approach that performs well in most cases, while the probabilistic method is based on the assumption that there is enough training data for each particular position to obtain reliable parameters to model a distribution for each signal at each survey location. As regards to the proximity algorithm, it is based on two assumptions; first, the ability to realistically simulate the propagation model of the signal, and second, the known positions of the emitter beacons. These conditions are not met in many scenarios, where changes in occupation, for example, modify the propagation model and thus the performance of the positioning system.

To illustrate the previous considerations, Table 1 shows the mean and the quartile errors in meters for the implemented algorithms, computed using the dataset included in the package. In this particular case, given the characteristics of the training data, knn performs better than the rest.

		Quartile error (m)				
algorithm	mean error (m)	0%	25%	50%	75%	100%
knn	3.3027	0.15172	1.46891	2.61281	4.08992	19.84650
probabilistic	6.0693	0.14289	3.26988	5.63051	8.19933	17.93031
proximity	8.0444	2.49865	5.71055	7.42602	9.88427	20.12029

Table 1: Comparison of the algorithms' accuracy on the dataset included in the package

To compare the performance of the proposed implementation of the previous positioning algorithms, we ran a benchmark test of 1000 iterations on each function, using the dataset included in the package. The results for the model fitting functions are shown in Table 2. As it can be seen, the proximity and knn algorithms are the fastest, as expected, since their model fitting process basically consists in storing the training data for later processing during the estimation stage. In contrast, the probabilistic algorith has to fit a normal distribution for each signal received at each position, and thus, it takes longer to complete the process.

The outcomes are different when considering the results for the estimation function (Table 3). The position estimation for the probabilistic algorithm is faster that the rest. For the knn algorithm, the estimation process could be improved using clustering techniques to avoid comparing the test fingerprint with all the instances in the training set. With regards to the estimation process for the

function	elapsed (sec)	relative
ipfKnn	0.031	1.409
ipfProbabilistic	1035.446	47065.727
ipfProximity	0.022	1.000

Table 2: Performance comparison of the model building functions

proximity algorithm, the fact that the result is computed by solving an unconstrained nonlinear optimization through an iterative method highly penalyzes its performance.

model	function	elapsed (sec)	relative
knn	ipfEstimate	2508.079	2.998
probabilistic	ipfEstimate	836.651	1.000
proximity	ipfEstimate	28259.110	33.776

Table 3: Performance comparison of the estimation functions on each model

Beacon position estimation

If the actual position of the beacons is unknown, it can be estimated in many ways from the RSSI data. Two basic methods for estimation of the beacons location have been included in the `ipft` package through the `ipfEstimateBeaconPositions` function. The 'centroid' and the 'weighted centroid' methods.

Both methods use the fingerprint data to guess the position of the beacons. Let q be the number of beacons and $\tau^{\mathcal{B}}$ be the set of beacons locations:

$$\tau^{\mathcal{B}} = \left\{ \nu^{\mathcal{B}}_{1,h}, \nu^{\mathcal{B}}_{2,h}, \nu^{\mathcal{B}}_{3,h} \right\},\ h \in [1, 2, ..., q]$$

the position of the h^{th} beacon is given by:

$$\tau^{\mathcal{B}}_h = \left\{ \sum_{i=1}^{n} \omega_i \nu^{tr}_{1,i},\ \sum_{i=1}^{n} \omega_i \nu^{tr}_{2,i},\ \sum_{i=1}^{n} \omega_i \nu^{tr}_{3,i} \right\}$$

where n is the number of fingerprints in the training set. The value of ω_i is:

$$\omega_i = \frac{1}{n}$$

for the 'centroid' method and:

$$\omega_i = \frac{\rho^{tr}_{h,i}}{\sum_{l=1}^{n} \rho^{tr}_{h,l}}$$

for the 'weighted centroid' method. Since the biggest weights have to be assigned to the strongest RSSI values, the fingerprint vector values should be positive, or at least, positively correlated to the beacon received intensity. This is checked by the function implementation so the input data is internally transformed to a positive range when needed.

This is the signature of the `ipfEstimateBeaconPositions` function:

```
ipfEstimateBeaconPositions <- function(fingerprints, positions, method = 'wcentroid',
                                       rssirange = c(-100, 0), norssi = NA)
```

where:

- `fingerprints`: is a data frame with the fingerprint vectors as rows.
- `positions`: a data frame with the position of the fingerprints.
- `method`: the method to use by the algorithm, either 'centroid' or 'wcentroid'.
- `rssirange`: the range of the signal strength values of the fingerprints.
- `norssi`: the value assigned in the fingerprints to a non detected beacon.

The following code uses the function `ipfEstimateBeaconPositions` with the 'weighted centroid' method to estimate the position of the wireless access points, under the assumption that this position is unknown. Finally, the function `ipfProximity` estimates the positions of the first 6 test fingerprints:

```
> bc_positions   <- ipfEstimateBeaconPositions(ts_fingerprints, ts_positions,
                                               method = 'wcentroid')
> proxModel      <- ipfProximity(bc_positions, rssirange = c(0.1, 1),
+                                norssi = NA)
> proxEstimation <- ipfEstimate(proxModel, fingerprints[1:6,],
+                               positions[1:6,])
> proxEstimation$location
      V1       V2
1 1.686950 12.02117
2 1.686950 12.02117
3 1.654255 10.91767
4 1.682121 10.96035
5 1.711448 10.88966
6 1.695007 10.09507
```

Data clustering

Clustering techniques can be used with the aim of enhancing localization performance and reducing computational overhead (Cramariuc et al., 2016b). The `ipft` package includes some functions for cluster analysis and grouping of the fingerprinting and location data. These functions can be used to create or detect clusters based on the position of the observations, on its signal levels, or on any other criteria that might be useful to group the data by. Performing RSSI clustering before the positioning process groups a large number of reference points into various clusters that can be used to perform first-level classification. This allows to assess the testing point location by using only the fingerprints in the matched cluster rather than the whole radio map. Furthermore, given the amplitude atenuation that building partitions cause to electromagnetic signals, clusters usually can be related to physical spaces such as buildings, floors or even rooms.

The main function for clustering tasks is `ipfCluster`. The more basic usage of the function takes the provided data and uses the k-means algorithm to classify it into k disjoint sets of observations, by selecting a set of k cluster centers to minimize the sum of the squared distances between the data vectors and their corresponding centers.

The k-means clustering procedure begins with an initial set of randomly selected centers, and iteratively tries to minimize the sum of the squared distances. This makes the algorithm very sensitive to the arbitrary selection of initial centers, and introduces variability in the results obtained from one execution to another. Besides, the number of clusters has to be established beforehand, and that may be inconvenient in some scenarios.

The signature of the `ipftCluster` function is:

```
ipfCluster <- function(data, method = 'k-means', k = NULL, grid = NULL, ...)
```

where

- `data`: is a data frame with the data to cluster. When using the *k-means* method, the data frame must not contain any `NA` values.
- `method`: the algorithm used to create clusters. The implemented algorithms are 'k-means' for k-means algorithm, 'grid' for clustering based on spatial grid partition, and 'AP' for affinity propagation algorithm.
- `k`: a numeric parameter for k-means algorithm.
- `grid`: a numeric vector with the size of the grid for the grid algorithm.

When using the default k-means algorithm, the function behaves as a wrapper around the k-means function of the **stats** package, and therefore, the usage can be further customized by passing extra parameters, as the number of iterations or the algorithm to be used ("Hartigan-Wong" is the default).

The following example will find $k = 30$ clusters of similar fingerprints in the `ipftrain` dataset. First the data set of fingerprints is transformed to eliminate the `NA` values that represent a not detected beacon. Then, the data is passed to the `ipfCluster` function to find the 30 clusters using the 'MacQueen' algorithm:

```
> set.seed(1)
> cl_fingerprints <- ipfTransform(tr_fingerprints, inNoRSSI = NA, outNoRSSI = 0)
> clusterData     <- ipfCluster(cl_fingerprints, k = 30, iter.max = 20,
+                               algorithm = "MacQueen")
> head(clusterData$clusters)
[1] 3 3 3 3 3 3
```

The outcome of the `ipfCluster` function is a list containing the indices of the k clusters and its centroids. Given the previous example, `clusterData$centers` will return the k centroids, and `clusterData$clusters` will return the cluster index $i \in [1, .., k]$ for every observation in `ipftrain`.

The `ipfCluster` function includes an implementation of the affinity propagation (AP) algorithm (Frey and Dueck, 2007) that can be used to estimate the number of distinct clusters present in the radio map. AP does not require the number of clusters to be determined before running it. It finds members of the input set, known as 'exemplars', that are representative of clusters by creating the centers and the corresponding clusters based on the constant exchanging of reading similarities between the observations. This message-passing process continues until a good set of centers and corresponding clusters emerges.

The following code uses AP clustering to find groups of similar RSSI vectors from the `ipftrain` data set. With no further parametrization, it will classify the RSSI data into 43 distinct clusters:

```
> clusterData     <- ipfCluster(tr_fingerprints, method = 'AP')
> dim(clusterData$centers)
[1]  43 168
```

Now, `clusterData$centers` holds the 43 'exemplars', those RSSI vectors from the radio map that are representative of a cluster, and `clusterData$clusters` contains the indices that link every observation of the data set with its assigned cluster.

To perform a more simple grouping based on a precise set of variables, the `ipfGroup` function provides a method to group the data by column name. The function signature is:

```
ipfGroup <- function(data, ...)
```

where

- `data`: is a data frame with the data to group.
- `...`: The variables to group the data by.

The `ipfGroup` function returns a numeric vector with the same length as the number of observations contained in the `data` data frame, containing the index of the group assigned to each observation. The following example groups the data according to the position of the observations, that in the `ipftrain` and `ipftest` datasets are represented by the columns 'X' and 'Y':

```
> groups <- ipfGroup(ipftrain, X, Y)
> head(groups)
[1]  4  4  4  4 22 22
> length(unique(groups))
[1]  41
```

Plotting functions

Indoor positioning generally involves statistical analysis of datasets, and the **ipft** provides some useful functions to produce graphs for exploring data. All the graphic functions included in the package are built upon the **ggplot2** package (Wickham, 2011), and return a `ggplot` object that can be plotted or further personalized with custom labels, theme, etc.

The `ipfPlotPdf` and the `ipfPlotEcdf` have already been introduced in Section The `ipfKnn` function. These functions will plot the probability density function and the empirical cumulative distribution function, respectively. Both functions take an `ipfEstimation` object to produce the plot, while the axis labels and plot tittle can be also supplied by the parameters `xlab`, `ylab` and `tittle`. Their respective signatures are:

```
ipfPlotPdf <- function(estimation, xlab = 'error', ylab = 'density',
                       title = 'Probability density function')

ipfPlotEcdf <- function(estimation, xlab = 'error',
```

```
                        ylab = 'cumulative density of error',
                        title = 'Empirical cumulative density function')
```

The function `ipfPlotLocation` will produce a plot of the location of the data. The following code shows its signature and presents an example of its use. The example calls the function with parameter `plabel` set to `TRUE`, to plot labels identifying each location, and `reverseAxis` set to `TRUE` to swap the axis. It also modifies the resulting object by changing the default **ggplot2** theme to the white one. The result is shown in Figure 5.

```
ipfPlotLocation <- function(positions, plabel = FALSE, reverseAxis = FALSE,
                            xlab = NULL, ylab = NULL, title = '')

library(ggplot2)
ipfPlotLocation(ipftrain[, 169:170], plabel = TRUE, reverseAxis = TRUE) + theme_bw()
```

The function `ipfPlotEstimation` plots the estimated position of the test observations based on an `ipfModel` object and an `ipfEstimation` object, as well as the actual position (parameter `testpos`), if known, and the position of the k selected fingerprints from the training set used to guess its location (parameter `showneighbors`). The green dots indicate the actual position of the observations, while the black dots indicate the estimated ones. The blue lines connect the estimated positions with the k neighbors from which the location has been estimated, and the red arrows connect the actual position of the fingerprint with the estimated one. The following code shows the function signature and provides an example of its usage. The result plot is shown in Figure 6:

```
ipfPlotEstimation <- function(model, estimation, testpos = NULL, observations = c(1),
                              reverseAxis = FALSE, showneighbors = FALSE,
                              showLabels = FALSE, xlab = NULL, ylab = NULL,
                              title = '')

library(ggplot2)
probModel <- ipfProbabilistic(ipftrain[, 1:168], ipftrain[, 169:170])
probEst   <- ipfEstimate(probModel, ipftest[, 1:168], ipftest[, 169:170])
ipfPlotEstimation(probModel, probEst, ipftest[, 169:170],
                  observations = c(61:62, 81:82), reverseAxis = TRUE,
                  showneighbors = TRUE, showLabels = TRUE) + theme_bw()
```

Summary

In this paper, the package **ipft** is presented. The main goal of the package is to provide researchers with a set of functions to manipulate, cluster, transform, create models and make estimations using indoor localization fingerprinting data. This package enables researchers to use a well established set of algorithms and tools to manipulate and model RSSI fingerprint data sets, and also allows them to customize the included algorithms with personalized parameters and functions to adapt the working mode to their particular research interests.

In this work some of the fundamental algorithms used in indoor fingerprinting localization techniques have been formally presented and illustrated, while detailed examples and information about its usage and implementation have been provided.

Future work

This package is an ongoing work, and future versions will implement new algorithms and tools with the aim of providing a base framework for researchers, and become a reference library for fingerprinting-based indoor positioning research.

In particular, future lines of work should consider the implementation of deep learning based algorithms. Many deep learning techniques can be exploited to try to obtain better positioning performance. Recurrent neural networks could be used to learn not only spatial but also temporal patterns of the received signals. Deep autoencoders can be implemented as a way to encode fingerprints and reduce their dimensionality to a few number of significant features. Their variational and generative extensions can be of use to better model the stochastic nature of RSSI data. These models can also be applied to generate new training data for deep learning-based clasisifiers, increasing the robustness of positioning systems and trying to address problems caused by heterogeneity of devices.

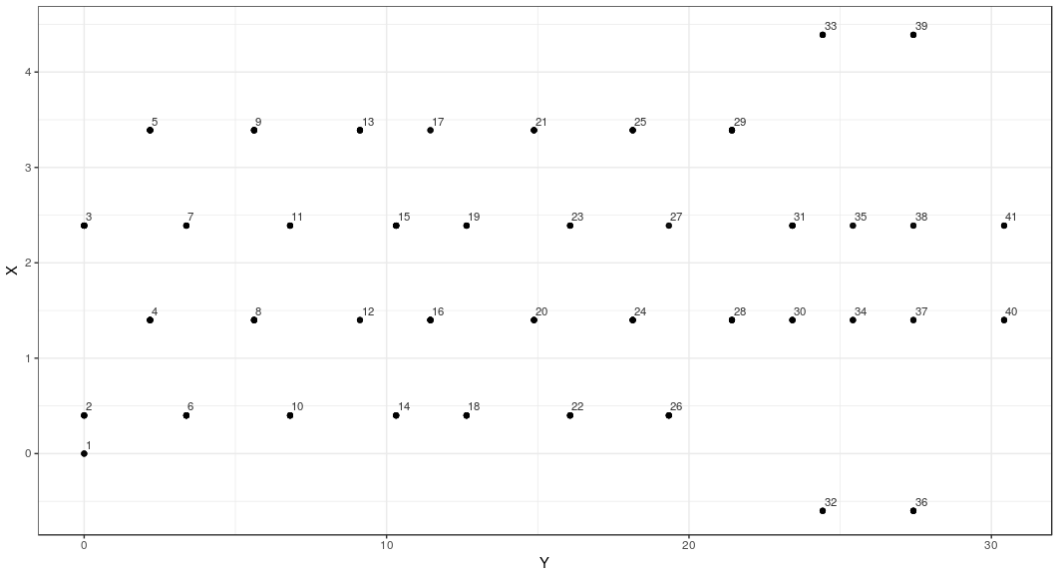


Figure 5: Location of fingerprints included in the `ipftrain` data frame. The labels indicate the group indices.

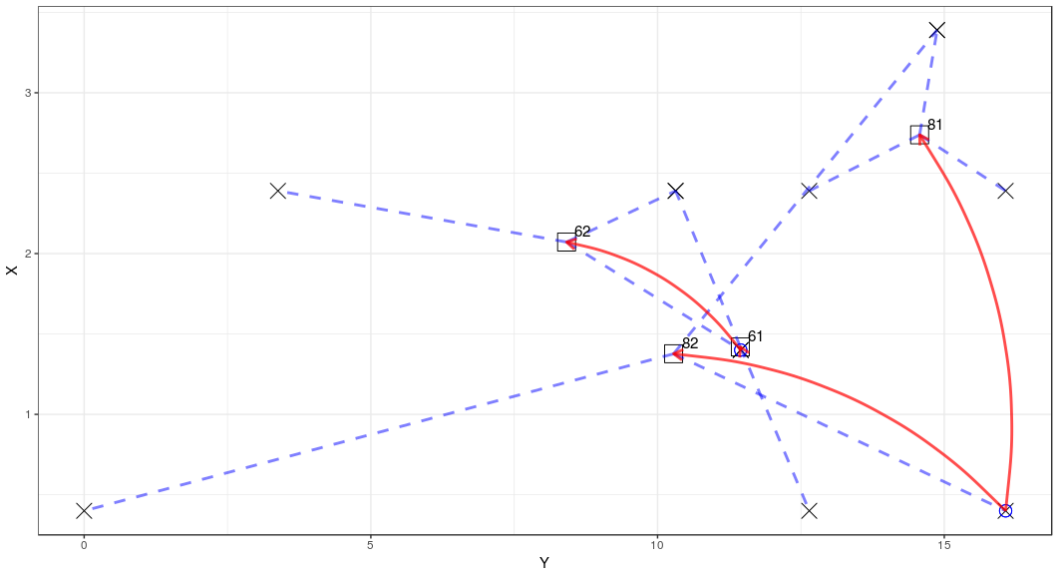


Figure 6: Estimated and actual positions of test observations 61, 62, 81 and 82 from the `ipftrain` data set. The circles indicate the actual positions of the observations. The squares show the estimated positions. The red arrows connect the actual positions with the estimated ones. The dashed lines connect the estimated positions with the k neighbors from which the location has been estimated, represented by the crosses.

Acknowledgements

The authors would like to thank the two anonymous reviewers for providing useful feedback that helped to improve the paper.

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness through the "Proyectos I + D Excelencia" programme (TIN2015-70202-P) and by Jaume I University "Research promotion plan 2017" programme (UJI-B2017-45).

Bibliography

E. Aarts and R. Wichert. Ambient intelligence. In *Technology Guide*, pages 244–249. Springer-Verlag, 2009. URL https://doi.org/10.1007/978-3-540-88546-7_47. [p268]

A. H. Ali, M. R. A. Razak, M. Hidayab, S. A. Azman, M. Z. M. Jasmin, and M. A. Zainol. Investigation of Indoor WIFI Radio Signal Propagation. In *Proceedings of the Symposium on Industrial Electronics and Applications, (ISIEA'10)*, pages 117–119, 2010. URL https://doi.org/10.1109/isiea.2010.5679486. [p277]

C. Broyden. A new double-rank minimisation algorithm. preliminary report. In *Notices of the American Mathematical Society*, volume 16, page 670. American Mathematical Society 201 Charles ST, Providence, RI 02940-2213, 1969. [p281]

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. URL https://doi.org/10.1109/tit.1967.1053964. [p271]

A. Cramariuc, H. Huttunen, and E. S. Lohan. Clustering benefits in mobile-centric wifi positioning in multi-floor buildings. In *2016 International Conference on Localization and GNSS (ICL-GNSS)*, pages 1–6. IEEE, 2016a. URL https://doi.org/10.1109/icl-gnss.2016.7533846. [p272]

A. Cramariuc, H. Huttunen, and E. S. Lohan. Clustering Benefits in Mobile-Centric WiFi Positioning in Multi-Floor Buildings. In *Proceedings of the 6th International Conference on Localization and GNSS (ICL-GNSS'16)*, pages 1–6, 2016b. URL https://doi.org/10.1109/icl-gnss.2016.7533846. [p271, 284]

R. Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970. [p281]

B. J. Frey and D. Dueck. Clustering by Passing Messages between Data Points. *Science*, 315(5814): 972–976, 2007. URL https://doi.org/10.1126/science.1136800. [p285]

T. Gigl, G. J. M. Janssen, V. Dizdarevic, K. Witrisal, and Z. Irahhauten. Analysis of a uwb indoor positioning system based on received signal strength. In *Proceedings of the 4th Workshop on Positioning, Navigation and Communication (PNC'07)*, pages 97–101, 2007. URL https://doi.org/10.1109/wpnc.2007.353618. [p269]

D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970. URL https://doi.org/10.1090/s0025-5718-1970-0258249-6. [p281]

A. Haeberlen, E. Flannery, A. M. Ladd, A. Rudys, D. S. Wallach, and L. E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom'04)*, pages 70–84, 2004. URL https://doi.org/10.1145/1023720.1023728. [p271, 277]

A. B. Harbicht, T. Castro-Santos, W. R. Ardren, D. Gorsky, and D. J. Fraser. Novel, continuous monitoring of fine-scale movement using fixed-position radiotelemetry arrays and random forest location fingerprinting. *Methods in Ecology and Evolution*, 8(7):850–859, 2017. URL https://doi.org/10.1111/2041-210x.12745. [p268]

S. He and S. H. G. Chan. Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons. *IEEE Communications Surveys & Tutorials*, 18(1):466–490, 2016. URL https://doi.org/10.1109/comst.2015.2464084. [p268]

N. E. Klepeis, W. C. Nelson, W. R. Ott, J. P. Robinson, A. M. Tsang, P. Switzer, J. V. Behar, S. C. Hern, and W. H. Engelmann. The national human activity pattern survey (nhaps): a resource for assessing exposure to environmental pollutants. *Journal Of Exposure Analysis And Environmental Epidemiology*, 11:231 EP –, 2001. URL https://doi.org/10.1038/sj.jea.7500165. [p268]

B. Li, J. Salter, A. Dempster, and C. Rizos. Indoor positioning techniques based on wireless lan. In *Proceedings of the 1st IEEE International Conference on Wireless Broadband and Ultra Wide-Band Communications (AusWireless'06)*, pages 13–16, 2006. URL https://opus.lib.uts.edu.au/bitstream/2100/170/1/113_Li.pdf. [p269]

H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 37(6): 1067–1080, 2007. URL https://doi.org/10.1109/tsmcc.2007.905750. [p268, 276]

Y. Liu, H. Du, and Y. Xu. The research and design of the indoor location system based on rfid. In *Proceedings of the 4th International Symposium on Computational Intelligence and Design (ISCID'11)*, pages 87–90, 2011. URL https://doi.org/10.1109/iscid.2011.123. [p269]

E. S. Lohan, K. Koski, J. Talvitie, and L. Ukkonen. WLAN and RFID Propagation Channels for Hybrid Indoor Positioning. In *Proceedings of the 4th International Conference on Localization and GNSS, (ICL-GNSS'14)*, 2014. URL https://doi.org/10.1109/icl-gnss.2014.6934184. [p274]

J. Luo and X. Zhan. Characterization of Smart Phone Received Signal Strength Indication for WLAN Indoor Positioning Accuracy Improvement. *Journal of Networks*, 9(3):739–746, 2014. URL https://doi.org/10.4304/jnw.9.3.739-746. [p277]

A. Popleteev, V. Osmani, O. Mayora, and A. Matic. Indoor localization using audio features of fm radio signals. In *International and Interdisciplinary Conference on Modeling and Using Context*, pages 246–249. Springer, 2011. URL https://doi.org/10.1007/978-3-642-24279-3_26. [p268]

Y. Quan, L. Lau, F. Jing, Q. Nie, A. Wen, and S.-Y. Cho. Analysis and machine-learning based detection of outlier measurements of ultra-wideband in an obstructed environment. In *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 997–1000. IEEE, 2017. URL https://doi.org/10.1109/indin.2017.8104909. [p268]

Research and markets. Indoor location market by component, deployment mode, application, vertical and region - global forecast to 2022. *Research and markets*, 2017. URL https://www.researchandmarkets.com/reports/4416241/indoor-location-market-by-component-deployment. [p268]

T. Roos, P. Myllymäki, H. Tirri, P. Misikangas, and J. Sievänen. A Probabilistic Approach to WLAN User Location Estimation. *International Journal of Wireless Information Networks*, 9(3):155–164, 2002. URL https://doi.org/10.1023/a:1016003126882. [p271]

E. Sansano. *ipft: Indoor Positioning Fingerprinting Toolset*, 2017. URL https://cran.r-project.org/web/packages/ipft/index.html. [p268]

Seybold, J.S. *Introduction to RF Propagation*. John Wiley & Sons, 2005. [p272, 281]

D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970. URL https://doi.org/10.1090/s0025-5718-1970-0274029-x. [p281]

S. Shrestha, J. Talvitie, and E. S. Lohan. On the Fingerprints Dynamics in WLAN Indoor Localization. In *Proceedings of the 13th International Conference on ITS Telecommunications (ITST'13)*, pages 122–126, 2013. URL https://doi.org/10.1109/itst.2013.6685532. [p274]

J. Torres-Sospedra, R. Montoliu, A. Martinez-Uso, J. P. Avariento, T. J. Arnau, M. Benedito-Bordonau, and J. Huerta. UJIIndoorLoc: A New Multi-Building and Multi-Floor Database for WLAN Fingerprint-Based Indoor Localization Problems. In *Proceedings of the 5th International Conference on Indoor Positioning and Indoor Navigation (IPIN'14)*, pages 261–270, 2015a. URL https://doi.org/10.1109/ipin.2014.7275492. [p272]

J. Torres-Sospedra, R. Montoliu, S. Trilles, Óscar Belmonte, and J. Huerta. Comprehensive Analysis of Distance and Similarity Measures for Wi-Fi Fingerprinting Indoor Positioning Systems. *Expert Systems with Applications*, 42(23):9263–9278, 2015b. URL https://doi.org/10.1016/j.eswa.2015.08.013. [p271, 273, 274]

Y. Wang, X. Yang, Y. Zhao, Y. Liu, and L. Cuthbert. Bluetooth positioning using rssi and triangulation methods. In *Proceedings of the 10th IEEE Consumer Communications and Networking Conference, (CCNC'13)*, pages 837–842, 2013. URL https://doi.org/10.1109/ccnc.2013.6488558. [p269]

W. Werner, J. Rabaey, and E. H. L. Aarts, editors. *Ambient Intelligence*. Springer-Verlag, 2005. ISBN 978-3-540-27139-0. URL https://doi.org/10.1007/b138670. [p268]

H. Wickham. ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2):180–185, 2011. URL https://doi.org/10.1002/wics.147. [p285]

J. Xiao, Z. Zhou, Y. Yi, and L. M. Ni. A Survey on Wireless Indoor Localization from the Device Perspective. *ACM Computing Surveys*, 49(2):1–31, 2016. URL https://doi.org/10.1145/2933232. [p269]

C. Yang, T. Nguyen, and E. Blasch. Mobile Positioning via Fusion of Mixed Signals of Opportunity. *IEEE Aerospace and Electronic Systems Magazine*, 29(4):34–46, 2014. URL https://doi.org/10.1109/maes.2013.130105. [p270]

J. yub Lee, C. hwan Yoon, H. Park, and J. So. Analysis of location estimation algorithms for wifi fingerprint-based indoor localization. In *Proceedings of the 2nd International Conference on Software Technology (SoftTech'13)*, pages 89–92, 2013. [p268]

Emilio Sansano
Institute of New Imaging Technologies
Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12017 Castelló de la Plana
Spain
esansano@uji.es

Raúl Montoliu
Institute of New Imaging Technologies
Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12017 Castelló de la Plana
Spain
montoliu@uji.es

Óscar Belmonte
Institute of New Imaging Technologies
Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12017 Castelló de la Plana
Spain
oscar.belmonte@uji.es

Joaquín Torres-Sospedra
Institute of New Imaging Technologies
Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12017 Castelló de la Plana
Spain
jtorres@uji.es

roahd Package: Robust Analysis of High Dimensional Data

by Francesca Ieva, Anna Maria Paganoni, Juan Romo and Nicholas Tarabelloni

Abstract The focus of this paper is on the open-source R package **roahd** (**RO**bust **A**nalysis of **H**igh dimensional **D**ata), see Tarabelloni et al. (2017). **roahd** has been developed to gather recently proposed statistical methods that deal with the robust inferential analysis of univariate and multivariate functional data. In particular, efficient methods for outlier detection and related graphical tools, methods to represent and simulate functional data, as well as inferential tools for testing differences and dependency among families of curves will be discussed, and the associated functions of the package will be described in details.

Introduction

Functional Data Analysis (FDA) has seen an impressive growth in the statistical research due to the more and more frequent production of complex data in many different research contexts (i.e., healthcare, environmental, engineering, etc.). According to the FDA model, data can be seen as measurements of a certain quantity (or a set of quantities) along a given, independent and continuous indexing variable (such as time or space). Observations are then treated as random functions and can be viewed as trajectories of stochastic processes defined on a given infinite dimensional functional space. In this context "high dimensional data" is meant in this sense: a high number of covariates/predictors (e.g., evaluations of a signal on a given grid) for a single sample unit (e.g. signal). We have to face the traditional "large p, small n" problem: the number of features can exceed the number of observations. Many research areas deal with this kind of data where features exceed observations, for example, biomedical signals, high resolution imaging, website analysis of stream data.

The research in FDA dates back to 1970s - 1980s. However, the first edition of Ramsay and Silverman (2005) and Ramsay and Silverman (2002) made the methods available to a larger audience with an enormous impact on the spread of this topic. The authors mainly cover exploratory methods, parametric and semi-parametric approaches. Other important books on functional data analysis are Ferraty and Vieu (2006), Horvath and Kokoszka (2012) and Kokoszka and Reimherr (2017). In addition to these monographs there is a vast quantity of scientific papers ranging from theoretical to applied techniques aimed at modelling and analysing functions.

In R ecosystem, the number of packages focused on general functional data analysis is rapidly increasing. In particular, **fda** (Ramsay et al. (2014)) presents functions to implement many methods of functional data analysis, including smoothing, plotting and regression models (see Ramsay and Silverman (2005), Ramsay et al. (2009)). The package **fda.usc** (Febrero-Bande and Oviedo de la Fuente (2012)) carries out exploratory and descriptive analysis of functional data such as depth measurements or functional outliers detection, as well as functional regression models (univariate, nonparametric), basis representation and Functional Principal Component Analysis (FPCA). The package **fdasrvf** (Tucker (2017)) performs alignment, FPCA, and modeling of univariate and multivariate functions, allowing for elastic analysis of functional data through phase and amplitude separation. The core of the package **fdapace** (Dai et al. (2018)) is FPCA for sparsely or densely sampled random trajectories and time courses, via the Principal Analysis by Conditional Estimation (PACE) algorithm or numerical integration. The package **rainbow** (Shang and Hyndman (2019)) provides tools for functional data display, exploratory analysis (plots, bagplots and boxplots) and outlier detection, while the package **fds** (cite(fds) contains 19 data sets with functional data. Other packages focus on more specific methods for functional data analysis, like regression, classification and clustering, registering and aligning, studying time series of functional data (see Zeileis (2005))

The focus of this paper is on the package **roahd** (**RO**bust **A**nalysis of **H**igh dimensional **D**ata), Tarabelloni et al. (2017). It has been developed to gather recently proposed statistical methods that deal with the robust statistical analysis of univariate and multivariate functional data. The latter is the case where each observation in a dataset is a set of possibly correlated functions, measured at discrete points. Despite the usefulness of robust statistics methods in data analysis (e.g., median, quantiles, trimmed means), their generalisation to the functional framework is not straightforward, due to the infinite-dimensional nature of the spaces embedding data. A possibility is to leverage on the concept of depth measures in order to create proper order statistics to be used in a suitable robust inferential framework.

In the multivariate context, there are many possible definitions of depth measures. Among others, see Tuckey (1975); Liu and Singh (1993); Liu et al. (1999); Zuo and Serfling (2000). For univariate

and multivariate functional data, two main approaches to the generalisation of depth measures have been considered so far. A first approach is to average a multivariate depth, say D_L, defined on $\mathbb{R}^L$ and computed at each point over the domain interval $I \subset \mathbb{R}$ using suitable weights (see Claeskens et al. (2014); Lopez-Pintado et al. (2014) and references therein). Without loss of generality we will refer to I as a compact interval representing the time domain where the (multivariate) functional data are defined. A second approach, that is followed in **roahd**, extends the notion of univariate (i.e. $L = 1$) functional band depth (introduced in López-Pintado and Romo (2007, 2009)) to the multivariate framework (i.e., $L > 1$). In particular, in Ieva and Paganoni (2013) the depth of each component of the multivariate functional data is defined as a measure of the time each curve spends within the envelope generated by any other family of curves. Then the global depth is computed weighing in a suitable way the contributions of each component of the multivariate signal. This approach is based on the intrinsic functional nature of the data since it looks at the position of the entire function with respect to envelopes generated by families of functions.

The main contributions of the package, described and detailed in the following sections, are

(a) the implementation of **S3** classes representing functional data (`fData` and `mfData` for the univariate and multivariate case, respectively), as well as a set of algebraic operations and convenience functions to expressively operate on them;

(b) the implementation of useful generators for functional data, such as `generate_gauss_fdata` and `generate_gauss_mfdata`. These can be used to simulate artificial datasets of Gaussian functional data with a target mean and covariance, which must be specified by the researcher as arguments of the related functions and could be very useful to test or illustrate existing and new methods;

(c) the implementation of efficient functions for computing depth measures and robust statistics, such as `MBD`, `MEI`, `median_mfData`, `cor_spearman`, for both univariate and multivariate functional data allowing rank observations from the center of the distribution-outward and down-upward/up-downard with respect to sample measurements;

(d) the implementation of graphical methods, like the functional boxplot (`fbplot`) and the outliergram (`outliergram`). These can be employed to carry out an explorative analysis of a functional dataset, and to robustify it by discarding shape, magnitude and covariance outliers.

Robust methods for functional data are generally rather computationally intensive, thus the package's functions have been implemented with an attention to computational efficiency, in order to allow the processing of realistic datasets.

The paper is structured as follows: in Section Representation of Functional Data we introduce the methods and the functions to represent and to generate functional data; in Section Robust Statistics we describe the robust statistics and indexes implemented in **roahd**; in Section Graphical tools the main graphical tools for outlier detection are detailed, and finally Section Conclusions contains discussion, conclusion and further potential developments.

Representation of Functional Data

The **S3** classes `fData` and `mfData` implement a simple and compact representation of univariate and multivariate functional datasets. They can be used by specifying, for each observation in the functional dataset, a set of measurements over a discrete grid, representing the dependent variable indexing the functional data (e.g., time). If we denote by $I = [t_0, t_1, \ldots, t_{P-1}]$ an evenly spaced grid ($t_j - t_{j-1} = h > 0 \; \forall j = 1, \ldots, P-1$) and imagine to deal with a dataset $D_{i,j} = X_i(t_j), \forall i = 1, \ldots, N$ and $\forall j = 0, \ldots, P-1$, the object `fData` requires the evenly spaced grid over which the functional observations are measured as parameter `grid`, and the values of the observations in the functional dataset, provided in form of a two dimensional data structure (e.g., matrix or array) having as rows the observations and as columns their measurements over the grid of length P `values`. When the constructor of the class is created, it checks that the grid is actually evenly spaced.

An example of the function's call is the following

```
grid <- seq(0, 1, length.out = 100)
values <- matrix(c( sin(2 * pi * grid),
cos(2 * pi * grid),
4 * grid * (1 - grid),
tan(grid),
log(grid)),
nrow = 5, ncol = 100, byrow = TRUE)
fD <- fData(grid, values)
plot(fD, main = 'Univariate FD', xlab = 'time [s]', ylab = 'values', lwd = 2)
```

In particular the number of rows is the sample size, i.e. the number of statistical units and each statistical unit is a function evaluated in the grid of points of length P. In the artificial example above we have 5 curves evaluated on a evenly spaced grid of 100 points. The resulting plot is shown in Figure 1.

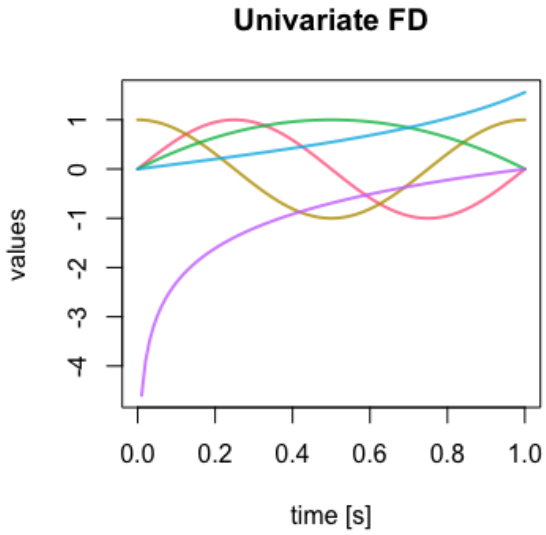


Figure 1: An example of a `fData` object, with 5 curves evaluated on a evenly spaced grid of 100 points.

An `mfData` object, instead, implements a multivariate functional dataset where each component is defined over the same indexing variable. In practice, we deal with a discrete grid $[t_0, t_1, \ldots, t_{P-1}]$ and a dataset of N statistical units, each one having L components observed over the same discrete grid: $D_{i,j,k} = X_{i,k}(t_j)$, $\forall i = 1, \ldots, N$, $\forall j = 0, \ldots, P-1$ and $\forall k = 1, \ldots, L$. The object `mfData` requires the evenly spaced grid of definition as parameter `grid` and a list containing the L components of the multivariate functional dataset, defined as 2D data structures (analogously to the constructor of `fData` class).

The **S3** implementation allows to enrich the package with expressive operations that enable an easy manipulation of datasets, keeping at the same time a light structure that allow users to easily access the inner state of objects. For instance, we added an overloaded operator `[.fData` (`[.mfData`) that allows one to use standard slices of `matrix` and `array` classes also for `fData` (`mfData`). We provided an overloaded implementation of the four basic algebraic operations, `+.fData`, `-.fData`, `*.fData`, `/.fData`, that allow one to write and evaluate simple expressions on `fData` objects without explicitly carry them out on the set of measurements. `+.fData`, `-.fData` support the sum or subtraction of compatible functional datasets (e.g, same definition grid I and same sample size N) or perform the element-wise (along the measurement direction, i.e., columns) sum or subtraction by a one-dimensional vector of compatible measurements (length P). The latter is useful, for instance, in order to translate or center data. The `*.fData`, `/.fData` operators perform an element-wise multiplication or division by a scalar quantity. We also added two convenience functions, `append_fData` and `append_mfData`, that can be used to concatenate two compatible (same definition grid I) functional datasets together.

Statistics functions for the computation of the mean (specification of the generic `mean` function of **R**), the median (through `median_fData` and `median_mfData`, see Section Robust Statistics for more details), or the covariance (`cov fun` and its specifications for `fData` and `mfData`), are also implemented for `fData` and `mfData`.

Finally, we implemented dedicated specialisations for the visualisation of functional data in `plot.fData` (`plot.mfData`). In case of `mfData` the graphical window is split into a rectangular lattice so that each component is plotted singularly. The rectangular frame has $\lfloor \sqrt{L} \rfloor$ rows and $\lceil \frac{L}{\lfloor \sqrt{L} \rfloor} \rceil$ columns. To give an example of this visualization method, consider the `mfD_healthy` dataset in the **roahd** package (see Figure 2).

```
data("mfD_healthy", package = "roahd")
```

The dataset `mfD healthy` collects preprocessed (denoised, smoothed and registered) 8-leads electrocardiographic (ECG) signals during a median heartbeat of a sample of 50 healthy subjects. An ECG signal records in a of voltage versus time the electrical activity of the heart using electrodes placed on the skin. These electrodes detect the small electrical changes that are a consequence of cardiac muscle depolarization followed by repolarization during each cardiac cycle (heartbeat). In a conventional 12-lead ECG, ten electrodes are placed on the patient's limbs and on the surface of the chest. The overall magnitude of the heart's electrical potential is then measured from twelve different angles ('leads'). Of these 12 leads, the first six are derived from the same three measurement points. Therefore, any two of these six leads include exactly the same information as the other four. So, the ECG traces are composed of 8 leads, called I, II, V1, V2, V3, V4, V5 and V6. Analogously the dataset `mfD LBBB` contains the ECGs of 50 patients affected by Left Bundle Branch Block, LBBB in the following. These data arise from the project PROMETEO (PROgetto sull'area Milanese Elettrocardiogrammi Teletrasferiti dall'Extra Ospedaliero) started with the aim of spreading the intensive use of ECG as pre-hospital diagnostic tool and of constructing a new database of ECGs with features never recorded before in any other data collection on heart diseases. Measures of electric potential are given in μV and the final sample grid after preprocessing consists of 1024 points at 1KHz. For more details on the original dataset, and on the preprocessing steps, see Ieva et al. (2013).

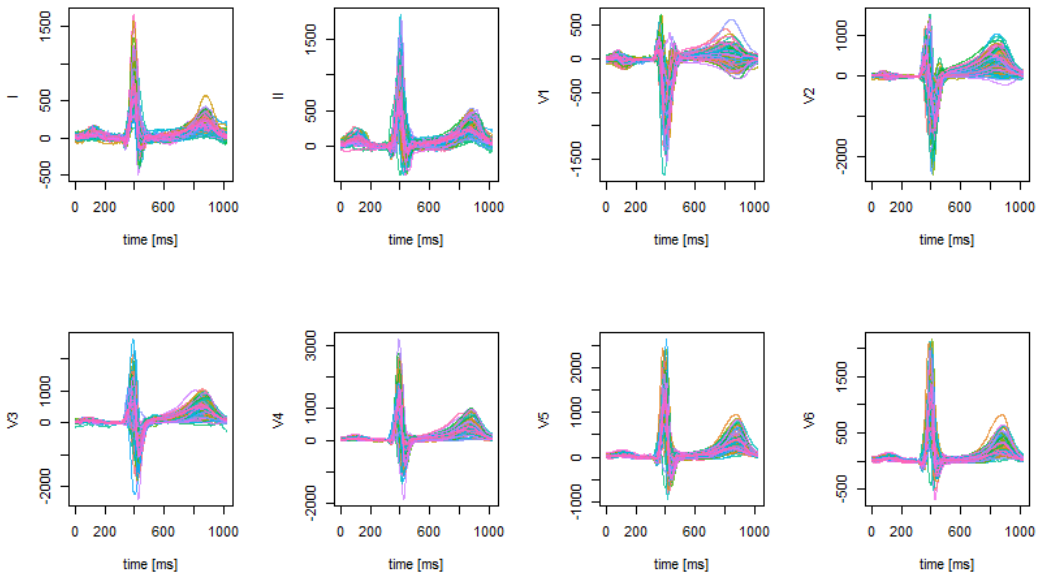


Figure 2: A visualization of the `mfD_healthy` dataset in the **roahd** package.

Simulation of functional data

The **roahd** package contains functions that can be used to simulate artificial data sets of functional data, both univariate and multivariate. The data are obtained as realisations of a Gaussian process over a discrete grid with a specific variance-covariance operator and mean, see Rasmussen and Williams (2006). In general, given a covariance function, $C(s,t) := \mathrm{Cov}(X(s), X(t))$, $s,t \in I$, and a mean function $\mu(t) := \mathbb{E}[X(t)]$, the model generating a data sample of size N is:

$$X_i(t) = \mu(t) + \varepsilon(t), \quad \mathrm{Cov}(\varepsilon(s), \varepsilon(t)) = C(s,t), \quad i = 1, \ldots, N, \qquad t \in I.$$

The finite-dimensional approximation on a discrete grid $I = [t_0, \ldots, t_{P-1}]$, with $t_{i+1} - t_i = h$, $\forall i = 0, \ldots, P-1$, instead, is generated according to:

$$X_{i,j} = X_i(t_j) = \mu(t_j) + \varepsilon(t_j), \quad \mathrm{Cov}(\varepsilon(t_j), \varepsilon(t_k)) = C_{j,k}, \quad i = 1, \ldots, N, \qquad t_j, t_k \in I.$$

The function `generate_gauss_fdata` requires the sample size `N`, the mean vector $[\mu(t_j)]$ as parameter `centerline`, the matrix representation of the desired variance-covariance operator $[C_{j,k}]$ as parameter `Cov` or, as alternative to `Cov`, its Cholesky factor `CholCov`, which is what is actually used to impose the desired covariance structure of the errors of the generating formulas. A built-in function can be used to generate exponential Matérn covariance functions, namely `exp_cov_function(grid,`

`alpha, beta`), returning the discretised version of a covariance of the form $C(s,t) = \alpha e^{-\beta|s-t|}$. Here, the parameter α controls the overall level of variability in the signal, while the parameter β affects the autocorrelation length of the signal's noise, with lower values of β leading to wider correlation lengths and vice-versa. This parameter is particularly useful to produce random distortions in the shape of the synthetic functional data around the desired centerline. An example of the function is the following:

```
generate_gauss_fdata( N=50,
centerline = sin( 2 * pi * seq( 0, 1, length.out = 10^3 ) ),
Cov =exp_cov_function( seq( 0, 1, length.out = 10^3 ),
alpha = 0.2, beta  = 0.3 ) )
```

The simulated data are show in Figure 3.

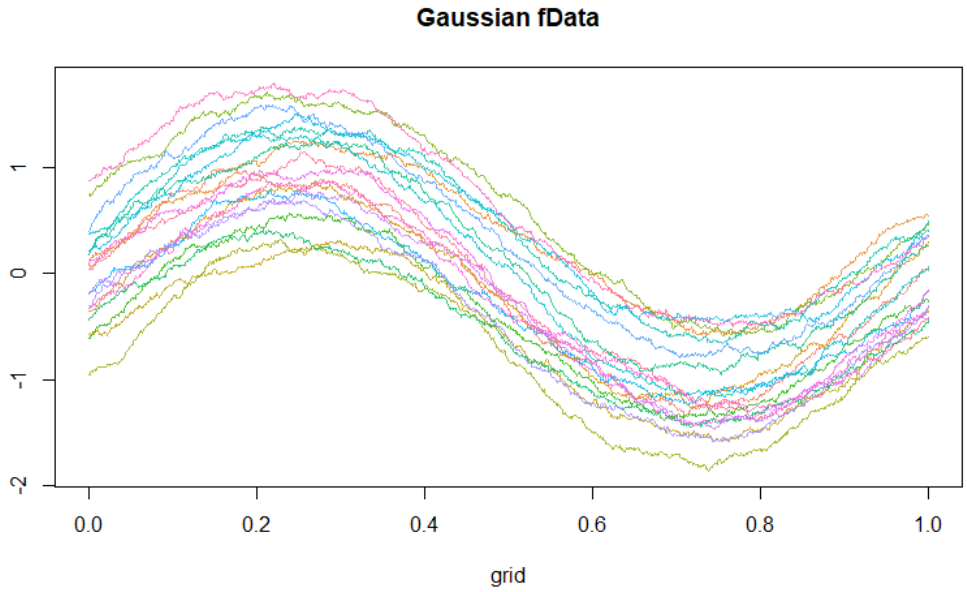


Figure 3: An example of simulated univariate functional data with 50 curves evaluated on a evenly spaced grid of 1000 points.

Similarly, we can use the function `generate_gauss_mfdata` to generate a sample of L-variate functional data according to the following model:

$$X_{i,k} = \mu_k(t) + \varepsilon_k(t), \quad \text{Cov}(\varepsilon_k(s), \varepsilon_k(t)) = C(s,t), \quad \forall i = 1,\ldots,N, \quad \forall k = 1,\ldots,L,$$

where $Cor(\varepsilon_j(t), \varepsilon_l(t)) = \rho_{j,l} \in [-1,1]$ specifies the synchronous, constant correlation structure among the components of the functional dataset.

The function requires the sample size `N`; the number of components of the multivariate data `L`; a matrix containing by rows the means of each component `centerline`; a vector `correlations` of length $1/2 \cdot L \cdot (L-1)$ containing all the correlation coefficients $\rho_{j,l}$ among the components; and either a list `listCov` containing the discretised covariance functions over the grid $I \times I$, or a list `listCholCov` of their Cholesky factors.

An example of the function is the following

```
generate_gauss_mfdata( N=100, 2,
centerline = matrix( c( sin( 2 * pi * seq( 0, 1, length.out = 10^3 )),
cos( 2 * pi * seq( 0, 1, length.out = 10^3 ) ) ), nrow = 2, byrow = TRUE ),
correlations = 0.5,
listCov = list(exp_cov_function( seq( 0, 1, length.out = 10^3 ),
alpha = 0.1, beta  = 0.5 ),
exp_cov_function( seq( 0, 1, length.out = 10^3 ),
alpha = 0.5, beta  = 0.1 )))
```

The simulated data are show in Figure 4.

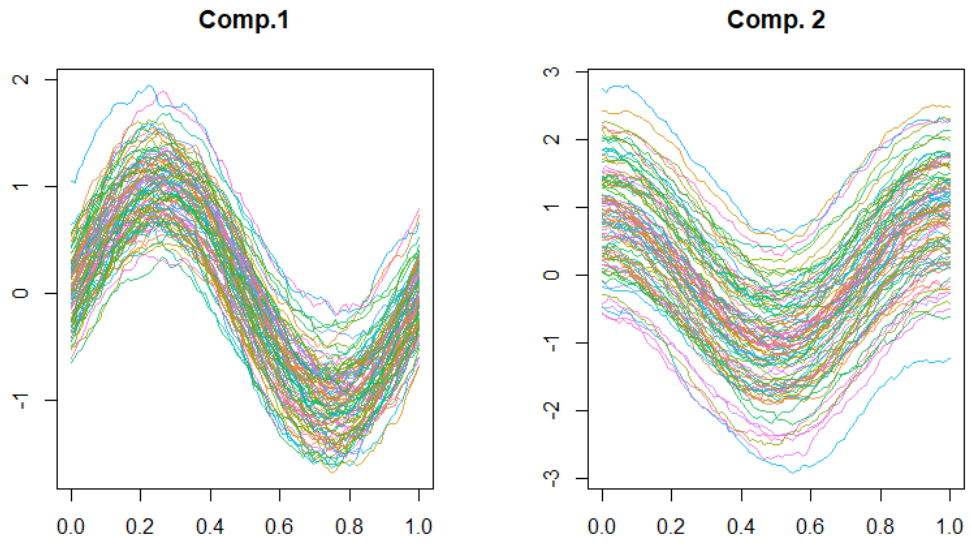


Figure 4: An example of simulated bivariate functional data with 100 bivariates curves evaluated on a evenly spaced grid of 1000 points.

Robust Statistics

In order to provide a center-outward and a down-upward/up-downward order of data, the Band Depth (BD) and Modified Band Depth (MBD) are implemented both for functional and multivariate functional data. Let us recall the empirical version of Band Depth for functional data, as introduced in López-Pintado and Romo (2009) and in López-Pintado and Romo (2011). Given a stochastic process X taking values on the space $\mathcal{C}(I)$ of real continuous functions on the compact interval I, the empirical version of the band depth of order $J \geq 2$ for a function $f \in \mathcal{C}(I)$ is

$$\mathrm{BD}_X^J(f) = \sum_{j=1}^{J} \binom{N}{j}^{-1} \sum_{i_1<i_2<\ldots,<i_j} \mathbb{I}\left\{G(f) \subset B(f_{i_1},\ldots,f_{i_j})\right\}, \tag{1}$$

where the subset of the plane $G(f) = \{(t, f(t)) : t \in I\}$ is the graph of the function f. $B(f_1, f_2, \ldots, f_j)$ is the band in $\mathbb{R}^2$ delimited by $f_1, f_2, \ldots, f_j$, realizations of independent copies of the stochastic process X (i.e. the functional dataset), and it is defined as:

$$B(f_1, f_2, \ldots, f_j) = \{(t, y(t)) : t \in I, \min_{r=1,\ldots,j} f_r(t) \leq y(t) \leq \max_{r=1,\ldots,j} f_r(t)\}.$$

To overcome the problem of heavy ties due to the presence of the indicator function, López-Pintado and Romo (2009) proposed the Modified Band Depth, where the time interval that f spends in the random band is weighted over I. The empirical version of the MBD is:

$$\mathrm{MBD}_X^J(f) = \sum_{j=2}^{J} \binom{N}{j}^{-1} \sum_{1\leq i_1<i_2<\cdots<i_j\leq n} \tilde{\lambda}\{E(f; f_{i_1},\ldots,f_{i_j})\}, \tag{2}$$

where $E(f) := E(f; f_{i_1},\ldots,f_{i_j})$=$\{t \in I, \min_{r=i_1,\ldots,i_j} f_r(t) \leq f(t) \leq \max_{r=i_1,\ldots,i_j} f_r(t)\}$ and $\tilde{\lambda}(g) = \lambda(E(g))/\lambda(I)$ with λ the Lebesgue measure on I. In López-Pintado and Romo (2009), the authors state that while the choice of J clearly increases the magnitude of depth, it does not affect the induced ordering and therefore the ranks. This was supported by a simulation study in Tarabelloni et al. (2015). To reduce the computational effort we set $J = 2$. Given a set of curves $(f, f_1, \ldots f_N)$, the MBD of f, that we will denote by $MBD_{\{f_1,\ldots,f_N\}}(f)$, measures the proportion of time interval I where the graph of f belongs to the envelopes of all the possible couples (f_{i_1}, f_{i_2}), $i_1 < i_2 = 1, \ldots, N$.

Both the functions `BD` and `MBD` require either an object of class `fData` or a matrix-like dataset of functional data (e.g., `fData$values`), with observations as rows and measurements over grid points as columns. They return a vector containing the values of depth for the given dataset. Thanks to these values, the dataset $f_1, \ldots, f_N$ can be center-outward ranked. We will denote $f_{[i]}$ the sample curve associated with

the $i-$th largest depth value, so $f_{[1]} = \text{argmax}_{f\in\{f_1,\ldots,f_N\}} MBD(f)$ is the *median* (deepest and most central) curve, and $f_{[N]} = \text{argmin}_{f\in\{f_1,\ldots,f_N\}} MBD(f)$ the most outlying one. Moreover, for $\alpha \in (0,1)$ we can construct the α% central region determined by a sample of curves as:

$$\mathcal{C}_\alpha = \left\{ (t, y(t)) : \min_{r=1,\ldots,\lceil \alpha n \rceil} f_{[r]}(t) \leq y(t) \leq \max_{r=1,\ldots,\lceil \alpha n \rceil} f_{[r]}(t) \right\}. \tag{3}$$

As an example, let us compute the depth measures and the median of the simulated dataset shown in Figure 3:

```
MBD(fD)
[1] 0.48503510 0.46228408 0.51505469 0.31594122 0.37595102
[6] 0.49060245 0.35306449 0.40934204 0.47676898 0.37799510
[11] 0.22585633 0.45465633 0.49204245 0.34556571 0.14763429
[16] 0.27375020 0.42478041 0.51758041 0.07788571 0.49579265
...
```

Another interesting down-upward/up-downward order of data can be built on top of Epigraph Index (EI) and Hypograph Index (HI) or of their corresponding Modified versions (MEI and MHI). We recall the definition of EI (HI) for univariate functional data as introduced in López-Pintado and Romo (2011). Given a stochastic process X taking values on the space $\mathcal{C}(I)$ the empirical version of the Epigraph (Hypograph) Index of a function $f \in \mathcal{C}(I)$ are:

$$EI_X(f) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left\{ f_i(t) \geq f(t), \quad \forall t \in I \right\}, \tag{4}$$

$$HI_X(f) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left\{ f_i(t) \leq f(t), \quad \forall t \in I \right\}, \tag{5}$$

where $f_1, f_2, ..., f_N$ are realizations of independent copies of the stochastic process X (i.e. the functional dataset). As before, to overcome the problem of heavy ties it is more suitable to use the modified version of these indexes, whose empirical version are:

$$MEI_X(f) = \frac{1}{N}\sum_{i=1}^{N} \tilde{\lambda}(\{t \in I, f_i(t) \geq f(t)\}). \tag{6}$$

$$MHI_X(f) = \frac{1}{N}\sum_{i=1}^{N} \tilde{\lambda}(\{t \in I, f_i(t) \leq f(t)\}). \tag{7}$$

Therefore, given a set of curves $(f, f_1, ...f_N)$ the MEI (MHI) of f, denoted by $MEI_{\{f_1,\ldots,f_N\}}(f)$ ($MHI_{\{f_1,\ldots,f_N\}}(f)$) accounts for the mean proportion of time interval I where f lies below (above) the curves of the sample. Like depths, all the functions `EI` (`HI`) and `MEI` (`MHI`) require either an object of class `fData` or a matrix-like dataset of functional data (e.g., `fData$values`), with observations as rows and measurements over grid points as columns. They return a vector containing the values of the corresponding indexes for the given dataset, that can provide the desired ordering of data. In López-Pintado and Romo (2011) the authors propose another well-posed definition of depth, the Modified Half Region Depth (MHRD) for functional data as:

$$MHRD_{\{f_1,\ldots,f_N\}}(f) = \min(MEI_{\{f_1,\ldots,f_N\}}(f), MHI_{\{f_1,\ldots,f_N\}}(f)). \tag{8}$$

The `MHRD` funtion in **roahd** computes the MHRD of elements of a univariate functional dataset, and has the same usage as the previous functions.

In Ieva et al. (2013) and Ieva and Paganoni (2017), these statistics have been generalized to multivariate functional framework. Let $\mathbf{X}$ be a stochastic process taking values in the space $C(I;\mathbb{R}^L)$ of continuous functions $\mathbf{f} = (f_1, ..., f_L) : I \to \mathbb{R}^L$, with I as before. We have a dataset F_N constituted of $N \in \mathbb{N}$ sample observations of this process, which we indicate by $\mathbf{f}_1, \ldots, \mathbf{f}_N$, where $\mathbf{f}_j = (f_{j1}, ..., f_{jL})$. The MBD of $\mathbf{f}$ with respect to F_n becomes

$$MBD_{\{\mathbf{f_1},\ldots,\mathbf{f_n}\}}(\mathbf{f}) = \sum_{k=1}^{L} p_k MBD^J_{\{f_{1k},\ldots,f_{Nk}\}}(f_k), \tag{9}$$

with $p_k > 0$, $\forall\, k = 1, ..., L$, $\quad \sum_{k=1}^{L} p_k = 1$.

Analogously we define the MEI (MHI) of $\mathbf{f}$ with respect to F_N as

$$MEI_{\{\mathbf{f_1},...,\mathbf{f_N}\}}(\mathbf{f}) = \sum_{k=1}^{L} p_k MEI_{\{f_{1k},...,f_{Nk}\}}(f_k), \tag{10}$$

$$MHI_{\{\mathbf{f_1},...,\mathbf{f_N}\}}(\mathbf{f}) = \sum_{k=1}^{L} p_k MHI_{\{f_{1k},...,f_{Nk}\}}(f_k), \tag{11}$$

with $p_k > 0, k = 1, ..., L, \quad \sum_{k=1}^{L} p_k = 1$. In (9), (10) and (11) the curves that form the envelops are the components of the curves in F_N. In **roahd**, the function `multiMBD` computes the MBD for a dataset of multivariate curves. In particular, `multiMBD` requires either an object of class `mfData` or a list of 2-dimensional matrices (`Data`) having as rows the units of that component and as columns the measurements of the functional data over the grid, as well as either a set of weights `weights` or the string *uniform* specifying that a set of uniform weights (of value $1/L$, where L is the number of components of the functional dataset) must be employed. The function returns a vector containing the depth of each element of the multivariate functional dataset. As an example let us compute the depth measures of the simulated dataset (named `mfD`) shown in Figure 4, using uniform weights:

```
multiMBD(mfD, weights="uniform")
[1] 0.40842020 0.45438788 0.24038384 0.31500606 0.35914343
[6] 0.48603636 0.44544040 0.42960606 0.37119798 0.21466667
[11] 0.46331111 0.37947677 0.46344646 0.39914141 0.30079394
[16] 0.48643838 0.14745657 0.47115354 0.41804242 0.32814545
...
```

The choice for the weights p_k's averaging the contribution of each component of the multivariate functional data is usually problem-driven, and in general no gold standards are available. If there is no a priori knowledge about the dependence structure between components, they could be chosen uniformly. In Tarabelloni et al. (2015) a different choice has been proposed, taking into account the distance between the estimated variance-covariance operators of the two groups identified by the binary outcome which was the focus of the study. In Ieva and Paganoni (2017) the weights p_k's are chosen taking into account the variability of each component of the multivariate functional process that generates data, so the weight of each component is proportional to the inverse of spectral norm of its variance-covariance operator:

$$q_k = 1/\lambda_k^{(1)} \quad \text{and} \quad p_k = \frac{q_k}{\sum q_k}, \tag{12}$$

where $\lambda_k^{(1)}$ is the maximum eigenvalue of the variance-covariance operator of the $k-$th component.

The functions `median_fData` (`median_mfData`) of the package compute the sample median of a univariate (multivariate) functional dataset based on a definition of depth for univariate (multivariate) functional data. Their input is the dataset whose median is required, in form of `fData` or `mfData` object, and a string specifying the name of the depth definition to use, as parameter `type`. This name should bind to a function actually defined in the workspace, such as the build-in ones of **roahd** (e.g., MBD, MHRD, etc.).

Figure 5 shows the plot of healthy ECG data (see `mfD_healthy`) with superimposed the multivariate functional median computed maximizing the multivariate MBD (9) with uniform weights.

```
median_mfData(mfD_healthy, type = "multiMBD")
```

Correlation coefficient

When dealing with multivariate functional data, it is possible to compute correlation coefficients between observations' univariate components that generalise the Spearman's coefficient ρ_s, see Valencia et al. (2015b,a). In particular, let us consider a set of bivariate ($L = 2$) functional data $\{\mathbf{f_1}, ..., \mathbf{f_N}\}$. The Spearman coefficient related to the data set is defined by

$$\hat{\rho}_s = \hat{\rho}_p(MEI_{\{f_{11},...,f_{N1}\}}(\mathbf{f_1}), MEI_{\{f_{12},...,f_{N2}\}}(\mathbf{f_2})), \tag{13}$$

where $\hat{\rho}_p$ is the usual Pearson correlation coefficient between vectors and

$$MEI_{\{f_{11},...,f_{N1}\}}(\mathbf{f_1}) = (MEI_{\{f_{11},...,f_{N1}\}}(f_{11}), ..., MEI_{\{f_{11},...,f_{N1}\}}(f_{N1})),$$
$$MEI_{\{f_{12},...,f_{N2}\}}(\mathbf{f_2}) = (MEI_{\{f_{12},...,f_{N2}\}}(f_{12}), ..., MEI_{\{f_{12},...,f_{N2}\}}(f_{N2})).$$

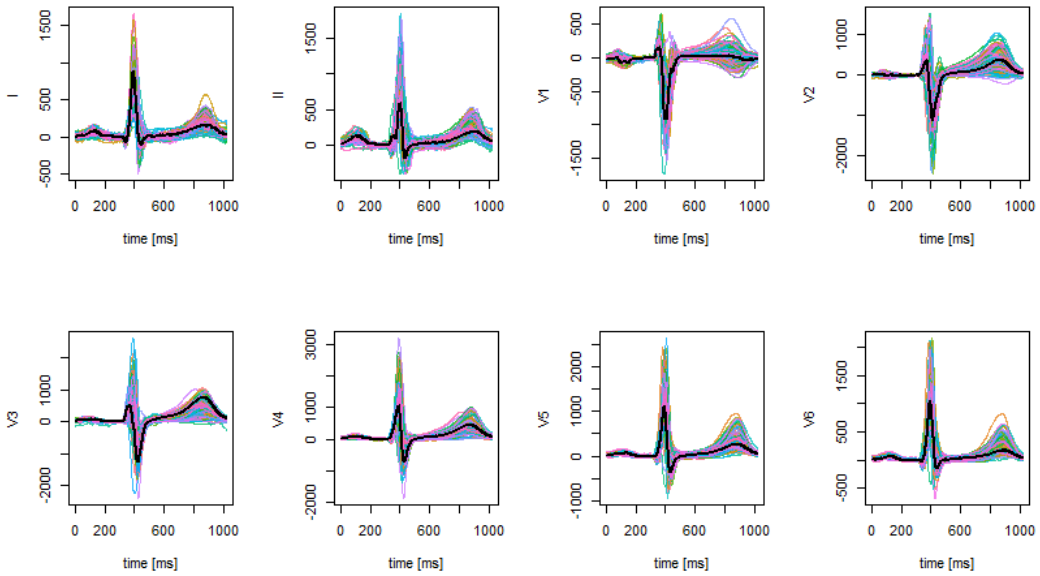


Figure 5: Plot of healthy data with superimposed the multivariate functional median computed maximizing the multivariate MBD (9) with uniform weights.

Another definition can be obtained by replacing MEI by MHI. The properties of $\hat{\rho}_s$ are detailed in Valencia et al. (2015b), where also its consistency is proved.

The function `cor_spearman` can be used to compute the Spearman correlation coefficient (13) for a bivariate `mfData` object, using the ordering definition specified by `ordering` (the default is to use `MEI`) to rank univariate components and then compute the correlation coefficient. Besides `MEI`, also `MHI` can be used to determine ranks.

Another well known measure for concordance in bivariate data is the Kendall's τ index. In Valencia et al. (2015a) the authors propose two generalizations of τ based on two different preorders between curves $\preceq$. Consider two functions g and h in $\mathcal{C}(I)$. Two possible preorders are:

$$g(t) \preceq_m h(t) \equiv \max_{t\in I} g(t) \leq \max_{t\in I} h(t), \tag{14}$$

$$g(t) \preceq_i h(t) \equiv \int_I (h(t) - g(t))dt \geq 0. \tag{15}$$

Let us consider a set of bivariate ($L = 2$) functional data $\{\mathbf{f_1}, ..., \mathbf{f_N}\}$, then the functional $\hat{\tau}$ is

$$\hat{\tau} = \binom{N}{2} \sum_{i<j}^{N} 2\mathbb{I}(f_{i1} \preceq f_{j1}, f_{i2} \preceq f_{j2}) + 2\mathbb{I}(f_{j1} \preceq f_{i1}, f_{j2} \preceq f_{i2}) - 1. \tag{16}$$

In fact it is possible to prove that $\hat{\tau}$ in (16) measures the difference between the number of concordant pairs of curves and the number of discordant pairs of curves, using a possible preorder. The function `cor_kendall` can be used to compute the Kendall correlation coefficient (16) for a bivariate `mfData` object, using the ordering definition specified by `ordering` (by default `max` is used, i.e., formula (14)) to rank univariate components, then to compute concordant and discordant pairs and then the correlation coefficient. Also `area` (i.e., formula (15)) can be used. As an example let us compute $\hat{\rho}$ and $\hat{\tau}$ on the simulated dataset (named `mfD`) shown in Figure 4:

```
cor_spearman(mfD, ordering ="MEI")
[1] 0.6098597

cor_kendall(mfD, ordering="area")
[1] 0.4222222
```

Inference on Spearman correlation

In Ieva et al. (2018) a boostrap-based inferential framework for the Spearman coefficient is introduced. In particular the authors suggest to compute a sample from the bootstrap distribution of the statistic $\hat{\rho}$, i.e. $(\hat{\rho}_1^*, ...\hat{\rho}_B^*)$, where $\hat{\rho}_j^*$ denotes the value of the statistics computed on a random sample of size N drawn with replacement from the population of N equally likely data, named boostrap sample, and B is the number of boostrap samples considered. Using the empirical quantiles of the boostrap distribution of the statistics, it's possible to compute a confidence interval of a fixed level (say $1-\alpha$, $\alpha \in (0,1)$):

$$(\hat{\theta_l}; \hat{\theta_u}) = (\hat{\rho}^*_{\alpha/2}; \hat{\rho}^*_{1-\alpha/2}); \tag{17}$$

where $\hat{\rho}^*_\alpha$ denotes the α% percentile of the sample boostrap distribution $(\hat{\rho}_1^*, ...\hat{\rho}_B^*)$. To mitigate the bad coverage performances of (17), as detailed in Efron and Tibshirani (1993), it's better to consider an improved version of the percentile method called Bias-Corrected and Accelerated (BCA) interval. This improved version of the confidence interval is implemented by the function `BCIntervalSpearman`. This function requires: two univariate functional datasets in form of `fData` objects, `fD1`, `fD2`; the ordering relation to be used in the Spearman's coefficient computation as the parameter `ordering`; the number of bootstrap iterations to use in order to estimate the confidence interval, `bootstrap_iterations` and the coverage probability (1-α).
As an example let us compute a BCA interval of confidence 0.95 for the Spearman correlation coefficient of the simulated dataset (named `mfD`) shown in Figure 4:

```
BCIntervalSpearman(mfD$fDList[[1]], mfD$fDList[[2]], ordering = 'MEI',
alpha=0.05, bootstrap_iterations = 1000)
$lower
[1] 0.6520883

$upper
[1] 0.9819355
```

A verbosity parameter can be set in function `BCIntervalSpearman` in order to log information on the function's progress when the computational time is long. The simple or Bias-Corrected and Accelerated version of the confidence interval allows for testing the presence of dependency among two families od univariate curves, i.e. $H_0 : \rho_s = 0$ vs $H_1 : \rho_s \neq 0$. Consider now the case of a multivariate functional dataset, where the observations are realizations of the stochastic process $\mathbf{X}$ taking values in the space $C(I; \mathbb{R}^L)$, $L > 2$. In this case, the pattern of dependence between the components can be expressed in a symmetrix matrix S, named Spearman matrix. Its entry $S_{i,j}$ is the Spearman coefficient between the data of the i-th and j-th components. Analogously, for each of the $L(L-1)/2$ coefficients the corresponding BCA confidence interval can be computed. The function `cor_spearman` applied to a dataset of type `mfData` returns the pointwise estimate of the Spearman matrix S, while the function `BCIntervalSpearmanMultivariate` returns two matrices containing the lower and upper bounds of the corresponding confidence intervals. To clarify their use, we show the results corresponding to the first two leads, i.e. I and II of `mfD_healthy`.

```
mfD_healthy_subset = as.mfData(list(mfD_healthy$fDList[[1]],
mfD_healthy$fDList[[2]]))

cor_spearman(mfD_healthy_subset, ordering='MEI')
[1] 0.6840466

BCIntervalSpearmanMultivariate(mfD_healthy_subset,
ordering='MEI', alpha=0.05, bootstrap_iterations = 1000)
$lower
          [,1]      [,2]
[1,] 1.0000000 0.4805781
[2,] 0.4805781 1.0000000

$upper
         [,1]     [,2]
[1,] 1.000000 0.820072
[2,] 0.820072 1.000000
```

In order to perform a comparison between correlation patterns across different populations of multivariate functional data, in Ieva et al. (2018) the authors propose a bootstrap procedure to test the

equality between the two corresponding Spearman matrices. Consider two multivariate functional datasets, where the observations are realizations of the stochastic processes $\mathbf{X}$ and $\mathbf{Y}$, respectively, both taking values in the space $C(I; \mathbb{R}^L)$. We want to perform a bootstrap test to check the hypothesis:

$$H_0 : S_X = S_Y \qquad \text{vs} \qquad H_1 : S_X \neq S_Y, \tag{18}$$

where S_X and S_Y denote the $L \times L$ matrices of Spearman correlation coefficients of the two populations. The proposed bootstrap test statistics is based on a norm of the differences between the two matrices. In particular, the authors consider three different norms in the space of the $L \times L$ matrices:

(i) One norm: $\|W\|_1 = \max_{j=1,\cdots,L} \sum_{i=1}^{L} |w_{ij}|$;

(ii) Infinity norm: $\|W\|_\infty = \max_{i=1,\cdots,L} \sum_{j=1}^{L} |w_{ij}|$;

(iii) Frobenius norm: $\|W\|_F = \sqrt{\sum_{i=1}^{L} \sum_{j=1}^{L} |w_{ij}|^2}$.

The function `BTestSpearman` performs the test described above and requires: two univariate functional samples in form of `mfData` object, `mfD1`, `mfD2`; the ordering relation to be used in the Spearman's coefficient computation `ordering`; the number of bootstrap iterations to be performed `bootstrap_iterations`; the norm to measure the differences between the Spearman correlation matrices of the two functional datasets, `normtype` (the allowed values are the same as for parameter `type` in **R**'s base function `norm`). The function returns the estimates of the test's p-value and statistics. As an example let us perform the test considering the first two components of `mfd_healthy` and `mfD_LBB` datasets provided by the **roahd** package.

```
mfD_healthy_subset = as.mfData(list(mfD_healthy$fDList[[1]],
mfD_healthy$fDList[[2]]))

mfD_LBBB_subset = as.mfData(list(mfD_LBBB$fDList[[1]],
mfD_LBBB$fDList[[2]]))

BTestSpearman(mfD_healthy_subset, mfD_LBBB_subset,
bootstrap_iterations = 1000,
ordering = "MEI", normtype = "f")

$pvalue
[1] 0.473

$phi
[1] 0.06562356
```

Graphical tools

The tools shown in this section (i.e., the functional boxplot and the outliergram) enable a complete inferential analysis of (multivariate) functional data based on robust statistics, like depth measures, described in Section Robust Statistics. These tools are very useful also in the outlier detection framework which is of primary interest in FDA, since outliers may deeply affect the inference of high dimensional data, especially whenever the sample size is small.
The functional boxplot (see Sun and Genton (2011)) is obtained by ranking functions from the center of the distribution outwards thanks to a suitable depth definition, computing the region of 50% most central functions, see Eq. (3). The fences are obtained by inflating such region by a factor F. Given the envelope of the functions entirely contained inside the inflated region, the data crossing these fences even for one time instant are considered outliers. Once the outlying observations have been identified, they can be isolated from the original dataset and either carefully examined or discarded as corrupted by undesired factors.

The function `fbplot` computes the depths of a dataset and marks outlying observations. If used with graphical option on (default behaviour), it also plots the functional boxplot of the dataset. `fbplot` requires: the univariate functional dataset whose functional boxplot must be determined in form of an `fData` object `Data`; either a vector containing the depths for each statistical unit of the dataset, or a string containing the name of the method you want to use to compute; the value of the inflation factor, `Fvalue` (the default value is 1.5). In Figure 6 we show the functional boxplot of the first lead, i.e. I of `mfD_healthy`.

```
fbplot(mfD_healthy$fDList[[1]], Depths="MBD", Fvalue=3,
main="Functional Boxplot")
```

```
$Depth
[1] 0.4399681 0.1534263 0.4097385 0.4116510 0.3872242
[6] 0.4123326 0.2387404 0.3568670 0.3669691 0.4601483
[11] 0.3006872 0.3744531 0.1360906 0.4319324 0.3206186
[16] 0.2400199 0.4340800 0.4462739 0.2805389 0.4638281
...

$Fvalue
[1] 3

$ID_outliers
2
```

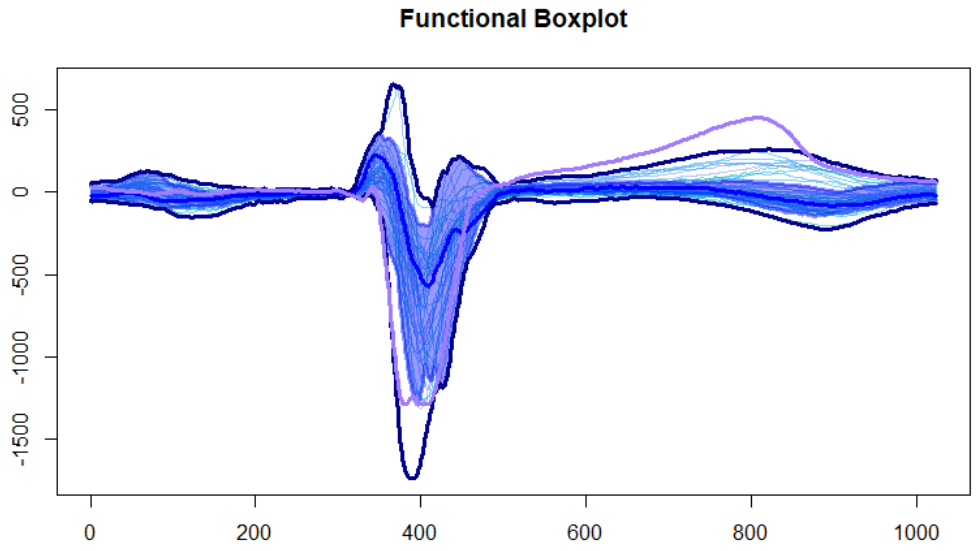


Figure 6: Functional Boxplot of the first lead, i.e., I of healthy data

The function `fbplot` also allows to automatically compute the best adjustment factor `F` that yields a desired proportion of outliers (True Positive Rate, `TPR`) of a Gaussian dataset with same center and covariance function as the `fData` object (see Sun and Genton (2012)). Such automatic tuning involves the simulation of a number `N_trials` of separate datasets of Gaussian functional data with same center and covariance as the original dataset (the covariance is robustly estimated with the function `covOGK` of the package **robustbase**, see Maronna and Zamar (2002)) of size `trial_size`, and the computation of `N_trials` values for `Fvalue` such that the desired proportion `TPR` of observations is flagged as outliers. The optimal value of `Fvalue` for the original population is then found as the average of the previously computed values `Fvalue`. The parameters to control the adjustment procedure can be passed through the argument `adjust`, whose default is `FALSE` and otherwise is a list with (some of) the fields:

- `N_trials`: the number of repetitions of the adjustment procedure based on the simulation of a Gaussisan dataset of functional data, each one producing an adjusted value of F, which will lead to the averaged adjusted value `Fvalue`. Default is 20;
- `trial_size`: the number of statistical units in the Gaussian population of functional data that will be simulated at each repetition of the adjustment procedure. Default is $8 \times N$;
- `TPR`: the True Positive Rate of outliers, i.e., the proportion of observations in a dataset without amplitude outliers that have to be considered outliers. Default is $2\phi(4z_{0.25})$ where ϕ and z_α denote, respectively, the cumulative distribution function and the quantile of order α of a standard Gaussian distribution.
- `F_min`: the minimum value of `Fvalue`, defining the left boundary for the optimisation problem aimed at finding the optimal value of `Fvalue`;
- `F_max`: the maximum value of `Fvalue`, defining the right boundary for the optimisation problem aimed at finding the optimal value of `Fvalue`;

- `tol`: the tolerance to be used in the optimisation problem aimed at finding the optimal value of `Fvalue`;
- `maxiter`: the maximum number of iterations to solve the optimisation problem aimed at finding the optimal value of `Fvalue`.

Due to the **S3** specialization, `fbplot` can construct also functional boxplots of a multivariate functional dataset (see Ieva and Paganoni (2013)). In Figure 7 we show the functional boxplot of the first two leads, i.e. I and II of `mfD_healthy`.

```
mfD_healthy_subset <- as.mfData(list(mfD_healthy$fDList[[1]],
mfD_healthy$fDList[[2]]))

fbplot( mfD_healthy_subset, Fvalue = 1.5, xlab = 'time',
ylab = list( 'Values 1', 'Values 2' ),
main = list( 'First component', 'Second component' ) )

$Depth
[1] 0.4061113 0.1695619 0.4086113 0.4204500 0.3005780
[6] 0.4233634 0.3170089 0.3522569 0.3821995 0.4625769
[11] 0.3055684 0.3535029 0.1630668 0.4213114 0.3553795
[16] 0.3079317 0.3853858 0.3711121 0.2640493 0.4489892
...

$Fvalue
[1] 1.5

$ID_outliers
[1]  2 16

$Depth
[1] 0.43298990 0.36373333 0.40771515 0.38667677 0.10381616
[6] 0.44186869 0.36482424 0.34480404 0.45246061 0.36904040
[11] 0.25263232 0.28706869 0.42344040 0.29333333 0.02921414
[16] 0.49872121 0.49638384 0.41457980 0.25275960 0.29689697
 ....

$Fvalue
[1] 2.5

$ID_outliers
[1]  5 15 27 31 32 33 34 35 44 50 52 55 57 59 63 67 71 73 75 85 23
```

Let us point out that the functional boxplot has been constructed mainly for detection of magnitude outliers, i.e., curves that lie far from the range of the majority bulk of data.

Outliergram

A method that can be used to detect shape outliers and covariance outliers is the outliergram (see Arribas-Gil and Romo (2014)), based on the computation of MBD and MEI (see (2) and (6)) of univariate functional data. Shape outliers are curves that present a different pattern with respect to the rest of the data in terms of their derivatives and covariance outliers are curves generated by a model that is different from the model of the majority of data just in terms of the variance and covariance operator that affects the second order moments of data. Given a set of data $f_1, \ldots, f_n$ in the space $\mathcal{C}(I; \mathbb{R})$ of the continuous functions the following inequality holds:

$$MBD_{\{f_1,\ldots,f_n\}}(f_j) \leq a_0 + a_1 MEI_{\{f_1,\ldots,f_n\}}(f_j) + a_2 n^2 (MEI_{\{f_1,\ldots,f_n\}}(f_j))^2, \quad j = 1, \cdots, n, \tag{19}$$

where $a_0 = a_2 = -2/(n(n-1))$ and $a_1 = 2(n+1)/(n-1)$. So considering a scatterplot of MBD against multivariate MEI of data, the points lying far from the quadratic boundary (19) correspond to shape outliers, and data with very low values of MBD are potential magnitude outliers (see Arribas-Gil and Romo (2014)). The function `outliergram` displays the outliergram of a univariate functional dataset of class `fData` (see Figure 8) and returns a vector of observation IDs indicating the outlying observations in the dataset.

The function `multivariate_outliergram` implements the generalisation of the outliergram to multivariate functional data, following Ieva and Paganoni (2017). Let $\mathbf{X}$ be a stochastic process taking

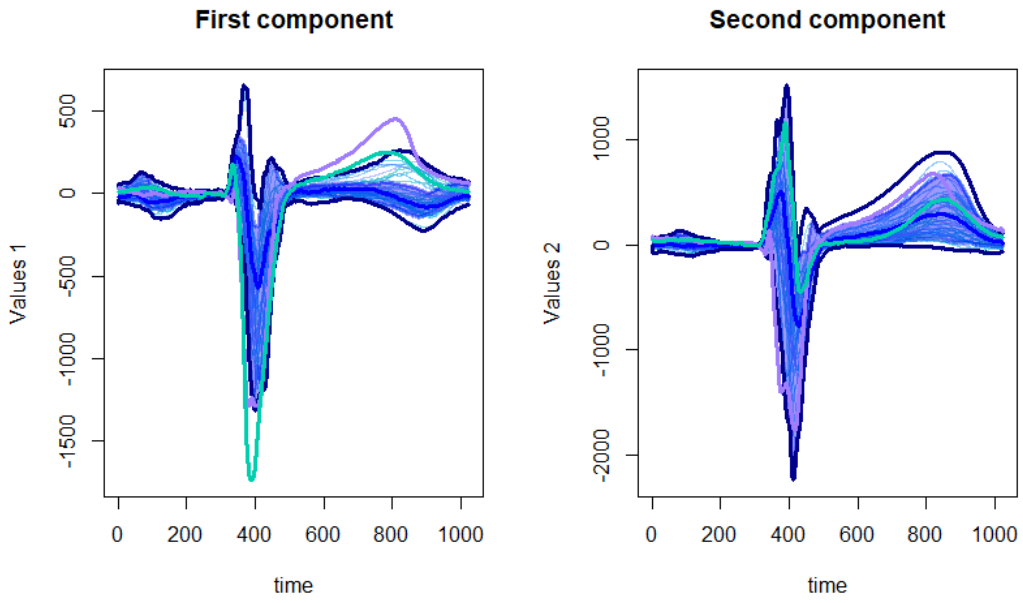


Figure 7: Functional Boxplot of the simulated data presented in Figure 4

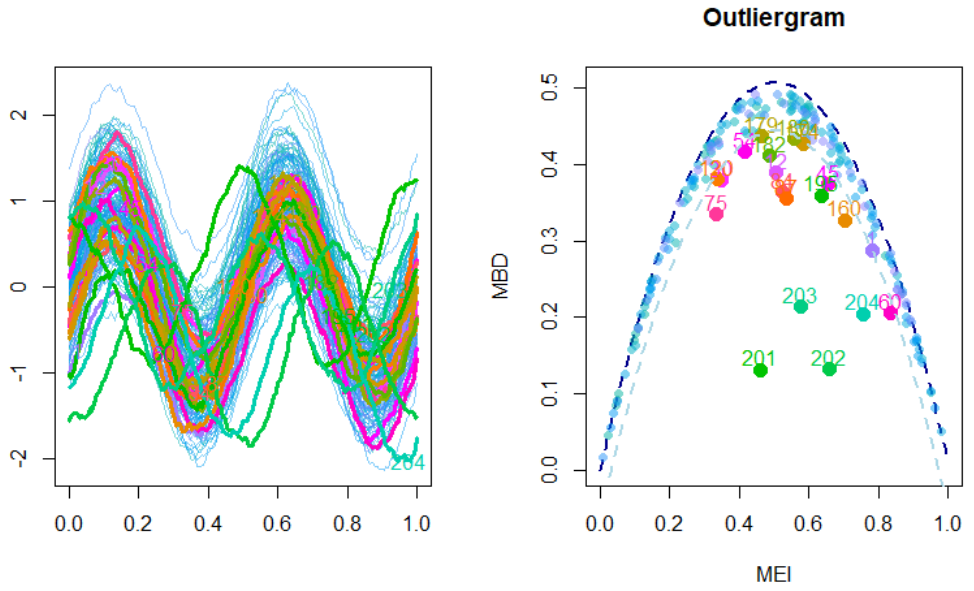


Figure 8: An example of outliergram on simulated univariate functional data

values in the space $C(I;\mathbb{R}^L)$ of continuous functions $\mathbf{f} = (f_1, ..., f_L) : I \to \mathbb{R}^L$, where I is a compact interval of $\mathbb{R}$. Let us consider a dataset F_N constituted of $N \in \mathbb{N}$ sample observations of this process, which we indicate by $\mathbf{f}_1, \ldots, \mathbf{f}_N$, $\mathbf{f}_j = (f_{j1}, ..., f_{jL})$. In Ieva and Paganoni (2017) the following inequality is proved:

$$MBD^J_{\{\mathbf{f_1},\ldots,\mathbf{f_n}\}}(\mathbf{f}) \leq a_0 + a_1 MEI_{\{\mathbf{f_1},\ldots,\mathbf{f_n}\}}(\mathbf{f}) + a_2 n^2 (MEI_{\{\mathbf{f_1},\ldots,\mathbf{f_n}\}}(\mathbf{f}))^2, \tag{20}$$

where $a_0 = a_2 = -2/(n(n-1))$ and $a_1 = 2(n+1)/(n-1)$, and $MBD^J_{\{\mathbf{f_1},\ldots,\mathbf{f_n}\}}(\mathbf{f})$ and $MEI_{\{\mathbf{f_1},\ldots,\mathbf{f_n}\}}(\mathbf{f})$ are defined in (9) and (10), respectively.
So the outliergram for multivariate functional data is constructed in analogy with the univariate one and based on the quadratic boundary 20. The function `multivariate_outliergram` displays the outliergram of a multivariate functional dataset of class `mfData` (see Figure 9) and returns a vector of

observation IDs indicating the outlying observations in the dataset.

```
multivariate_outliergram(mfD, Fvalue = 2, shift=TRUE)

$Fvalue
2

$Depth
[1] 0.0386524565 0.0267086844 0.1297124989 0.0474675556
[5] 0.0296031652 0.0446769503 0.0353957777 0.0294200492


$ID_outliers
[1]  12 18 32 47 70 83 91 96
```

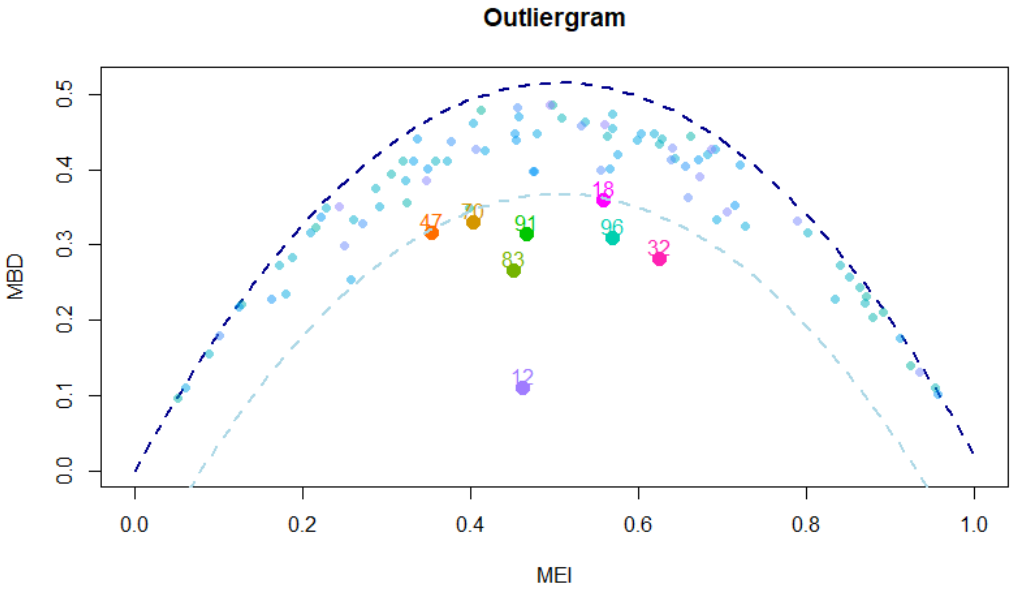


Figure 9: The outliergram of simulated data presented in Figure 4.

Conclusions

In this paper we have described the implementation in the **roahd** package of several statistical methods that deal with the robust statistical analysis of univariate and multivariate functional data, and some graphical tools mainly aimed at identifying and discarding outliers from a dataset of (potentially multivariate) functional data. The package should simplify the access and use of these strongly nonparametric methods to perform a suitable robust inferential analysis of high dimensional and complex data.

Bibliography

A. Arribas-Gil and J. Romo. Shape outlier detection and visualization for functional data: The outliergram. *Biostatistics*, 15(4):603–619, 2014. [p303]

G. Claeskens, M. Hubert, L. Slaets, and K. Vakili. Multivariate functional halfspace depth. *Journal of the American Statistical Association*, 109(505):411–423, 2014. [p292]

X. Dai, P. Z. Hadjipantelis, K. Han, and H. Ji. *Fdapace: Functional Data Analysis and Empirical Dynamics*, 2018. URL https://CRAN.R-project.org/package=fdapace. R package version 0.4.0. [p291]

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993. [p300]

M. Febrero-Bande and M. Oviedo de la Fuente. Statistical computing in functional data analysis: The R package fda.usc. *Journal of Statistical Software*, 51(4):1–28, 2012. [p291]

F. Ferraty and P. Vieu. *Nonparametric Functional Data Analysis: Theory and Practice*. Springer-Verlag, New York, 2006. [p291]

L. Horvath and P. Kokoszka. *Inference for Functional Data with Applications*. Springer-Verlag, New York, 2012. [p291]

F. Ieva and A. M. Paganoni. Depth measures for multivariate functional data. *Communication in Statistics - Theory and Methods*, 42(7):1265 – 1276, 2013. [p292, 303]

F. Ieva and A. M. Paganoni. Component-wise outlier detection methods for robustifying multivariate functional samples. *Statistical Papers*, 2017. URL https://doi.org/10.1007/s00362-017-0953-1. [p297, 298, 303, 304]

F. Ieva, A. M. Paganoni, D. Pigoli, and V. Vitelli. Multivariate functional clustering for the morphological analysis of ecg curves. *Journal of the Royal Statistical Society C*, 62(3):401 – 418, 2013. [p294, 297]

F. Ieva, F. Palma, and J. Romo. Bootstrap-based inference for dependence in multivariate functional data. MOX Report 30/2018, Politecnico di Milano, 2018. URL https://www.mate.polimi.it/biblioteca/add/qmox/30-2018.pdf. [p300]

P. Kokoszka and M. Reimherr. *Introduction to Functional Data Analysis*. Chapman & Hall, 2017. [p291]

R. Liu and K. Singh. A quality index based on data depth and multivariate rank tests. *Journal of the American Statistical Association*, 88(421):252 – 260, 1993. [p291]

R. Liu, J. M. Parelius, and K. Singh. Multivariate analysis by data depth: Descriptive statistics, graphics and inference. *The Annals of Statistics*, 27(3):783–858, 1999. [p291]

S. Lopez-Pintado, Y. Sun, and M. G. Genton. Simplicial band depth for multivariate functional data. *Advances in Data Analysis and Classification*, 8:321–338, 2014. [p292]

S. López-Pintado and J. Romo. Depth-based inference for functional data. *Computational Statistics & Data Analysis*, 51(10):4957–4968, 2007. [p292]

S. López-Pintado and J. Romo. On the concept of depth for functional data. *Journal of the American Statistical Association*, 104(486):718 – 734, 2009. [p292, 296]

S. López-Pintado and J. Romo. A half-region depth for functional data. *Computational Statistics & Data Analysis*, 55(4):1679–1695, 2011. [p296, 297]

R. A. Maronna and R. H. Zamar. Robust estimates of location and dispersion for high-dimensional datasets. *Technometrics*, 44(4):307–317, 2002. [p302]

J. O. Ramsay and B. W. Silverman. *Applied Functional Data Analysis: Methods and Case Studies*. Springer-Verlag, New York, 2002. [p291]

J. O. Ramsay and B. W. Silverman. *Functional Data Analysis*. Springer-Verlag, New York, 2nd edition, 2005. [p291]

J. O. Ramsay, G. Hooker, and S. Graves. *Functional Data Analysis with R and MATLAB*. Springer-Verlag, 1st edition, 2009. [p291]

J. O. Ramsay, H. Wickham, S. Graves, and G. Hooker. *Fda: Functional Data Analysis*, 2014. URL https://CRAN.R-project.org/package=fda. R package version 2.4.4. [p291]

C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. [p294]

H. L. Shang and R. Hyndman. *Rainbow: Bagplots, Boxplots and Rainbow Plots for Functional Data*, 2019. URL https://CRAN.R-project.org/package=rainbow. R package version 3.6. [p291]

Y. Sun and M. G. Genton. Functional boxplots. *Journal of Computational and Graphical Statistics*, 20(2): 316–334, 2011. [p301]

Y. Sun and M. G. Genton. Adjusted functional boxplots for spatio-temporal data visualization and outlier detection. *Environmetrics*, 23(1):54–64, 2012. [p302]

N. Tarabelloni, F. Ieva, R. Biasi, and A. M. Paganoni. Use of depth measure for multivariate functional data in disease prediction: An application to electrocardiograph signals. *The international journal of biostatistics*, 11(2):189–201, 2015. [p296, 298]

N. Tarabelloni, A. Arribas-Gil, F. Ieva, A. M. Paganoni, and J. Romo. *Roahd: Robust Analysis of High Dimensional Data*, 2017. URL https://CRAN.R-project.org/package=roahd. R package version 1.3. [p291]

J. D. Tucker. *Fdasrvf: Elastic Functional Data Analysis*, 2017. URL https://CRAN.R-project.org/package=fdasrvf. R package version 1.8.3. [p291]

J. Tuckey. Mathematics and the picturing of data. In *Proceedings of the International Congress of Mathematicians, Vancouver*, volume 2, pages 523 – 531, 1975. [p291]

D. Valencia, J. Romo, and R. Lillo. A kendall correlation coefficient for functional dependence. Technical report, Universidad Carlos III de Madrid, http://EconPapers.repec.org/RePEc:cte:wsrepe:ws133228, 2015a. [p298, 299]

D. Valencia, J. Romo, and R. Lillo. Spearman coefficient for functions. Technical report, Universidad Carlos III de Madrid, http://EconPapers.repec.org/RePEc:cte:wsrepe:ws133329, 2015b. [p298, 299]

A. Zeileis. CRAN task views. *R News*, 5(1):39–40, 2005. URL https://CRAN.R-project.org/doc/Rnews/. [p291]

Y. Zuo and R. Serfling. General notions of statistical depth function. *The Annals of Statistics*, 28(2): 461–482, 2000. [p291]

Francesca Ieva
MOX - Department of Mathematics
Politecnico di Milano
Italy
francesca.ieva@polimi.it

Anna Maria Paganoni
MOX - Department of Mathematics
Politecnico di Milano
Italy
anna.paganoni@polimi.it

Juan Romo
Department of Statistics
Universidad Carlos III de Madrid
Spain
juan.romo@uc3m.es

Nicholas Tarabelloni
MOX - Department of Mathematics
Politecnico di Milano
Italy
nicholas.tarabelloni@polimi.it

The IDSpatialStats R Package: Quantifying Spatial Dependence of Infectious Disease Spread

by John R. Giles, Henrik Salje, and Justin Lessler

Abstract Spatial statistics for infectious diseases are important because the spatial and temporal scale over which transmission operates determine the dynamics of disease spread. Many methods for quantifying the distribution and clustering of spatial point patterns have been developed (e.g. *K*-function and pair correlation function) and are routinely applied to infectious disease case occurrence data. However, these methods do not explicitly account for overlapping chains of transmission and require knowledge of the underlying population distribution, which can be limiting when analyzing epidemic case occurrence data. Therefore, we developed two novel spatial statistics that account for these effects to estimate: 1) the mean of the spatial transmission kernel, and 2) the τ-statistic, a measure of global clustering based on pathogen subtype. We briefly introduce these statistics and show how to implement them using the IDSpatialStats R package.

Introduction

The transmission process which drives an epidemic can be characterized by the spatial distance separating linked cases. When these transmission events accumulate over time, they are observed as areas of elevated disease prevalence. Knowledge of the extent of the affected area and where new cases may arise is crucial for many disease control strategies (e.g. ring vaccination, vector control etc). In epidemiology, case occurrence data—(x,y) coordinates with temporal information (t) and other covariates—are often used to understand these types of infectious disease dynamics. These data are typically treated as a generic point process so that they can be described in terms of spatial intensity (expected number of cases per unit area) or clustering due to spatial dependence (covariance in x,y space).

In the broader field of spatial statistics, there are many methods that measure the spatial intensity or clustering of a generic point process on a Cartesian (x,y) coordinate system (Table 1). These methods primarily fall into three categories: first-order first-moment (FOFM), first-order second-moment (FOSM), and second-order second-moment (SOSM). The FOFM measures use quadrature (aggregate counts of points within cells) to quantify intensity of the point pattern continuously over (x,y) space. Packages such as **lgcp** (Taylor et al., 2013, 2015) and **ppmlasso** (Renner and Warton, 2013) allow users to model first-order intensity as a count process using a regressive function of (x,y) coordinates and other covariates. The FOSM measures—Moran's I and Geary's C (Moran, 1950; Geary, 1954)—also use quadrature, but they describe general covariance among cell counts across the (x,y) dimensions. These spatial statistics can be calculated using the **spdep** R package (Bivand and Piras, 2015). The SOSM measures, such as the *K*-function and its non-cumulative analogue, the pair correlation function, quantify clustering among neighboring points. Both the FOSM and SOSM measures are considered global spatial statistics because they describe spatial dependence for the entire study area. However, the SOSM can further describe how spatial dependence changes as a function of distance by comparing the observed intensity of neighboring points within distance d to that expected under complete spatial randomness. The *K*-function and pair correlation function can be calculated using the **ads** (Pélissier and Goreaud, 2015), **spatstat** (Baddeley et al., 2016), and **splancs** Rowlingson and Diggle (2017) R packages.

These classic spatial measures are limited in their ability to describe infectious disease dynamics primarily because they treat case occurrence data as a generic point process. The FOFM and FOSM measures use quadrature, which make them vulnerable to error associated with data aggregation (Robinson, 2009) and the modifiable areal unit problem (Openshaw and Taylor, 1979). The SOSM measures, like the *K*-function and pair correlation function, are more common in epidemiology. However, their statistical interpretation is less intuitive in terms of classic epidemiological quantities of relative disease risk, such as the incidence rate ratio or hazard ratio. Additionally, even the temporal forms of these functions (e.g. the space-time *K*-function) do not capture the typical distances traveled in a single transmission generation as they quantify the overall spatial dependence between all cases, not just those epidemiologically linked. The mean distance between sequential cases in a transmission chain is an important epidemiological quantity because it provides insight into potential mechanisms driving spread as well as helping inform interventions. Therefore, we developed novel measures that build upon concepts in spatial statistics to characterize infectious disease spread using case occurrence

Table 1: A selective list of R packages for the analysis of spatial point pattern data. This list is not exhaustive. Visit the *Spatial* CRAN Task View for a more comprehensive list of resources.

Package	Description	Citation
ads	K function for enclosed point patterns	Pélissier and Goreaud (2015)
DCluster	disease clustering for count data	Gómez-Rubio et al. (2005)
lgcp	modeling point patterns with log-Gaussian Cox processes	Taylor et al. (2013, 2015)
ppmlasso	modeling point patterns with LASSO regularization	Renner and Warton (2013)
SGCS	third order clustering of point processes	Rajala (2017)
sparr	spatial relative risk functions with kernel smoothing	Davies et al. (2011)
spatstat	comprehensive tools for analyzing point patterns in many dimensions	Baddeley et al. (2016)
spdep	classic statistics to test for spatial dependence	Bivand and Piras (2015)
splancs	kernel smoothing and Poisson cluster processes	Rowlingson and Diggle (2017)

data. Importantly, these measures are robust to heterogeneities in the underlying population, and substantial case under-reporting, which is common in epidemiology.

We describe these two measures of spatial dependence for infectious diseases and show how they can be calculated with the **IDSpatialStats** R package in the following three sections. First, we introduce a function which simulates infectious disease spread as a spatial branching process. This function is primarily intended to simulate example datasets for the `est.transdist` family of functions and τ-statistics that use temporal information to indicate linked cases. Second, we demonstrate how to estimate the mean and standard deviation of the spatial transmission kernel (Salje et al., 2016b). Estimating the spatial transmission kernel requires an understanding of the number of transmission generations separating cases at different time points of the epidemic. This method provides a measure of fine-scale spatial dependence between two cases, which can be interpreted as the mean distance between sequential cases in a transmission chain. Third, we describe a measure of global clustering—the τ-statistic—that calculates the relative risk of infection given some criteria to identify cases closely related along a chain of transmission (Lessler et al., 2016). The τ-statistic is a global clustering statistic—like the K-function and pair correlation function—that provides an overall measure of clustering for the entire course of an outbreak. Depending on the parameterization, the τ-statistic represents the odds of observing another case with distance d of an infected case compared with either the underlying population or other pathogen types. The following sections contain a brief introduction to each statistic to provide context to the code implementation—for more detailed description of each statistic, see Lessler et al. (2016) and Salje et al. (2016b).

We have implemented these tools in the **IDSpatialStats** R package version 0.3.5 and above. The latest stable release depends on the `doParallel` and `foreach` packages (Microsoft and Weston, 2017; Corporation and Weston, 2018) and can be downloaded from CRAN. A development version of the package is also available on Github at `https://github.com/HopkinsIDD/IDSpatialStats`.

Simulating spatial disease spread

We use a stochastic spatial branching process to simulate epidemiological data in the `sim.epidemic` function. Simulations begin with an index case at $(x, y, t) = (0, 0, 0)$ and transmission events that link two cases follow according to a random Markov process in (x, y) space (i.e. Brownian motion). The number of transmission events occurs according to a Poisson distribution, with its mean and variance set to the effective reproduction number R of the infecting pathogen. The spatial distance traversed by each transmission event is given by a user specified probability distribution which serves as the dispersal kernel function. When specifying the dispersal kernel, the `trans.kern.func` argument expects a list object containing a probability distribution function and its named arguments. For example, to simulate an epidemic where transmission typically occurs at the local level, but long distance transmission events sometimes occur, an exponential transmission kernel might be used because of its long tail. Alternatively, if transmission is expected to consistently occur within a given range, then a normal kernel may be more appropriate.

In simulations where the basic reproductive number R_0 is used to define a constant R-value and $R_0 > 1$, the number of cases will continue to increase with each time step. This effect may not be appropriate when simulating settings where intervention efforts or depletion of susceptible individuals causes heterogeneity in R over the course of the epidemic. Thus, the `sim.epidemic`

function accepts either a scalar value for a constant R value or a vector of R values with a length equal to `tot.generations`, allowing simulations with a variable R value, as shown in the following R code.

```
# Epidemic simulations with variable R value
R1 <- 2
R2 <- 0.5
tot.gen <- 12
R.vec <- seq(R1, R2, (R2 - R1)/(tot.gen - 1))
dist.func <- alist(n=1, a=1/100, rexp(n, a))
sim <- sim.epidemic(R=R.vec, gen.t.mean=7, gen.t.sd=2, min.cases=100,
                    tot.generations=tot.gen, trans.kern.func=dist.func)

head(sim, n=4)
          x           y t
1   0.00000   0.000000 0
2  24.46125   3.280527 3
3 -60.73475 184.885784 7
4 -12.79933 -57.798696 4

sim.plot(sim)
```

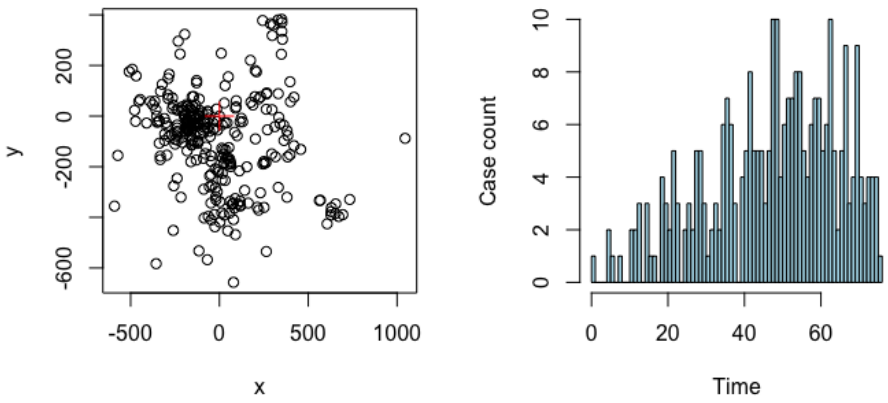


Figure 1: Left: the spatial distribution of simulated cases with the red cross showing the index case. Right: the epidemic curve for a simulation with an R value decreasing from 2 to 0.5 over the course of the epidemic.

The mean transmission distance

In Salje et al. (2016b), we introduced a method to estimate the mean and standard deviation of the spatial transmission kernel using case occurrence data. These data include location (x, y) and onset time t of each case (case times) and the infecting pathogen's generation time $g(x)$. To estimate these spatial statistics, we use the Wallinga-Teunis (WT) method (Wallinga and Teunis, 2004) to probabilistically estimate the number of transmission events required to link two cases, denoted as θ. In settings where a phylogenetic model or contact tracing provide information on transmission pathways, the spatial kernel can be empirically estimated using the distribution of observed distances among all linked cases. The mean and standard deviation of this kernel can then be calculated for any time interval between t_1 and t_2 to give $\mu_t^{obs}(t_1, t_2)$, with the assumption that the number of transmission events separating all case pairs is homogeneous ($\theta = 1$). When data that indicate case linkage are lacking, this assumption is incorrect because the distance between two cases depends on the number of transmission events separating them. In this case, the mean transmission distance at each time interval μ_t must be estimated as a weighted mean:

$$\mu_t(t_1, t_2, \mu_k, \sigma_k) = \sum_i w(\theta = i, t_1, t_2) \cdot \mu_a(\theta = i, \mu_k, \sigma_k).$$

Where, $w(\theta = i, t_1, t_2)$ gives the weight for each of the i elements of θ and the second term $\mu_a(\theta = i, \mu_k, \sigma_k)$ gives the mean distance separating case pairs that are linked by the ith value of θ.

We have implemented four nested functions that are used to estimate $w(\theta = i, t_1, t_2)$ and describe them briefly below. Listed in order, they are comprised of `est.wt.matrix.weights`, `est.wt.matrix`, `get.transdist.theta`, and `est.transdist.theta.weights`. Although, these functions are documented separately, they are all driven by the `est.transdist` family of functions and do not need to be run manually unless desired.

Wallinga-Teunis matrices

The `est.wt.matrix.weights` function builds upon code from the `R0` package (Obadia et al., 2012) to calculate the basic WT matrix (Wallinga and Teunis, 2004). This matrix gives the probability that a case at time t_i (rows) infected a case at time t_j (columns), i.e. $\theta = 1$, based on the generation time distribution of the pathogen $g(x)$. For an epidemic with t unique case times, `est.wt.matrix.weights` gives a $T \times T$ matrix.

The `est.wt.matrix` function produces a WT type matrix for all infector-infectee case pairs. Given the WT matrix produced by `est.wt.matrix.weights` and total case count n, this function calculates an $n \times n$ matrix giving the probability that case i (rows) infected case j (columns). The WT matrix object can be handed directly to `est.wt.matrix` via the `basic.wt.weights` argument, or if this argument is `NULL`, the `est.wt.matrix.weights` function is called automatically.

```
# Calculating Wallinga-Teunis matrices
case.times <- c(1,2,2,3,3) # times each case occurs
g.x <- c(0, 2/3, 1/3, 0, 0) # hypothetical generation time of a pathogen

mat.wts <- est.wt.matrix.weights(case.times=case.times, gen.t.dist=g.x)

# Calculate infector-infectee Wallinga-Teunis matrix
wt.mat1 <- est.wt.matrix(case.times=case.times, gen.t.dist=g.x,
                         basic.wt.weights=mat.wts)
wt.mat2 <- est.wt.matrix(case.times=case.times, gen.t.dist=g.x)

identical(wt.mat1, wt.mat2) # the two methods are equivalent
[1] TRUE
```

Estimation of θ weights

The `get.transdist.theta` function estimates the number of transmission events θ separating pairs of cases using the probabilities in the infector-infectee WT matrix produced by the `est.wt.matrix` function. Sampling all possible transmission trees is impractical for most datasets, so this function constructs a transmission tree by randomly selecting the infector of each case in the epidemic and then θ is determined by finding the product of all probabilities in the chain of transmission that link the randomly sampled case pairs.

The object `theta.wts` (in the code segment below) contains a three-dimensional array [i,j,k], where the rows i and columns j represent unique case times and the third dimension k is the number of transmission events θ. Each cell gives the probability that two cases occurring at times i and j are connected by θ transmission events in the randomly sampled transmission tree. Probabilities in each [i,j,$\cdot$] row are normalized across all θ values. The `get.transdist.theta` function samples a single randomized transmission tree from the epidemic data, therefore we want to simulate many iterations of this random sampling to get a better estimate of the true distribution of θ.

The `est.transdist.theta.weights` function estimates the distribution of θ across all t_i and t_j combinations by simulating many iterations of transmission trees using the `get.transdist.theta` function. Its output is the same as the `get.transdist.theta` function, however, it represents the normalized probabilities after `n.rep` number of simulations.

```
# Estimate theta weights
case.times <- c(1,2,2,3,3) # times each case occurs
g.x <- c(0, 2/3, 1/3, 0, 0) # hypothetical generation time distribution of a pathogen
gen.time <- 1 # mean generation time
n.gen <- round((max(case.times) - min(case.times)) / gen.time) + 1 # total generations

# Calculate infector-infectee Wallinga-Teunis matrix
wt.mat <- est.wt.matrix(case.times=case.times, gen.t.dist=g.x)
```

```
# Estimated theta weights from five randomized transmission trees
theta.wts <- est.transdist.theta.weights(case.times=case.times, n.rep=5,
                                         gen.t.mean=gen.time, t1=0, t.density=g.x)
theta.wts[,,1]
      [,1]   [,2] [,3]
[1,] 0.000    NaN  NaN
[2,] 0.625 0.0000  NaN
[3,] 0.000 0.4375    0
```

Estimating mean of transmission kernel

To estimate the mean transmission distance over the duration of the epidemic we must use the observed distances between case pairs given the time they occurred $\mu_t^{obs}(t_i, t_j)$ and combine them into an overall estimate of the mean of the transmission kernel μ_k. The workhorse function `est.trandsdist` estimates the overall mean μ_k and standard deviation σ_k of the kernel. This function first calls the `est.wt.matrix.weights`, `est.wt.matrix`, `get.transdist.theta`, and `est.transdist.theta.weights` functions described above to estimate the distribution of θ across all case pairs and then calculates each of the weights $w(\theta = i, t_1, t_2)$. The weights are calculated as the proportion of all case pairs occurring at t_i and t_j that are separated by each estimated θ over all simulations:

$$\hat{w}(\theta = i, t_1, t_2) = \frac{\sum_{k=1}^{N_{sim}} \sum_{i=1}^{n} \sum_{j=1}^{n} \boldsymbol{I}_1(t_i = t_1, t_j = t_2, \Theta_{ij} = \theta)}{N_{sim} \sum_{i=1}^{n} \sum_{j=1}^{n} \boldsymbol{I}_2(t_i = t_1, t_j = t_2)}.$$

Here, the functions $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ indicate if two cases occurred at time t_i and t_j and were linked by θ transmission events, or if they just occurred at t_i and t_j respectively. In words this can be written as:

$$\hat{w}(\theta = i, t_1, t_2) = \frac{\text{Total cases at } t_1 \text{ and } t_2 \text{ across all simulations separated by } \theta \text{ transmission events}}{\text{Total cases at } t_1 \text{ and } t_2 \text{ across all simulations}}.$$

Once the weights of the θ values are estimated, the `est.transdist` function calculates μ_k and σ_k as the average weighted estimate over all combinations of t_i and t_j. If we now let k index the vector of θ values, then:

$$\hat{\mu}_k = \hat{\sigma}_k = \frac{1}{\sum_i \sum_j n_{ij}} \sum_i \sum_j \frac{2 \cdot \mu_t^{obs}(t_i, t_j) \cdot n_{ij}}{\sum_k \hat{w}(\theta = k, t_i, t_j) \cdot \sqrt{2\pi k}}.$$

For a derivation of these equations, see sections 2.3 and 2.4 of Salje et al. (2016b).

The `est.trandist` function requires case occurrence data—a matrix with three columns [*x*,*y*,*t*]—and the mean and standard deviation of the infecting pathogen's generation time (for calculating WT matrices) as input. The function returns estimates of μ_k and σ_k of the spatial transmission kernel. These estimates are made under the assumption that $\mu_k = \sigma_k$, so the upper bound of $\hat{\mu}_k$ and $\hat{\sigma}_k$ are also provided for when this assumption is violated. Bound estimates are equal to $\sqrt{2}$ times the values estimated under the $\mu_k = \sigma_k$ assumption (see section 2.5 of Salje et al. (2016b)). Additional constraints on the estimation of μ_k and σ_k can be defined in the remaining arguments, such as the time step in which the analysis should begin (`t1`), the maximum number of time steps (`max.sep`) and maximum spatial distance (`max.dist`) to consider when estimating θ, and the number of randomized transmission tree simulations to run (`n.transtree.reps`).

To estimate the uncertainty around $\hat{\mu}_k$ due to sampling or observation error, we have implemented a wrapper function called `est.transdist.bootstrap.ci` that performs bootstrap iterations using the `est.transdist` function. Upon each iteration, the epidemiological data are resampled with replacement and μ_k is re-estimated. The `est.transdist.bootstrap.ci` function contains all the same arguments as the `est.transdist` function, with additional arguments defining the number of bootstrapped iterations to perform, the high and low boundaries of the desired confidence interval, and options for running the bootstrap analysis in parallel.

When parallel computation is enabled (the default is `parallel = FALSE`), the function uses the `makeCluster()` function of the `parallel` package to make the appropriate cluster type for the operating system of the local machine (SOCK cluster for Windows or a Fork cluster for Unix-like machines). The cluster is then registered as the parallel backend for the `foreach` package, which is used to run the bootstraps in parallel. The user can define the number of cores to use when running in parallel using the `n.cores` argument. If `parallel = TRUE` and `n.cores = NULL`, the function will use half the total cores on the local machine.

```
# Estimate transmission distance for simulated data
set.seed(123)
```

```
dist.func <- alist(n=1, a=1/100, rexp(n, a)) # Dispersal kernel
sim <- sim.epidemic(R=2, gen.t.mean=7, gen.t.sd=2, min.cases=100,
                    tot.generations=8, trans.kern.func=dist.func)

# Estimate mean transmission distance
sim.transdist <- est.transdist(epi.data=sim, gen.t.mean=7, gen.t.sd=2, t1=0,
                               max.sep=1e10, max.dist=1e10, n.transtree.reps=10)

sim.transdist
$mu.est
[1] 92.79699
$sigma.est
[1] 91.45614
$bound.mu.est
[1] 131.2348
$bound.sigma.est
[1] 129.3385

# Estimate confidence intervals around mean
sim.transdist.ci <- est.transdist.bootstrap.ci(epi.data=sim,
                                               gen.t.mean=7,
                                               gen.t.sd=2,
                                               t1=0,
                                               max.sep=1e10,
                                               max.dist=1e10,
                                               n.transtree.reps=10,
                                               boot.iter=5,
                                               ci.low=0.025,
                                               ci.high=0.975)
sim.transdist.ci
$mu.est
[1] 131.2124
$mu.ci.low
2.5%
128.2505
$mu.ci.high
97.5%
134.3312
```

Change in mean transmission distance over time

An estimate of μ_k over the duration of an epidemic is indicative of the overall spatial dependence. However, conditions may change over the course of an epidemic that alter the spatial scale upon which transmission operates. To quantify temporal heterogeneity in the mean transmission distance, we have implemented the `est.transdist.temporal` and `est.transdist.temporal.bootstrap.ci` functions, which estimate the change in $\hat{\mu}_k$ over time and its bootstrapped confidence intervals respectively.

When applying the temporal versions of the `est.transdist` functions, it is important to consider the sample size at each time step because the `est.transdist.temporal` function estimates μ_k for all cases leading up to each unique time step. Some time steps at the beginning of an epidemic may be returned as `NA` if there are not enough unique cases to estimate μ_k. Furthermore, in scenarios where time steps in the beginning of an epidemic have low sample sizes, such as an epidemic with a low R_0, $\hat{\mu}_k$ may be over- or under-estimated and display larger confidence intervals due to sampling error. Therefore, we recommend either setting the `t1` argument to the first time step that contains a sufficient sample size, or plotting results along with cumulative sample size as we have done in Figure 3.

```
# Estimate temporal transmission distance for simulated data
set.seed(123)
dist.func <- alist(n=1, a=1/100, rexp(n, a)) # Dispersal kernel
sim <- sim.epidemic(R=2, gen.t.mean=7, gen.t.sd=2, min.cases=100,
                    tot.generations=8, trans.kern.func=dist.func)

# Estimate mean and confidence intervals at each time step
sim.temp.transdist.ci <- est.transdist.temporal.bootstrap.ci(epi.data=sim,
```

```
                                                gen.t.mean=7,
                                                gen.t.sd=2,
                                                t1=0,
                                                max.sep=1e10,
                                                max.dist=1e10,
                                                n.transtree.reps=10,
                                                boot.iter=5,
                                                ci.low=0.025,
                                                ci.high=0.975)

head(sim.temp.transdist.ci)
   t    pt.est    ci.low   ci.high n
1  0        NA        NA        NA 1
2  3        NA        NA        NA 2
3  8        NA        NA        NA 3
4  9  44.61359  35.52047  52.24099 4
5 10 101.55313  43.99469 203.11247 5
6 11 189.29767 113.79560 224.79960 6
```

Application to foot-and-mouth disease

To provide an example of how the functions shown above can be applied to real data, we estimate the mean transmission distance for the 2001 foot-and-mouth epidemic among farms in Cumbria, UK. These data can be found in the `fmd` data object included in the `sparr` package (Davies et al., 2011). It contains transformed (x,y) coordinates of the infected farms and the time step (t) in which it was infected, which is given in days since the index farm was infected (Figure 2). The generation time for foot-and-mouth disease is estimated to have a mean of 6.1 days and a standard deviation of 4.6 days (Haydon et al., 2003), so we use these in the `gen.t.mean` and `gen.t.sd` arguments in the `est.transdist.temporal.bootstrap.ci` function.

```
library(sparr)
data(fmd)
fmd <- cbind(fmd$cases$x, fmd$cases$y, fmd$cases$marks)

head(fmd, n=3)
[,1]     [,2] [,3]
[1,] 333.0328 541.3405   52
[2,] 336.1428 543.3462   46
[3,] 341.4762 551.1794   38

sim.plot(fmd)
```

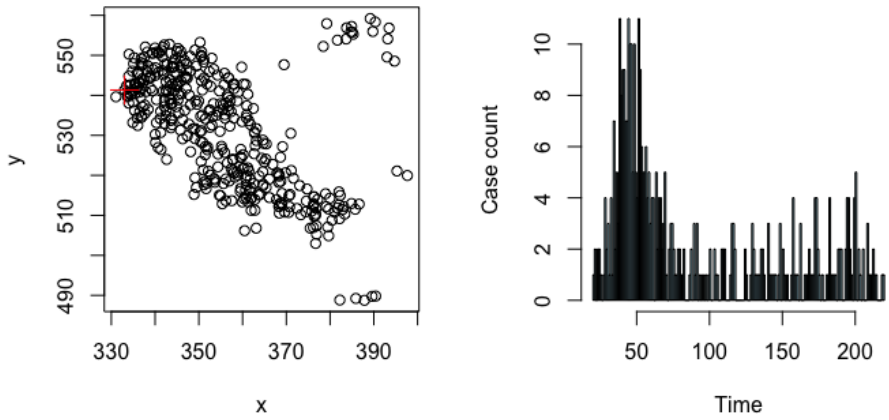


Figure 2: The spatial and temporal distribution of case farms from the 2001 foot-and-mouth epidemic among farms in Cumbria, UK; plotted using the `sim.plot` function. The `x` and `y` axis in the left plot represent transformations of UTM coordinates in kilometers. On the right, case counts are plotted by days since the index case. Data are provided by the `sparr` package (Davies et al., 2011).

```
# NOTE: this function may take a while depending on the data set
fmd.trans <- est.transdist.temporal.bootstrap.ci(epi.data=fmd,
                                                 gen.t.mean=6.1,
                                                 gen.t.sd=4.6,
                                                 t1=0,
                                                 max.sep=1e10,
                                                 max.dist=1e10,
                                                 n.transtree.reps=5,
                                                 boot.iter=10,
                                                 ci.low=0.025,
                                                 ci.high=0.975,
                                                 parallel=TRUE,
                                                 n.cores=detectCores())
par(mfrow=c(1,1))
fmd.trans[,2:4] <- fmd.trans[,2:4]/1000 # Convert to km
plot(fmd.trans$t, fmd.trans$pt.est, pch=19, col='grey', ylim=range(fmd.trans[,3:4], na.rm=T),
     xlab='Time step', ylab='Estimated mean of transmission kernel (km)')

tmp <- seq(1, nrow(fmd.trans), 5)
axis(3, tmp, fmd.trans[tmp,5])
mtext('Sample size (n)', side=3, line=3)

tmp <- which(fmd.trans$n >= 30)[1]
abline(v=tmp, lty=2)
text(16, 1, 'n = 30')

tmp <- tmp:nrow(fmd.trans)
lty <- c(NA,1,2,2)

for(i in 2:4) {
     low <- loess(fmd.trans[tmp,i] ~ as.vector(tmp), span=0.3)
     low <- predict(low, newdata=data.frame(as.vector(tmp)))
     lines(c(rep(NA, tmp[1]), low), lwd=2, lty=lty[i], col='blue')
}
```

Using our approach described above, we estimated the mean transmission distance between case farms in the `sparr` package foot-and-mouth disease data to be $\hat{\mu}_k = 5.8$ km (95% CI = 5.7–6.1 km). Interestingly, this estimate of μ_k is lower than that reported in Salje et al. (2016b), where $\hat{\mu}_k = 9.1$ km (95% CI = 8.4–9.7 km). The difference in $\hat{\mu}_k$ is likely due to differences in data sources. The values estimated in Salje et al. (2016b) include case farms from both Cumbria and Dumbfriesshire, UK with the additional constrain that only case farms where the source farm was confirmed were included. The `sparr` data set, on the other hand, contains all case farms from only Cumbria.

Global clustering: the τ-statistic

Estimating the mean of the spatial transmission kernel (above) provides information on the small spatial scale of individual transmission events. After subsequent generations of transmission where different transmission chains overlap in space, a larger area of elevated disease prevalence will be observed. To describe this larger-scale process, we introduced the τ-statistic in Lessler et al. (2016). The τ-statistic measures global clustering with an epidemiological interpretation—the relative risk of an individual being a related case (under some definition) given they are within a particular distance from another case. The spatial distances where the relative risk is high represent an area of elevated prevalence that is likely to have greater public health utility compared with the scale of individual transmission because interventions must account for ongoing transmission at the population level to contain an outbreak. Therefore, the τ-statistic provides a more intuitive global measure of spatial clustering, which can be interpreted as the relative risk of infection.

Formulation of the τ-statistic has a mathematical relationship to the K-function and pair correlation function. The K-function quantifies the expected number of neighboring points within distance d of a typical point $\mathbf{Z}$ relative to the intensity of the underlying population distribution λ.

$$K(d) = \frac{1}{\lambda} E[\text{neighbors within distance } d \mid x, y \text{ coordinates of } \mathbf{Z}]$$

In the simplest scenario, λ is assumed to be homogeneously distributed, so that $\lambda = N/A$, where N

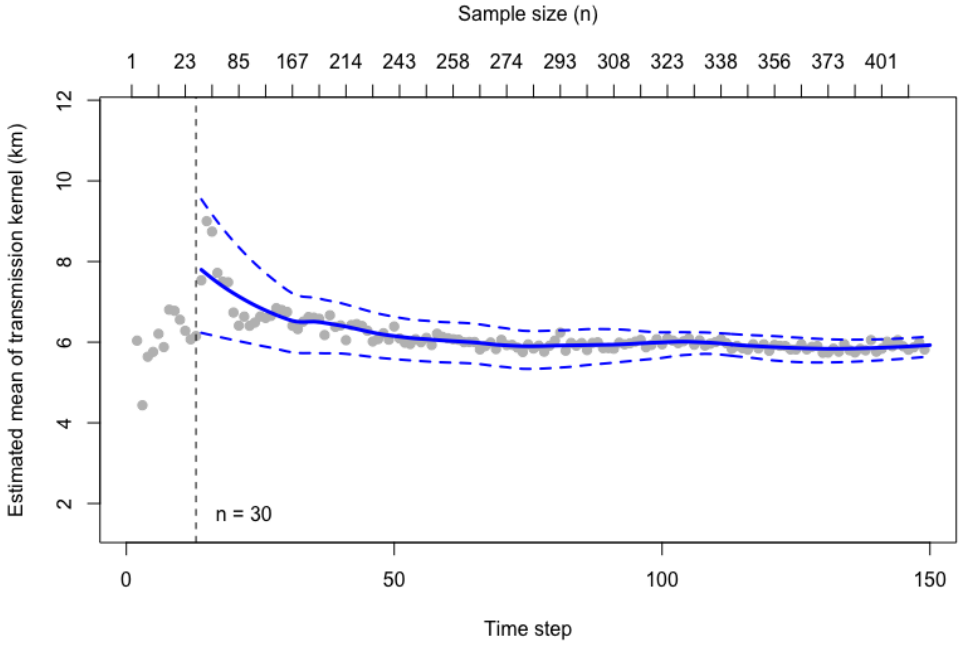


Figure 3: Output from the `est.transdist.temporal.bootstrap.ci` function showing the change in the mean transmission distance over the course of the 2001 foot-and-mouth disease epidemic for case farms in the `fmd` data set in the `sparr` package. The point estimates are plotted as grey circles and a loess smooth of the mean estimate is plotted (blue line) along with its 95% bootstrapped confidence intervals (dashed blue lines). The loess smooth begins with the first time step that contains a cumulative sample size of 30, indicated by the dashed line.

is the total number of cases and A is the total study area. Under the assumption of a heterogeneous underlying population distribution, λ becomes the location specific intensity $\lambda(S)$, where $S \subset A$ within distance d of location $\mathbf{Z}$. In both cases, the K-function is plotted with the theoretical value of the K-function for a homogeneous Poisson process πd^2, which indicates clustering or dispersion relative to complete randomness. The cumulative aspect of the K-function (using all neighbors within distance d) is, however, a well-known constraint that makes it difficult to interpret changes in clustering over distance. The pair correlation function alleviates this constraint by applying the K-function within a distance range (d_1, d_2) and standardizing it by the complete spatial randomness expectation for a homogeneous Poisson process within this range:

$$G(d_1, d_2) = \frac{K(d_1 + \Delta d) - K(d_1)}{2\pi d_1 \Delta d + \pi \Delta d^2},$$

where, $\Delta d = d_2 - d_1$. Both the K-function and pair correlation functions have seen general application due to developments that accommodate inhomogeneous underlying population distribution, clustering between typed points, and edge effects. However, these functions assume complete knowledge of the underlying population distribution and use a null statistical hypothesis of complete spatial randomness, which is precarious for scenarios in epidemiology where the underlying population is unknown and relative risk is used to understand disease dynamics.

To incorporate other metrics of global clustering, the **IDSpatialStats** package provides wrapper functions for calculating both the cross K- and cross pair correlation functions using the `Kcross` and `PCFcross` functions from the **spatstat** package (Baddeley et al., 2016). These wrapper functions allow for straightforward calculation of these statistics using typed epidemiological data that is formatted for **IDSpatialStats** functions (Figure 4).

```
# Calculate cross-K and cross pair correlation functions with simulated data
data(DengueSimRepresentative)

r.vals <- seq(0, 1000, 20)
labs <- seq(0, 1000, 200)

k <- get.cross.K(epi.data=DengueSimRepresentative, type=5, hom=1, het=NULL,
                 r=r.vals, correction='border')
```

```
head(k, n=3)
   r      theo    border
1  0     0.000     0.000
2 20  1256.637  2166.362
3 40  5026.548  5956.887

g <- get.cross.PCF(epi.data=DengueSimRepresentative, type=5, hom=1, het=NULL,
                   r=r.vals, correction='border')
head(g, n=3)
   r theo      pcf
1  0    1 1.000000
2 20    1 1.720848
3 40    1 1.178406

par(mfrow=c(1,2))
plot(k[,3], type='l', lwd=2, xaxt='n', xlab='distance (m)', ylab='cross K function')
lines(k[,2], col='red', lty=2, lwd=2)
axis(1, at=which(r.vals %in% labs), labels=labs)

legend(-3, 4.15e6, legend=c("Theoretical Poisson process", "Observed function"),
       col=c("red", "black"), lty=2:1, box.lty=0, bg='transparent',
       x.intersp=0.7, y.intersp = 1.2)

plot(g$pcf, type='l', lwd=2, xaxt='n', xlab='distance (m)',
     ylab='cross pair correlation function')
abline(h=1, col='red', lty=2, lwd=2)
axis(1, at=which(r.vals %in% labs), labels=labs)
```

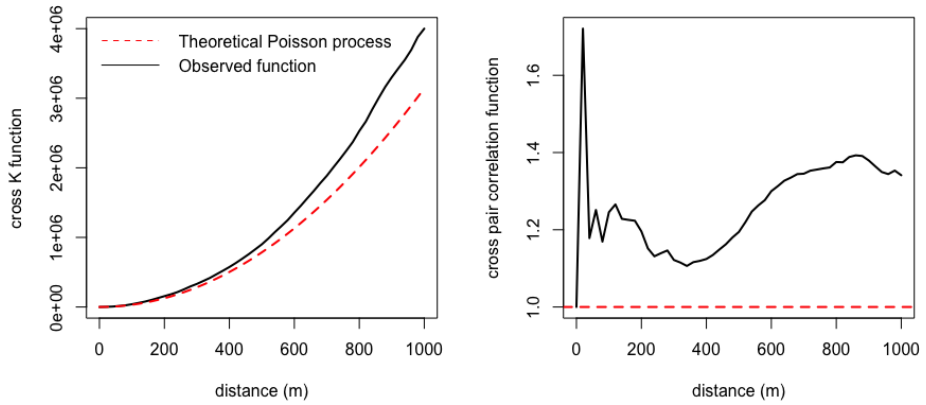


Figure 4: Output from the cross K function (left) and the cross pair correlation function (right) with the observed function shown in black and the value of a theoretical homogeneous Poisson process shown in red.

A measure of relative risk that does not assume knowledge of the underlying population distribution was developed for point pattern data in veterinary epidemiology (Diggle et al., 2005). This function, which is implemented in the `sparr` (Davies et al., 2011) and `spatstat` (Baddeley et al., 2016) packages, uses spatial kernel functions to calculate a ratio of spatial intensity for two different point types $\rho(S) = \lambda_1(S)/\lambda_0(S)$. This formulation can quantify the relative risk for case-control point data or case occurrences with multiple types, but it is not a global clustering statistic.

The τ-statistic can be computed in two ways depending on the underlying assumption that the true population distribution is known. If this assumption is true, then the τ-statistic is similar to other common measures of global clustering that rely on knowledge of the background population distribution to quantify generic clustering of a point process.

$$\hat{\tau}(d_1, d_2) = \frac{\hat{\pi}(d_1, d_2)}{\hat{\pi}(0, \infty)},$$

where $\hat{\pi}(d_1, d_2)$ represents the incidence rate within distance d_1 to d_2 of a case and $\hat{\pi}(0, \infty)$ represents the incidence rate over the entire extent of the study area. This can be implemented by defining the numerator and denominator using the `get.pi` function or by using the generalized `get.tau`

function with `comparison.type = 'representative'` argument. The $\hat{\pi}(\cdot)$ terms in the numerator and denominator use the occurrence data to calculate the incidence rates for linked cases within some defined distance. Therefore, a critical step in performing an analysis with the τ-statistic is specifying which cases are linked through some defined relationship (homologous) and those that are not (non-homologous). Homology can be defined statically or dynamically. When defined statically, the `get.pi.typed` and `get.tau.typed` functions can be used to assign case types based on a `type` column supplied by the data matrix. When defined dynamically, an indicator function $I(\cdot)$ is used to delineate linked and unlinked cases in the data, which allows greater flexibility when defining case type homology.

```
# Calculate tau-statistic using get.pi.typed functions
data(DengueSimRepresentative)

type <- 2 - (DengueSimRepresentative[,'serotype'] == 1)
typed.data <- cbind(DengueSimRepresentative, type=type)
d2 <- seq(20, 1000, 20)
d1 <- d2 - 20

# Static definition of case type homology
num <- get.pi.typed(typed.data, typeA=1, typeB=2, r.low=d1, r=d2)
den <- get.pi.typed(typed.data, typeA=1, typeB=2, r.low=0, r=1e10)
head(num$pi/den$pi, n=4)
[1] 0.2641154 0.2104828 0.2451847 0.2487042

tau <- get.tau.typed(typed.data, typeA=1, typeB=2, r.low=d1, r=d2,
                     comparison.type = "representative") # Equivalent
head(tau, n=4)
  r.low  r       tau
1     0 20 0.2641154
2    20 40 0.2104828
3    40 60 0.2451847
4    60 80 0.2487042

# Calculate tau-statistic using dynamic expression indicating serotype homology
ind.func <- function(a, b){
     if (a[5] == 1 & b[5] == 1) {
          x <- 1
     } else {
          x <- 2
     }
     return(x)
}

num <- get.pi(posmat=DengueSimRepresentative, fun=ind.func, r.low=d1, r=d2)
den <- get.pi(posmat=DengueSimRepresentative, fun=ind.func, r.low=0, r=Inf)
head(num$pi/den$pi, n=4)
[1] 5.084735 4.967885 4.605805 4.409876

tau <- get.tau(posmat=DengueSimRepresentative, fun=ind.func, r.low=d1,  r=d2,
               comparison.type="representative") # Equivalent
head(tau, n=4)
  r.low  r      tau
1     0 20 5.084735
2    20 40 4.967885
3    40 60 4.605805
4    60 80 4.409876

plot(tau$r.low+tau$r/2, tau$tau, type='l', lwd=2, col='blue', xlab='distance (m)')
abline(h=1, lty=2, lwd=2, col='red')
abline(v=100)
```

The interpretation of the τ-statistic is analogous to that of the pair-correlation function in two ways. First, the τ-statistic is not compared to the theoretical measure of a random Poisson process because the metric is an incidence rate ratio with epidemiological meaning. Instead, this measure is plotted in comparison to a ratio of 1, indicating no relative difference in disease risk among homologous cases.

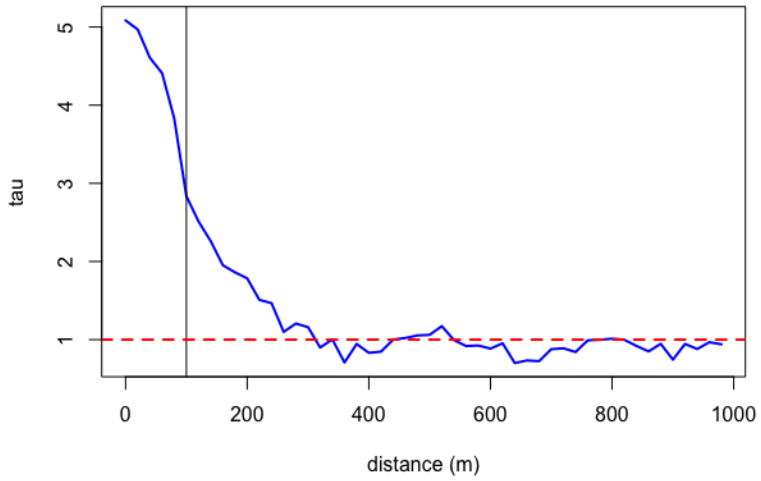


Figure 5: The τ-statistic calculated using the `get.tau` function (blue line) with the theoretical value of no relative difference in disease risk shown by the dashed red line. The vertical black line indicates the mean of the spatial dispersal kernel (100m) used to simulate the `DengueSimRepresentative` data set.

Second, the τ-statistic measures relative risk using case pairs within a distance range $(d_1 \leq d_{ij} < d_2)$. This approach can describe how fine-scale spatial dependence changes over distance. However, if the user wishes to estimate cumulative spatial dependence (analogous to the K-function), then d_1 can be fixed at zero $(0 \leq d_{ij} < d_2)$.

Estimating the τ-statistic with $\hat{\theta}$

If the underlying population distribution is unknown, then the τ-statistic can be computed so that it quantifies global clustering in terms of relative risk. This goes beyond classic measures of clustering by utilizing some relationship between linked and unlinked cases to distinguish transmission chains and quantify relative clustering between them. To do so, we use the function $\hat{\theta}(\cdot)$ which gives the odds ratio of cases related to case i to those independent of i to give an estimate of the τ-statistic that is not biased by assumptions about the underlying population distribution.

$$\hat{\tau}(d_1, d_2) = \frac{\hat{\theta}(d_1, d_2)}{\hat{\theta}(0, \infty)},$$

where,

$$\hat{\theta}(d_1, d_2) = \frac{\sum_{\forall i} \sum_{\forall j} I_1(z_{ij} = 1,\ d_1 \leq d_{ij} < d_2)}{\sum_{\forall i} \sum_{\forall j} I_2(z_{ij} = 0,\ d_1 \leq d_{ij} < d_2)}.$$

The indicator function $I(\cdot)$ is applied to all ij case pairs within distance d_1 and d_2. It returns a binary response which is equal to 1 when case pairs meet user-specified criteria to be homologous and equal to 2 when they are non-homologous. The result is an $i \times j$ relation matrix z_{ij} which is used to find the sums of homologous and non-homologous case pairs. Using an indicator function also allows additional criteria to be used to define case type homology, such as temporal proximity (Figure 6).

```
# Calculate tau-statistic using serotype homology and time
data(DengueSimR01)
d2 <- seq(20, 1000, 20)
d1 <- d2 - 20

# Dynamic expression indicating serotype homology and temporal proximity
ind.func <- function(a, b, t.limit=20){
     if (a[5] == b[5] & (abs(a[3] - b[3]) <= t.limit)){
          x <- 1
     } else {
          x <- 2
     }
     return(x)
}
```

```
num <- get.theta(DengueSimR01, ind.func, r.low=d1, r=d2)
den <- get.theta(DengueSimR01, ind.func, r.low=0, r=Inf)
head(num$theta/den$theta, n=4)
[1] 3.9148969 3.5145802 4.5963608 5.1082210

tau <- get.tau(posmat=DengueSimR01, fun=ind.func, r.low=d1, r=d2,
               comparison.type="independent") # Equivalent
head(tau, n=4)
  r.low  r      tau
1     0 20 3.914897
2    20 40 3.514580
3    40 60 4.596361
4    60 80 5.108221

plot(tau$r, tau$tau, type='l', lwd=2, col='blue', xlab='distance (m)')
abline(h=1, lty=2, lwd=2, col='red')
abline(v=100)
```

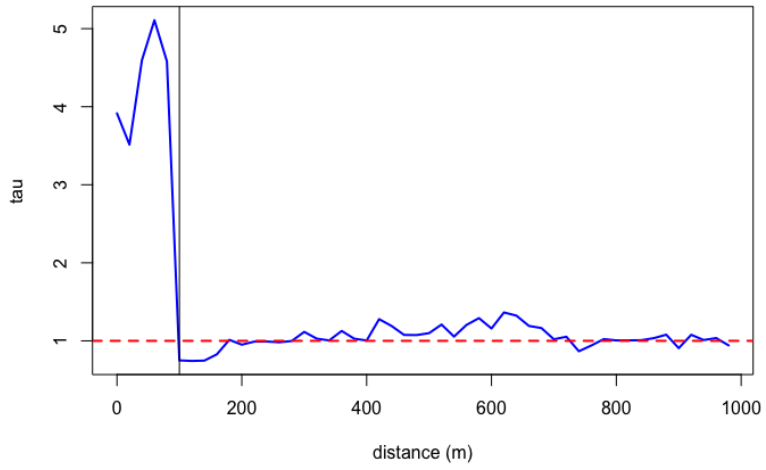


Figure 6: The τ-statistic calculated using `get.tau` with an indicator function based on serotype homology and temporal proximity (blue line) with the theoretical value of no relative difference in disease risk shown by the dashed red line. The vertical black line indicates the mean of the spatial dispersal kernel (100m) used to simulate the `DengueSimR01` data set.

Calculating variance in point estimates

In the examples above, the `get.pi`, `get.theta`, and `get.tau` function families calculate point estimates for $\hat{\pi}$, $\hat{\theta}$, and $\hat{\tau}$ respectively. In scenarios where observation error, sampling bias, or measurement error are expected to introduce additional variance, users may wish to place confidence intervals around these point estimates. For this purpose, each family of functions contains a function ending with a `.bootstrap` suffix, which generates point estimates for `boot.iter` number of bootstrapped samples of the data (Efron and Tibshirani, 1994). Functions ending with a `.ci` suffix are wrappers that calculate user specified confidence intervals based on the bootstrapped samples (Figure 7).

```
# Calculate variance around point estimates of the tau-statistic
data(DengueSimR02)

d2 <- seq(20, 1000, 20)
d1 <- d2 - 20

# Function indicating genotype homology
ind.func <- function(a, b){
     if(a[4] == b[4]){
          x = 1
     } else{
```

```
          x = 2
     }
     return(x)
}

# Bootstrapped estimates of tau
tau.boot <- get.tau.bootstrap(DengueSimR02, ind.func, r.low=d1, r=d2, boot.iter=5)
head(tau.boot, n=4)
  r.low  r       X1       X2       X3       X4       X5
1     0 20 51.04283 49.17736 60.45922 43.36588 37.26332
2    20 40 20.80095 29.62483 26.54935 34.11416 31.32279
3    40 60 34.05415 35.66984 40.21975 31.02943 32.77966
4    60 80 30.52361 35.46972 27.77247 36.64628 32.43156

# Wrapper function of get.tau.bootstrap calculates confidence intervals
tau.ci <- get.tau.ci(DengueSimR02, ind.func, r.low=d1, r=d2, boot.iter=25)
head(tau.ci, n=4)
  r.low  r   pt.est   ci.low  ci.high
1     0 20 44.05161 22.73147 59.44465
2    20 40 30.83943 19.68758 42.05249
3    40 60 37.57434 30.48664 45.78121
4    60 80 33.54134 28.12390 38.76330

plot(tau.ci$r, tau.ci$pt.est, ylim=range(tau.ci[,4:5]), type="l", lwd=2, col='blue',
     xlab='distance (m)', ylab='tau')
lines(tau.ci$r, tau.ci$ci.low, lty=2, lwd=1, col='blue')
lines(tau.ci$r, tau.ci$ci.high, lty=2, lwd=1, col='blue')
abline(h=1, lty=2, lwd=2, col='red')
```

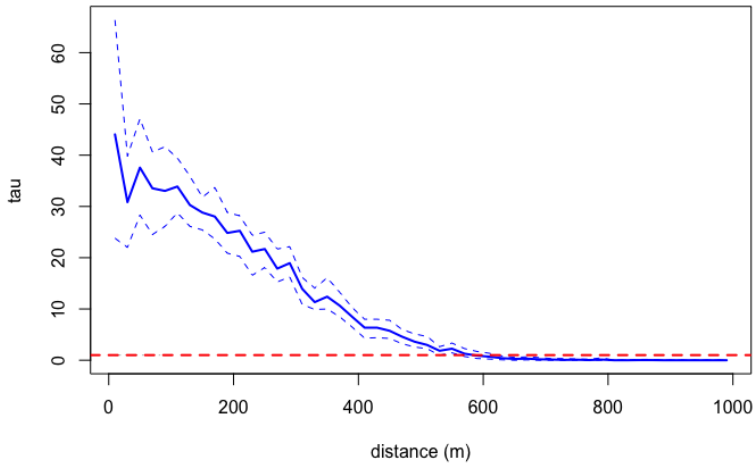


Figure 7: The τ-statistic calculated using `get.tau` with an indicator function based on genotype homology (blue line). The dashed blue lines show the bounds for the 95% confidence intervals calculated by the `get.tau.ci` function. The theoretical value of no relative difference in disease risk shown by the dashed red line.

Null hypothesis testing

A common approach for interpreting spatial clustering statistics includes hypothesis testing using simulation envelopes to assess whether an observed spatial measure is statistically significant (Ripley, 1979; Baddeley et al., 2014). To enable null hypothesis tests, we have implemented a permutation method (Good, 2010) to simulate the nonparametric distribution of $\hat{\pi}$, $\hat{\theta}$, and $\hat{\tau}$ under the null hypothesis of no spatial dependence. The permutation algorithm simulates the null distribution by randomly reassigning case coordinates to observations upon each permutation. Null distributions can be computed using functions ending in the `.permute` suffix and then plotted with observed measures to assess statistical significance as a function of distance (Figure 8).

```
# Compare tau statistic to its null distribution using permutation
data(DengueSimR02)
set.seed(123)

d2 <- seq(20, 1000, 20)
d1 <- d2 - 20

# Compare spatial dependence by time case occurs
type <- 2 - (DengueSimR02[,"time"] < 120)
typed.data <- cbind(DengueSimR02, type=type)

typed.tau <- get.tau.typed(typed.data, typeA=1, typeB=2, r.low=d1, r=d2,
                           comparison.type = "independent")

head(typed.tau, n=4)
  r.low  r       tau
1     0 20 0.4040661
2    20 40 0.5471728
3    40 60 0.7897655
4    60 80 0.8901166

# Perform permutations of observed case times and locations for null distribution
typed.tau.null <- get.tau.typed.permute(typed.data, typeA=1, typeB=2, r.low=d1, r=d2,
                                        permutations=100,
                                        comparison.type = "independent")
head(typed.tau.null[,1:7], n=4)
  r.low  r        X1        X2        X3        X4        X5
1     0 20 1.2570945 0.8530284 3.5019060 1.0326133 1.0775095
2    20 40 1.1448539 0.7045255 0.7224212 2.0742058 0.8754765
3    40 60 0.6947101 0.8904419 0.8249682 0.6990984 0.5128531
4    60 80 1.6266250 1.0916873 0.8326210 0.8432683 1.4815756

# 95% confidence intervals of null distribution
null.ci <- apply(typed.tau.null[,-(1:2)], 1, quantile, probs=c(0.025, 0.975))

plot(typed.tau$r, typed.tau$tau, type='l', lwd=2, ylim=range(c(typed.tau$tau, null.ci)),
     xlab="distance (m)", ylab="tau")
lines(typed.tau$r, null.ci[1,], lty=2)
lines(typed.tau$r, null.ci[2,], lty=2)
abline(h=1, lty=2, lwd=2, col='red')
```

Summary

Conventional spatial statistics are often used to describe the intensity or clustering of point processes. Quantifying spatial dependence of infectious disease spread, however, requires a modified approach that considers overlapping transmission chains and the likelihood of case linkage. Therefore, we have implemented two types of spatial statistics in the **IDSpatialStats** package (the mean transmission distance μ_k, and the τ-statistic) that can be used along with other measures of spatial dependence (e.g. the cross K-function and cross pair correlation function) to understand the spatial spread of infectious diseases.

We showed how to simulate epidemiological data and estimate μ_k, and the τ-statistic, which can be used as templates for other analyses. First, the `transdist` family of functions provides a measure of fine-scale spatial dependence by estimating the mean of the transmission distance $\hat{\mu}_k$ between sequentially linked cases in a transmission chain (Salje et al., 2016b). Second, the `get.tau` family of functions measure spatial dependence on a larger-scale by estimating the τ-statistic, which describes the area of elevated prevalence surrounding cases. This family of functions does so by estimating the relative risk of a case being homologous compared with non-homologous case types. The definition of case type homology is flexible and can utilize temporal or biological information, such as genotype and serotype of the pathogen.

The generalized structure of the `get.tau` family allows for diverse applications of the τ-statistic to epidemiological data. Previous studies have used the τ-statistic to quantify spatial and/or temporal dependence of transmission for Dengue, Cholera, HIV, and Measles disease systems (see Table 2 for

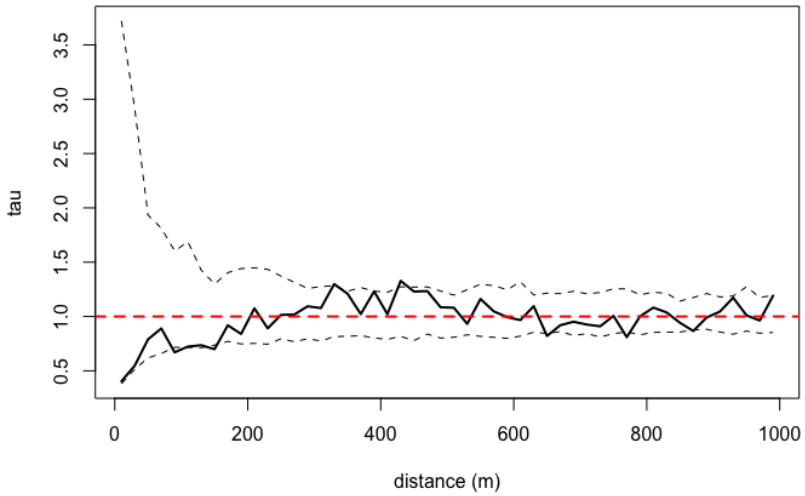


Figure 8: A null hypothesis test for the τ-statistic calculated using the `get.tau.typed.permute` function. The point estimate for $\hat{\tau}$ is shown by the black line and the 95% confidence bounds of permuted $\hat{\tau}$ values are indicated by the dashed lines. The theoretical value of no relative difference in disease risk is shown by the dashed red line. The plot indicates that the point estimates for $\hat{\tau}$ are not statistically significant in this example because they are within the bounds of the null distribution.

detailed descriptions). These studies illustrate that, regardless of the system under study, analyses are enhanced when bootstrapping, permutation tests, and/or assessment of observation error is employed to understand the distribution of error and statistical significance for estimates of the τ-statistic.

The **IDSpatialStats** package is undergoing continued development. Future directions include expanding the implementation of the τ-statistic to facilitate estimation of spatio-temporal dependence by incorporating a temporal interval into the spatial search window. This technique was used by Salje et al. (2012) in the form of the ϕ-statistic to estimate $\hat{\phi}(d_1, d_2, t_1, t_2)$. Additional developments include a theoretical framework for the τ-statistic that incorporates uncertainty due to pathogen generation time, and to define case type homology more continuously using genetic distance matrices. We hope these developments will enable users to address more complex questions and incorporate more sources of uncertainty into estimates of spatial dependence. Check Github at `https://github.com/HopkinsIDD/IDSpatialStats` for latest development release.

Table 2: Descriptions of how previous studies have used the τ-statistic to quantify spatial dependence for infectious diseases. Listed in chronological order.

Description	Citation
Spatial and temporal dependence of homotypic and heterotypic Dengue virus serotypes over a 5 year period in Bangkok, Thailand	Salje et al. (2012)
Clustering of HIV prevalence and incidence around HIV-seropositive individuals using cohort data from rural Rakai District, Uganda	Grabowski et al. (2014)
Overview of the τ-statistic, its performance given observation error, and illustrations using Dengue, HIV, and Measles	Lessler et al. (2016)
Spatial dependence of seroconverted individuals in the 2012–2013 Chikungunya outbreak in the Phillipines	Salje et al. (2016a)
Comparison of spatial dependence in endemic transmission of Dengue virus serotypes in Bangkok and Ho Chi Min City, Thailand	Quoc et al. (2016)
Risk of Cholera transmission within spatial and temporal zones after case presentation during urban epidemics in Chad and D.R. Congo	Azman et al. (2018)
Summary statistic to fit micro-simulations of Cholera interventions to epidemic data using Approximate Bayesian Computation	Finger et al. (2018)
Temporal clustering of subclinical infections and homologous serotypes within schools using Dengue cohort data in Thailand	Salje et al. (2018)

Bibliography

A. S. Azman, F. J. Luquero, H. Salje, N. N. Mbaïbardoum, N. Adalbert, M. Ali, E. Bertuzzo, F. Finger, B. Toure, L. A. Massing, R. Ramazani, B. Saga, M. Allan, D. Olson, J. Leglise, K. Porten, and J. Lessler. Micro-hotspots of risk in urban cholera epidemics. 218(7):1164–1168, 2018. ISSN 0022-1899. doi: 10.1093/infdis/jiy283. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6107744/. [p324]

A. Baddeley, P. J. Diggle, A. Hardegen, T. Lawrence, R. K. Milne, and G. Nair. On tests of spatial pattern based on simulation envelopes. *Ecological Monographs*, 84(3):477–489, 2014. ISSN 0012-9615. doi: 10.1890/13-2042.1. URL http://doi.wiley.com/10.1890/13-2042.1. [p321]

A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. CRC Press, 2016. ISBN 978-1-4822-1021-7. [p308, 309, 316, 317]

R. Bivand and G. Piras. Comparing implementations of estimation methods for spatial econometrics. *Journal of Statistical Software*, 63(18):1–36, 2015. URL https://www.jstatsoft.org/v63/i18/. [p308, 309]

M. Corporation and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2018. URL https://CRAN.R-project.org/package=doParallel. R package version 1.0.14. [p309]

T. M. Davies, M. L. Hazelton, and J. C. Marshall. sparr: Analyzing spatial relative risk using fixed and adaptive kernel density estimation in r. 39(1), 2011. ISSN 1548-7660. doi: 10.18637/jss.v039.i01. URL http://www.jstatsoft.org/v39/i01/. [p309, 314, 317]

P. Diggle, P. Zheng, and P. Durr. Nonparametric estimation of spatial segregation in a multivariate point process: bovine tuberculosis in Cornwall, UK. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 54(3):645–658, 2005. ISSN 1467-9876. doi: 10.1111/j.1467-9876.2005.05373.x. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9876.2005.05373.x. [p317]

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. CRC Press, 1994. ISBN 9780412042317. [p320]

F. Finger, E. Bertuzzo, F. J. Luquero, N. Naibei, B. Touré, M. Allan, K. Porten, J. Lessler, A. Rinaldo, and A. S. Azman. The potential impact of case-area targeted interventions in response to cholera outbreaks: A modeling study. 15(2):e1002509, 2018. ISSN 1549-1676. doi: 10.1371/journal.pmed.1002509. URL https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.1002509. [p324]

R. C. Geary. The contiguity ratio and statistical mapping. 5(3):115–146, 1954. ISSN 1466-9404. doi: 10.2307/2986645. URL http://www.jstor.org/stable/2986645. [p308]

P. I. Good. *Permutation, Parametric, and Bootstrap Tests of Hypotheses*. Springer New York, 2010. ISBN 9781441919076. [p321]

M. K. Grabowski, J. Lessler, A. D. Redd, J. Kagaayi, O. Laeyendecker, A. Ndyanabo, M. I. Nelson, D. A. T. Cummings, J. B. Bwanika, A. C. Mueller, S. J. Reynolds, S. Munshaw, S. C. Ray, T. Lutalo, J. Manucci, A. A. R. Tobian, L. W. Chang, C. Beyrer, J. M. Jennings, F. Nalugoda, D. Serwadda, M. J. Wawer, T. C. Quinn, R. H. Gray, and t. R. H. S. Program. The role of viral introductions in sustaining community-based HIV epidemics in rural uganda: Evidence from spatial clustering, phylogenetics, and egocentric transmission models. 11(3):e1001610, 2014. ISSN 1549-1676. doi: 10.1371/journal.pmed.1001610. URL https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.1001610. [p324]

V. Gómez-Rubio, J. Ferrándiz-Ferragud, and A. Lopez-Quílez. Detecting clusters of disease with r. *Journal of Geographical Systems*, 7(2):189–206, 2005. [p309]

D. T. Haydon, M. Chase–Topping, D. J. Shaw, L. Matthews, J. K. Friar, J. Wilesmith, and M. E. J. Woolhouse. The construction and analysis of epidemic trees with reference to the 2001 UK foot–and–mouth outbreak. 270(1511):121–127, 2003. ISSN 0962-8452. doi: 10.1098/rspb.2002.2191. [p314]

J. Lessler, H. Salje, M. K. Grabowski, and D. A. T. Cummings. Measuring Spatial Dependence for Infectious Disease Epidemiology. *PLoS ONE*, 11(5):e0155249, 2016. ISSN 1932-6203. doi: 10.1371/journal.pone.0155249. [p309, 315, 324]

Microsoft and S. Weston. *foreach: Provides Foreach Looping Construct for R*, 2017. URL https://CRAN.R-project.org/package=foreach. R package version 1.4.4. [p309]

P. a. P. Moran. Notes on continuous stochastic phenomena. 37(1):17–23, 1950. ISSN 0006-3444. [p308]

T. Obadia, R. Haneef, and P.-Y. Boëlle. The r0 package: a toolbox to estimate reproduction numbers for epidemic outbreaks. 12:147, 2012. ISSN 1472-6947. doi: 10.1186/1472-6947-12-147. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3582628/. [p311]

S. Openshaw and P. Taylor. A million or so correlation coefficients: Three experiments on the modifiable areal unit problem. *Statistical Application in the Spatial Sciences*, 1272144, 1979. [p308]

R. Pélissier and F. Goreaud. ads package for R: A fast unbiased implementation of the *k*-function family for studying spatial point patterns in irregular-shaped sampling windows. *Journal of Statistical Software*, 63(6):1–18, 2015. URL http://www.jstatsoft.org/v63/i06/. [p308, 309]

C. H. Quoc, S. Henrik, R.-B. Isabel, Y. In-Kyu, N. V. V. Chau, N. T. Hung, H. M. Tuan, P. T. Lan, B. Willis, A. Nisalak, S. Kalayanarooj, D. A. T. Cummings, and C. P. Simmons. Synchrony of dengue incidence in ho chi minh city and bangkok. 10(12):e0005188, 2016. ISSN 1935-2735. doi: 10.1371/journal.pntd.0005188. URL http://journals.plos.org/plosntds/article?id=10.1371/journal.pntd.0005188. [p324]

T. Rajala. *SGCS: Spatial Graph Based Clustering Summaries for Spatial Point Patterns*, 2017. URL https://CRAN.R-project.org/package=SGCS. R package version 2.6. [p309]

I. W. Renner and D. I. Warton. Equivalence of maxent and poisson point process models for species distribution modeling in ecology. *Biometrics*, 69(6):274–281, 2013. [p308, 309]

B. D. Ripley. Tests of 'Randomness' for Spatial Point Patterns. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(3):368–374, 1979. ISSN 0035-9246. URL https://www.jstor.org/stable/2985065. [p321]

W. S. Robinson. Ecological Correlations and the Behavior of Individuals. *International Journal of Epidemiology*, 38(2):337–341, 2009. ISSN 0300-5771, 1464-3685. doi: 10.1093/ije/dyn357. URL http://ije.oxfordjournals.org/content/38/2/337. [p308]

B. Rowlingson and P. Diggle. *splancs: Spatial and Space-Time Point Pattern Analysis*, 2017. URL https://CRAN.R-project.org/package=splancs. R package version 2.01-40. [p308, 309]

H. Salje, J. Lessler, T. P. Endy, F. C. Curriero, R. V. Gibbons, A. Nisalak, S. Nimmannitya, S. Kalayanarooj, R. G. Jarman, S. J. Thomas, D. S. Burke, and D. A. T. Cummings. Revealing the microscale spatial signature of dengue transmission and immunity in an urban population. 109(24):9535–9538, 2012. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1120621109. URL http://www.pnas.org/content/109/24/9535. [p323, 324]

H. Salje, S. Cauchemez, M. T. Alera, I. Rodriguez-Barraquer, B. Thaisomboonsuk, A. Srikiatkhachorn, C. B. Lago, D. Villa, C. Klungthong, I. A. Tac-An, S. Fernandez, J. M. Velasco, V. G. Roque, A. Nisalak, L. R. Macareo, J. W. Levy, D. Cummings, and I.-K. Yoon. Reconstruction of 60 years of chikungunya epidemiology in the philippines demonstrates episodic and focal transmission. 213(4):604–610, 2016a. ISSN 0022-1899. doi: 10.1093/infdis/jiv470. URL https://academic.oup.com/jid/article/213/4/604/2459450/Reconstruction-of-60-Years-of-Chikungunya. [p324]

H. Salje, D. A. T. Cummings, and J. Lessler. Estimating infectious disease transmission distances using the overall distribution of cases. *Epidemics*, 17:10–18, 2016b. ISSN 1755-4365. doi: 10.1016/j.epidem.2016.10.001. [p309, 310, 312, 315, 322]

H. Salje, D. A. T. Cummings, I. Rodriguez-Barraquer, L. C. Katzelnick, J. Lessler, C. Klungthong, B. Thaisomboonsuk, A. Nisalak, A. Weg, D. Ellison, L. Macareo, I.-K. Yoon, R. Jarman, S. Thomas, A. L. Rothman, T. Endy, and S. Cauchemez. Reconstruction of antibody dynamics and infection histories to evaluate dengue risk. 557(7707):719–723, 2018. ISSN 1476-4687. doi: 10.1038/s41586-018-0157-4. URL https://www.nature.com/articles/s41586-018-0157-4. [p324]

B. M. Taylor, T. M. Davies, B. S. Rowlingson, and P. J. Diggle. lgcp: An R package for inference with spatial and spatio-temporal log-Gaussian Cox processes. *Journal of Statistical Software*, 52(4):1–40, 2013. URL http://www.jstatsoft.org/v52/i04/. [p308, 309]

B. M. Taylor, T. M. Davies, B. S. Rowlingson, and P. J. Diggle. Bayesian inference and data augmentation schemes for spatial, spatiotemporal and multivariate log-Gaussian Cox processes in R. *Journal of Statistical Software*, 63(7):1–48, 2015. URL http://www.jstatsoft.org/v63/i07/. [p308, 309]

J. Wallinga and P. Teunis. Different epidemic curves for severe acute respiratory syndrome reveal similar impacts of control measures. *American Journal of Epidemiology*, 160(6):509–516, 2004. doi: 10.1093/aje/kwh255. [p310, 311]

John R Giles
Department of Epidemiology
Johns Hopkins Bloomberg School of Public Health
615 N Wolfe St
Baltimore, MD, USA 21205
ORCiD: 0000-0002-0954-4093
giles@jhu.edu

Henrik Salje
Mathematical Modelling of Infectious Diseases Unit, Institut Pasteur
25-28 Rue du Dr Roux, Paris, France 75015
henrik.salje@pasteur.fr

Justin Lessler
Department of Epidemiology
Johns Hopkins Bloomberg School of Public Health
615 N Wolfe St
Baltimore, MD, USA 21205
justin@jhu.edu

Comparing namedCapture with other R packages for regular expressions

by Toby Dylan Hocking

Abstract Regular expressions are powerful tools for manipulating non-tabular textual data. For many tasks (visualization, machine learning, etc), tables of numbers must be extracted from such data before processing by other R functions. We present the R package **namedCapture**, which facilitates such tasks by providing a new user-friendly syntax for defining regular expressions in R code. We begin by describing the history of regular expressions and their usage in R. We then describe the new features of the namedCapture package, and provide detailed comparisons with related R packages (**rex**, **stringr**, **stringi**, **tidyr**, **rematch2**, **re2r**).

Introduction

Today regular expression libraries are powerful and widespread tools for text processing. A regular expression *pattern* is typically a character string that defines a set of possible *matches* in some other *subject* strings. For example the pattern `o+` matches one or more lower-case o characters; it would match the last two characters in the subject `foo`, and it would not match in the subject `bar`.

The focus of this article is regular expressions with capture groups, which are used to extract subject substrings. Capture groups are typically defined using parentheses. For example, the pattern `[0-9]+` matches one or more digits (e.g. `123` but not `abc`), and the pattern `[0-9]+-[0-9]+` matches a range of integers (e.g. `9-5`). The pattern `([0-9]+)-([0-9]+)` will perform matching identically, but provides access by number/index to the strings matched by the capturing sub-patterns enclosed in parentheses (group 1 matches `9`, group 2 matches `5`). The pattern `(?P<start>[0-9]+)-(?P<end>[0-9]+)` further provides access by name to the captured sub-strings (`start` group matches `9`, `end` group matches `5`). In R named capture groups are useful in order to create more readable regular expressions (names document the purpose of each sub-pattern), and to create more readable R code (it is easier to understand the intent of named references than numbered references).

In this article our original contribution is the R package **namedCapture** which provides several new features for named capture regular expressions. The main new ideas are (1) group-specific type conversion functions, (2) a user-friendly syntax for defining group names with R argument names, and (3) named output based on subject names and the `name` capture group.

The organization of this article is as follows. The rest of this introduction provides a brief history of regular expressions and their usage in R, then gives an overview of current R packages for regular expressions. The second section describes the proposed functions of the **namedCapture** package. The third section provides detailed comparisons with other R packages, in terms of syntax and computation times. The article concludes with a summary and discussion.

Origin of regular expressions and named capture groups

Regular expressions were first proposed in a theoretical paper by Kleene (1956). Among the first uses of a regular expression in computers was for searching in a text editor (Thompson, 1968) and lexical processing of source code (Johnson et al., 1968).

A capture group in a regular expression is used to extract text that matches a sub-pattern. In 1974, Thompson wrote the `grep` command line program, which was among the first to support capture groups (Friedl, 2002). In that program, backslash-escaped parentheses `\(\)` were used to open and close each capture group, which could then be referenced by number (`\1` for the first capture group, etc).

The idea for named capture groups seems to have originated in 1994 with the contributions of Tracy Tims to Python 1.0.0, which used the `\(<labelname>...\)` syntax (Python developers, 1997a). Python 1.5 introduced the `(?P<labelname>...)` syntax for name capture groups (Python developers, 1997b); the P was used to indicate that the syntax was a Python extension to the standard.

Perl-Compatible Regular Expressions (PCRE) is a C library that is now widely used in free/open-source software tools such as Python and R. PCRE introduced support for named capture in 2003, based on the Python syntax (Hazel, 2003). Starting in 2006, it supported the `(?<labelname>...)` syntax without a P, and the `(?'labelname'...)` syntax with single quotes, to be consistent with Perl and .NET (Hazel, 2003).

In the R NEWS files, the first mention of regular expression support was in 1997 with R-0.60,

C library	RE2	PCRE	ICU	TRE
Output group names	yes	yes	no	no
(?<group>pattern)	no	yes	yes	no
(?P<group>pattern)	yes	yes	no	no
Worst case linear time	yes	no	no	no
Backreferences	no	yes	yes	yes
Atomic groups / possessive quantifiers	no	yes	yes	no
Unicode properties	no	no	yes	no
Lookaround	no	yes	yes	no
Recursion	no	yes	no	no

Table 1: Features of C libraries for regular expressions usable in R.

"Regular expression matching is now done with system versions of the regexp library" (R Core Team, 1997). Starting with R-0.99.0, "R now compiles in the GNU version of regex" (R Core Team, 1997). PCRE was first included in R version 1.6.0 in 2002 (R Core Team, 2002). R-2.10 in 2009 was the first version to deprecate basic regular expressions, `extended=FALSE`, which are no longer supported (R Core Team, 2009). TRE is another C library for regular expressions that was included in R starting in R-2.10 (Laurikari, 2019). Although TRE supports capture groups, it does not allow capture groups to be named. The base R functions `regexpr` and `gregexpr` can be given the `perl=TRUE` argument in order to use the PCRE library, or `perl=FALSE` to use the TRE C library. Recently created packages (**stringi**, **re2r**) have provided R interfaces to the ICU and RE2 libraries.

Each library has different characteristics in terms of supported regex features and time complexity (Table 1). The most important feature for the purposes of this paper is "Output group names" which means the C library supports specifying capture group names in the regular expression pattern via `(?<group>pattern)` or `(?P<group>pattern)`, and then extracting those names for use in R (typically as column names in the resulting match matrix). "Worst case linear time" means that the match time is linear in the length of the input string, which is only guaranteed by the RE2 library. "Backreferences" can be used in patterns such as `(.)\1`, which means to match any character that appears twice in a row. "Atomic grouping" or "possessive quantifiers" means that only the greediest option of all possible alternatives will be considered; an example is the pattern `(?>.+)bar` which does not match the subject `foobar` (whereas the analog without the atomic group does). "Unicode properties" means support for regular expressions such as `\p{EMOJI_Presentation}`, which only work with ICU. "Lookaround" means support for zero-length assertions such as `foo(?=bar)` which matches `foo` only when it is followed by `bar` (but `bar` is not included in the match). "Recursion" is useful for matching balanced parentheses, and is only supported in PCRE; a simple recursive pattern is `a(?R)?z` which matches one or more `a` followed by exactly the same number of `z` (e.g. `aaazzz`).

The original versions of `regexpr` and `gregexpr` only returned the position/length of the text matched by an entire regex, not the capture groups (even though this is supported in TRE/PCRE). The C code that uses PCRE to extract each named capture group was accepted into R starting with version 2.14 (Hocking, 2011a). A lightning talk at useR 2011 showcased the new functionality (Hocking, 2011b).

Related R packages for capturing regular expressions

Since the introduction of named capture support in base R version 2.14, several packages have been developed which use this functionality, and other packages have been developed which use other C libraries (Table 2). Each package supports different options for subject/pattern input, extracted text outputs, named capture groups, and type conversion (Table 3). In this section we give an overview of the features of each package; in the section "Comparisons with other R packages" we show detailed comparisons including sample R code and outputs.

The **utils** package now includes the `strcapture` function, which uses the base `regexec` function (also introduced in R-2.14) to extract the first match as a data.frame with one row per subject, and one column per capture group. It allows capture group names/types to be specified in a prototype data.frame argument, but does not allow capture group names in the regex pattern. PCRE is used with `perl=TRUE` and TRE is used with `perl=FALSE`.

The **rematch2** package provides the `re_match` function which extracts the first match using the base `regexpr` function (Csárdi, 2017). It also provides `bind_re_match` for matching data.frame columns, and `re_match_all` which extracts all matches using the base `gregexpr` function. All functions output a data.frame with one row for each subject (for all matches a list column is used). PCRE is used with `perl=TRUE` and TRE is used with `perl=FALSE`. Although TRE supports capture groups (and can be

Package	First match	All matches	C library
namedCapture	`*_match_variable`	`str_match_all_variable`	PCRE/RE2
base	`regexpr`	`gregexpr`	PCRE/TRE
utils	`strcapture`	NA	PCRE/TRE
rematch2	`re_match`, `bind_re_match`	`re_match_all`	PCRE/TRE
rex	`re_matches(global=FALSE)`	`re_matches(global=TRUE)`	PCRE
stringr	`str_match`	`str_match_all`	ICU
stringi	`stri_match`	`stri_match_all`	ICU
tidyr	`extract`	NA	ICU
re2r	`re2_match`	`re2_match_all`	RE2

Table 2: R packages that provide functions for extracting first/all regex matches, and C library used.

used via the base R `regexec` function), capture groups are not supported in rematch2 with `perl=FALSE` (because it uses the base `regexpr/gregexpr` functions which do not return group info for TRE). Named capture groups are supported in rematch2 with `perl=TRUE`.

The **stringi** package provides the `stri_match` and `stri_match_all` functions, which have strong unicode support due to the underlying ICU C library (Gagolewski, 2018). The **stringr** package provides the `str_match` and `str_match_all` functions, which simply call the analogous functions from **stringi**. In ICU regular expressions, named groups are supported for use in backreferences since version 55 (ICU developers, 2015a,b). However, the ICU library does not report group names to R, so groups must be extracted by number in R. The `stri_match` function returns a character matrix with one row for each subject and one column for each capture group. The `stri_match_all` function returns a list with one element for each subject; each element is a data frame with one row for each match, and one column for each capture group.

The **re2r** package provides the `re2_match` and `re2_match_all` functions, which use the RE2 C++ library (Wenfeng, 2017). The outputs of these functions are consistent with the **stringi**/**stringr** packages. The input regex pattern may be specified as a character string or as a pre-compiled regex object (which results in faster matching if the regex is used with several calls to matching functions). Like TRE, the RE2 library guarantees linear time complexity, which is useful to avoid denial-of-service attacks from malicious patterns (see Section "Comparing computation times of R regex packages").

The **rex** package provides the `re_matches` function which supports named capture groups, and always uses PCRE (Ushey et al., 2017). By default it returns the first match (using the base `regexpr` function), as a data.frame with one row for each subject, and one column for each capture group. If the `global=TRUE` argument is given, `gregexpr` is used to return all matches as a list of data.frames. A unique feature of the **rex** package is a set of functions for defining a regular expression in R code, which is then converted to a standard PCRE regex pattern string (for a detailed comparison with the proposed syntax of the **namedCapture** package, see Section "Comparing **namedCapture** variable argument syntax with **rex**").

The **tidyr** package provides the `extract` function which uses the ICU library, so does not support regex patterns with named capture groups (Wickham and Henry, 2018). The subject is specified via the first two arguments: (1) a data.frame, and (2) a column name. The pattern is specified via the second two arguments: (3) a character vector for the capture group names, and (4) the regex pattern string (it is an error if the number of capture group names does not match the number of un-named capture groups in the regex pattern). The pattern is used to find the first match in each subject. The return value is a data.frame with the same number of rows as the input, but without the subject column, and with an additional column for each capture group.

The namedCapture package

The **namedCapture** package provides functions for extracting numeric data tables from non-tabular text data using named capture regular expressions. By default, **namedCapture** uses the RE2 C library if the **re2r** package is available, and PCRE otherwise (via the base `regexpr` and `gregexpr` functions). RE2 is preferred because it is guaranteed to find a match in linear time (see Section "Comparing computation times of R regex packages"). However, PCRE supports some regex features (e.g. backreferences) that RE2 does not. To tell **namedCapture** to use PCRE rather than RE2, `options(namedCapture.engine="PCRE")` can be specified. For patterns that are supported by both engines, **namedCapture** functions return the resulting match in the standard output format described below.

The main design features of the namedCapture package are inspired by the base R system, which

Package/function	subject	pattern	outputs	named	types
namedCapture	chr/df	chr/verbose	mat/list/df	yes	any
base	chr	chr	mat/list	yes	no
utils::strcapture	chr	chr	df	no	some
rematch2	chr/df	chr	df	yes	no
rex	chr	chr/verbose	df/list	yes	no
stringr	chr	chr	mat/list	no	no
stringi	chr	chr	mat/list	no	no
tidyr::extract	df	chr	df	no	some
re2r	chr	chr/compiled	df/list	yes	no

Table 3: R packages provide different options for subject/pattern input, extracted text outputs, named capture groups output to R, and type conversion. Abbreviations: chr=character vector, df=data.frame, mat=character matrix, verbose=regex defined in R code which is translated to a character string, compiled=regex string may be compiled and saved to an R object for later use.

First match	All matches	Arguments
str_match_named	str_match_all_named	chr subject, chr pattern, functions
str_match_variable	str_match_all_variable	chr subject, chr/list/function, ...
df_match_variable	NA	df subject, chr/list/function, ...

Table 4: Functions of the namedCapture package. The first argument of each function specifies the subject, as either a character vector (for str_*) functions, or a data.frame (for df_match_variable). The *_named functions require three arguments, and are mostly for internal use; the *_variable functions take a variable number of arguments and are the recommended functions to use.

provides good support for naming objects, and referring to objects by name. In particular, the namedCapture package supports

- Standard syntax for specifying capture groups with names in a regular expression string, and stopping with an informative error if there are un-named capture groups.
- A new/alternative syntax for specifying capture group names via named arguments in R code.
- Output with rownames or list names taken from subject names.
- Output with rownames taken from the name capture group.
- Specifying a type conversion function for each named capture group.
- Saving sub-patterns to R variables, and re-using them multiple times in one or several patterns in order to avoid repetition.

The main functions provided by the namedCapture package are summarized in Table 4. We begin by introducing the *_named functions, which take three arguments.

Three argument syntax: str_match_named and str_match_all_named

The most basic functions of the namedCapture package are str_match_named and str_match_all_named, which accept exactly three arguments:

- subject: a character vector from which we want to extract tabular data.
- pattern: the (character scalar) regular expression with named capture groups used for extraction.
- fun.list: a list with names that correspond to capture groups, and values are functions used to convert the extracted character data to other (typically numeric) types.

Since introduction of the variable argument syntax (explained later in this section), these functions are mostly for internal use. Here we give an example of their usage, because it is similar to other R regex packages which some readers are probably already familiar with. Consider subjects containing genomic positions:

```
> chr.pos.subject <- c("chr10:213,054,000-213,055,000", "chrM:111,000",
+   "this will not match", NA, "chr1:110-111 chr2:220-222")
```

These subjects consist of a chromosome name string, a colon, a start position, and optionally a dash and and end position. The following pattern is used to extract those data:

```
> chr.pos.pattern <- paste0(
+   "(?P<chrom>chr.*?)",
+   ":",
+   "(?P<chromStart>[0-9,]+)",
+   "(?:",
+     "-",
+     "(?P<chromEnd>[0-9,]+)",
+   ")?")
```

The pattern above is defined using `paste0`, writing each named capture group on a separate line, which increases readability of the pattern. Note that an optional non-capturing group begins with `(?:` and ends with `)?`. In the code below, we use the `str_match_named` function on the previously defined subject and pattern:

```
> (match.mat <- namedCapture::str_match_named(
+   chr.pos.subject, chr.pos.pattern))

     chrom   chromStart    chromEnd
[1,] "chr10" "213,054,000" "213,055,000"
[2,] "chrM"  "111,000"     ""
[3,] NA      NA            NA
[4,] NA      NA            NA
[5,] "chr1"  "110"         "111"
```

When the third argument is omitted, the return value a character matrix with one row for each subject and one column for each capture group. Column names are taken from the group names that were specified in the regular expression pattern. Missing values indicate missing subjects or no match. The empty string is used for optional groups which are not used in the match (e.g. chromEnd group/column for second subject). This output format is similar to the output of `stringi::stri_match` and `stringr::str_match`; these other functions also report a column for the entire match, whereas `namedCapture::str_match_named` only reports a column for each named capture group.

However we often want to extract numeric data; in this case we want to convert chromStart/End to integers. You can do that by supplying a named list of conversion functions as the third argument. Each function should take exactly one argument, a character vector (data in the matched column/group), and return a vector of the same size. The code below specifies the `int.from.digits` function for both chromStart and chromEnd:

```
> int.from.digits <- function(captured.text)as.integer(gsub("[^0-9]", "", captured.text))
> conversion.list <- list(chromStart=int.from.digits, chromEnd=int.from.digits)
> match.df <- namedCapture::str_match_named(
+   chr.pos.subject, chr.pos.pattern, conversion.list)
> str(match.df)

'data.frame':        5 obs. of  3 variables:
 $ chrom     : chr  "chr10" "chrM" NA NA ...
 $ chromStart: int  213054000 111000 NA NA 110
 $ chromEnd  : int  213055000 NA NA NA 111
```

Note that a data.frame is returned when the third argument is specified, in order to handle non-character data types returned by the conversion functions.

In the examples above the last subject has two possible matches, but only the first is returned by `str_match_named`. Use `str_match_all_named` to get all matches in each subject (not just the first match).

```
> namedCapture::str_match_all_named(
+   chr.pos.subject, chr.pos.pattern, conversion.list)

[[1]]
  chrom chromStart  chromEnd
1 chr10  213054000 213055000

[[2]]
  chrom chromStart chromEnd
1  chrM     111000       NA

[[3]]
```

```
data frame with 0 columns and 0 rows

[[4]]
data frame with 0 columns and 0 rows

[[5]]
  chrom chromStart chromEnd
1  chr1        110      111
2  chr2        220      222
```

As shown above, the result is a list with one element for each subject. Each list element is a data.frame with one row for each match.

Named output for named subjects

If the subject is named, its names will be used to name the output (rownames or list names).

```
> named.subject <- c(ten="chr10:213,054,000-213,055,000",
+   M="chrM:111,000", two="chr1:110-111 chr2:220-222")
> namedCapture::str_match_named(
+   named.subject, chr.pos.pattern, conversion.list)

    chrom chromStart  chromEnd
ten chr10  213054000 213055000
M    chrM     111000        NA
two  chr1        110       111

> namedCapture::str_match_all_named(
+   named.subject, chr.pos.pattern, conversion.list)

$ten
  chrom chromStart  chromEnd
1 chr10  213054000 213055000

$M
  chrom chromStart chromEnd
1  chrM     111000       NA

$two
  chrom chromStart chromEnd
1  chr1        110      111
2  chr2        220      222
```

This feature makes it easy to select particular subjects/matches by name.

The `name` group specifies row names of output

If the pattern specifies the `name` group, then it will be used for the rownames of the output, and it will not be included as a column. However if the subject has names, and the `name` group is specified, then to avoid losing information the subject names are used to name the output (and the `name` column is included in the output).

```
> name.pattern <- paste0(
+   "(?P<name>chr.*?)",
+   ":",
+   "(?P<chromStart>[0-9,]+)",
+   "(?:",
+     "-",
+     "(?P<chromEnd>[0-9,]+)",
+   ")?")
> namedCapture::str_match_named(
+   named.subject, name.pattern, conversion.list)

     name chromStart  chromEnd
ten chr10  213054000 213055000
```

```
M    chrM     111000        NA
two  chr1        110       111

> namedCapture::str_match_all_named(
+   named.subject, name.pattern, conversion.list)

$ten
      chromStart  chromEnd
chr10  213054000 213055000

$M
     chromStart chromEnd
chrM     111000       NA

$two
     chromStart chromEnd
chr1        110      111
chr2        220      222
```

Readable and efficient variable argument syntax used in `str_match_variable`

In this section we introduce the variable argument syntax used in the `*_variable` functions, which is the recommended way to to use **namedCapture**. This new syntax is both readable and efficient, because it is motivated by the desire to avoid repetitive/boilerplate code. In the previous sections we defined the pattern using the `paste0` boilerplate, which is used to break the pattern over several lines for clarity. We begin by introducing `str_match_variable`, which extracts the first match from each subject. Using the variable argument syntax, we can omit `paste0`, and simply supply the pattern strings to `str_match_variable` directly,

```
> namedCapture::str_match_variable(
+   named.subject,
+   "(?P<chrom>chr.*?)",
+   ":",
+   "(?P<chromStart>[0-9,]+)",
+   "(?:",
+     "-",
+     "(?P<chromEnd>[0-9,]+)",
+   ")?")

    chrom   chromStart    chromEnd
ten "chr10" "213,054,000" "213,055,000"
M   "chrM"  "111,000"     ""
two "chr1"  "110"         "111"
```

The variable argument syntax allows further simplification by removing the named capture groups from the strings, and adding names to the corresponding arguments. For `name1="pattern1"`, namedCapture internally generates/uses the regex `(?P<name1>pattern1)`.

```
> namedCapture::str_match_variable(
+   named.subject,
+   chrom="chr.*?",
+   ":",
+   chromStart="[0-9,]+",
+   "(?:",
+     "-",
+     chromEnd="[0-9,]+",
+   ")?")

    chrom   chromStart    chromEnd
ten "chr10" "213,054,000" "213,055,000"
M   "chrM"  "111,000"     ""
two "chr1"  "110"         "111"
```

We can also provide a type conversion function on the same line as a named group:

```
> namedCapture::str_match_variable(
+   named.subject,
+   chrom="chr.*?",
+   ":",
+   chromStart="[0-9,]+", int.from.digits,
+   "(?:",
+     "-",
+     chromEnd="[0-9,]+", int.from.digits,
+   ")?")

    chrom chromStart  chromEnd
ten chr10  213054000 213055000
M    chrM     111000        NA
two  chr1        110       111
```

Note the repetition in the chromStart/End lines — the same pattern and type conversion function is used for each group. This repetition can be avoided by creating and using a sub-pattern list variable,

```
> int.pattern <- list("[0-9,]+", int.from.digits)
> namedCapture::str_match_variable(
+   named.subject,
+   chrom="chr.*?",
+   ":",
+   chromStart=int.pattern,
+   "(?:",
+     "-",
+     chromEnd=int.pattern,
+   ")?")

    chrom chromStart  chromEnd
ten chr10  213054000 213055000
M    chrM     111000        NA
two  chr1        110       111
```

Finally, the non-capturing group can be replaced by an un-named list:

```
> namedCapture::str_match_variable(
+   named.subject,
+   chrom="chr.*?",
+   ":",
+   chromStart=int.pattern,
+   list(
+     "-",
+     chromEnd=int.pattern
+   ), "?")

    chrom chromStart  chromEnd
ten chr10  213054000 213055000
M    chrM     111000        NA
two  chr1        110       111
```

In summary, the `str_match_variable` function takes a variable number of arguments, and allows for a shorter, less repetitive, and thus more user-friendly syntax:

- The first argument is the subject character vector.
- The other arguments specify the pattern, via character strings, functions, and/or lists.
- If a pattern (character/list) is named, we use the argument name in R for the capture group name in the regex.
- Each function is used to convert the text extracted by the previous named pattern argument. (type conversion can only be used with named R arguments, NOT with explicitly specified named groups in regex strings)
- R sub-pattern variables may be used to avoid repetition in the definition of the pattern and type conversion functions.
- Each list generates a group in the regex (named list = named capture group, un-named list = non-capturing group).
- All patterns are pasted together in the order that they appear in the argument list.

Extract all matches from a multi-line text file via `str_match_all_variable`

The variable argument syntax can also be used with `str_match_all_variable`, which is for the common case of extracting each match from a multi-line text file. In this section we demonstrate how to use `str_match_all_variable` to extract data.frames from a non-tabular text file.

```
> trackDb.txt.gz <- system.file(
+   "extdata", "trackDb.txt.gz", package="namedCapture")
> trackDb.lines <- readLines(trackDb.txt.gz)
```

Some representative lines from that file are shown below.

```
> show.width <- 55
> substr(trackDb.lines[78:107], 1, show.width)
```

```
 [1] "track peaks_summary"
 [2] "type bigBed 5"
 [3] "shortLabel _model_peaks_summary"
 [4] "longLabel Regions with a peak in at least one sample"
 [5] "visibility pack"
 [6] "itemRgb off"
 [7] "spectrum on"
 [8] "bigDataUrl http://hubs.hpc.mcgill.ca/~thocking/PeakSegF"
 [9] ""
[10] ""
[11] " track bcell_McGill0091"
[12] " parent bcell"
[13] " container multiWig"
[14] " type bigWig"
[15] " shortLabel bcell_McGill0091"
[16] " longLabel bcell | McGill0091"
[17] " graphType points"
[18] " aggregate transparentOverlay"
[19] " showSubtrackColorOnUi on"
[20] " maxHeightPixels 25:12:8"
[21] " visibility full"
[22] " autoScale on"
[23] ""
[24] "  track bcell_McGill0091Coverage"
[25] "  bigDataUrl http://hubs.hpc.mcgill.ca/~thocking/PeakSe"
[26] "  shortLabel bcell_McGill0091Coverage"
[27] "  longLabel bcell | McGill0091 | Coverage"
[28] "  parent bcell_McGill0091"
[29] "  type bigWig"
[30] "  color 141,211,199"
```

Each block of text begins with `track` and includes several lines of data before the block ends with two consecutive newlines. That pattern is coded below:

```
> fields.mat <- namedCapture::str_match_all_variable(
+   trackDb.lines,
+   "track ",
+   name="\\S+",
+   fields="(?:\n[^\n]+)*",
+   "\n")
> head(substr(fields.mat, 1, show.width))
```

```
                      fields
bcell                 "\nsuperTrack on show\nshortLabel bcell\nlongLabel bcell Ch"
kidneyCancer          "\nsuperTrack on show\nshortLabel kidneyCancer\nlongLabel k"
kidney                "\nsuperTrack on show\nshortLabel kidney\nlongLabel kidney "
leukemiaCD19CD10BCells "\nsuperTrack on show\nshortLabel leukemiaCD19CD10BCells\nl"
monocyte              "\nsuperTrack on show\nshortLabel monocyte\nlongLabel monoc"
skeletalMuscleCtrl    "\nsuperTrack on show\nshortLabel skeletalMuscleCtrl\nlongL"
```

Note that this function assumes that its subject is a character vector with one element for each line in a file. The elements are pasted together using newline as a separator, and the regex is used to find

all matches in the resulting multi-line string. The code above creates a data frame with one row for each track block, with rownames given by the track line (because of the `name` capture group), and one `fields` column which is a string with the rest of the data in that block.

Each block has a variable number of lines/fields. Each line starts with a field name, followed by a space, followed by the field value. That regex is coded below:

```
> fields.list <- namedCapture::str_match_all_named(
+   fields.mat[, "fields"], paste0(
+     "\\s+",
+     "(?P<name>.*?)",
+     " ",
+     "(?P<value>[^\n]+)"))
> substr(fields.list$bcell_McGill0091Coverage, 1, show.width)

           value
bigDataUrl "http://hubs.hpc.mcgill.ca/~thocking/PeakSegFPOP-/sample"
shortLabel "bcell_McGill0091Coverage"
longLabel  "bcell | McGill0091 | Coverage"
parent     "bcell_McGill0091"
type       "bigWig"
color      "141,211,199"
```

The result is a list of data frames. There is a list element for each block, named by track. Each list element is a data frame with one row per field defined in that block (rownames are field names). The names/rownames make it easy to write R code that selects individual elements by name.

In the example above we extracted all fields from all tracks (using two regexes, one for the track, one for the field). In the example below we use a single regex to extract the name of each track, and split components into separate columns. It also demonstrates how to use nested named capture groups, via a named list which contains other named patterns.

```
> match.df <- namedCapture::str_match_all_variable(
+   trackDb.lines,
+   "track ",
+   name=list(
+     cellType=".*?",
+     "_",
+     sampleName=list(
+       "McGill",
+       sampleID=int.pattern),
+     dataType="Coverage|Peaks",
+     "|",
+     "[^\n]+"))
> match.df["bcell_McGill0091Coverage", ]

                        cellType sampleName sampleID dataType
bcell_McGill0091Coverage    bcell McGill0091       91 Coverage
```

Exercise for the reader: modify the above in order to capture the bigDataUrl field, and three additional columns (red, green, blue) from the color field.

`df_match_variable` extracts new columns from character columns in a data.frame

We also provide `namedCapture::df_match_variable` which extracts text from several columns of a data.frame, using a different named capture regular expression for each column.

- It requires a data.frame as the first argument.
- It takes a variable number of other arguments, all of which must be named. For each other argument we call `str_match_variable` on one column of the input data.frame.
- Each argument name specifies a column of the data.frame which will be used as the subject in `str_match_variable`.
- Each argument value specifies a pattern, in list/character/function variable argument syntax.
- The return value is a data.frame with the same number of rows as the input, but with an additional column for each named capture group. New columns are named using the convention `subjectColumnName.groupName`.

This function can greatly simplify the code required to create numeric data columns from character data columns. For example consider the following data which was output from the SLURM sacct command line program.

```
> (sacct.df <- data.frame(
+   Elapsed=c("07:04:42", "07:04:42", "07:04:49", "00:00:00", "00:00:00"),
+   JobID=c("13937810_25", "13937810_25.batch", "13937810_25.extern",
+     "14022192_[1-3]", "14022204_[4]"), stringsAsFactors=FALSE))

   Elapsed              JobID
1 07:04:42        13937810_25
2 07:04:42  13937810_25.batch
3 07:04:49 13937810_25.extern
4 00:00:00     14022192_[1-3]
5 00:00:00       14022204_[4]
```

Say we want to filter by the total Elapsed time (which is reported as hours:minutes:seconds), and base job id (which is the number before the underscore in the JobID column). We begin by defining a pattern that matches a range of integer task IDs in square brackets, and applying that pattern to the JobID column:

```
> range.pattern <- list(
+   "[[]",
+   task1=int.pattern,
+   list(
+     "-",
+     taskN=int.pattern
+   ), "?",
+   "[]]")
> namedCapture::df_match_variable(sacct.df, JobID=range.pattern)

   Elapsed              JobID JobID.task1 JobID.taskN
1 07:04:42        13937810_25          NA          NA
2 07:04:42  13937810_25.batch          NA          NA
3 07:04:49 13937810_25.extern          NA          NA
4 00:00:00     14022192_[1-3]           1           3
5 00:00:00       14022204_[4]           4          NA
```

The result shown above is another data frame with an additional column for each named capture group. Next, we define another pattern that matches either one task ID or the previously defined range pattern:

```
> task.pattern <- list(
+   "_", list(
+     task=int.pattern,
+     "|",#either one task(above) or range(below)
+     range.pattern))
> namedCapture::df_match_variable(sacct.df, JobID=task.pattern)

   Elapsed              JobID JobID.task JobID.task1 JobID.taskN
1 07:04:42        13937810_25         25          NA          NA
2 07:04:42  13937810_25.batch         25          NA          NA
3 07:04:49 13937810_25.extern         25          NA          NA
4 00:00:00     14022192_[1-3]         NA           1           3
5 00:00:00       14022204_[4]         NA           4          NA
```

Below, we use the previously defined patterns to match the complete JobID column, along with the Elapsed column:

```
> future::plan("multiprocess")
> namedCapture::df_match_variable(
+   sacct.df,
+   JobID=list(
+     job=int.pattern,
+     task.pattern,
+     list(
```

```
+       "[.]",
+       type=".*"
+     ), "?"),
+   Elapsed=list(
+     hours=int.pattern,
+     ":",
+     minutes=int.pattern,
+     ":",
+     seconds=int.pattern))

   Elapsed              JobID JobID.job JobID.task JobID.task1 JobID.taskN
1 07:04:42        13937810_25  13937810         25          NA          NA
2 07:04:42  13937810_25.batch  13937810         25          NA          NA
3 07:04:49 13937810_25.extern  13937810         25          NA          NA
4 00:00:00     14022192_[1-3]  14022192         NA           1           3
5 00:00:00       14022204_[4]  14022204         NA           4          NA
  JobID.type Elapsed.hours Elapsed.minutes Elapsed.seconds
1                        7               4              42
2      batch             7               4              42
3     extern             7               4              49
4                        0               0               0
5                        0               0               0
```

The code above specifies two named arguments to `df_match_variable`. Each named argument specifies a column from which tabular data are extracted using the corresponding pattern. The final result is a data frame with an additional column for each named capture group.

Comparisons with other R packages

In this section we compare the proposed functions in the **namedCapture** package with similar functions in other R packages for regular expressions.

Comparing namedCapture variable argument syntax with rex

In this section we compare **namedCapture** verbose variable argument syntax with the similar **rex** package. We have adapted the log parsing example from the **rex** package:

```
> log.subject <- 'gate3.fmr.com - - [05/Jul/1995:13:51:39 -0400] "GET /shuttle/
+ curly02.slip.yorku.ca - - [10/Jul/1995:23:11:49 -0400] "GET /sts-70/sts-small.gif
+ boson.epita.fr - - [15/Jul/1995:11:27:49 -0400] "GET /movies/sts-71-mir-dock.MPG
+ 134.153.50.9 - - [13/Jul/1995:11:02:50 -0400] "GET /icons/text.xbm'
> log.lines <- strsplit(log.subject, split="\n")[[1]]
```

The goal is to extract the time and filetype for each log line. The code below uses the `rex` function to define a pattern for matching the filetype:

```
> library(rex)
> library(dplyr)
> (rex.filetype.pattern <- rex(
+   non_spaces, ".",
+   capture(name = 'filetype',
+           none_of(space, ".", "?", double_quote) %>% one_or_more())))

[^[:space:]]+\.(?<filetype>(?:[^[:space:].?"])+)
```

Note that rex defines R functions (e.g. `capture`, `one_or_more`) and constants (`non_spaces`, `double_quote`) which are translated to standard regular expression syntax via the `rex` function. These regex objects can be used as sub-patterns in other calls to `rex`, as in the code below:

```
> rex.pattern <- rex(
+   "[",
+   capture(name = "time", none_of("]") %>% zero_or_more()),
+   "]",
+   space, double_quote, "GET", space,
+   maybe(rex.filetype.pattern))
```

Finally, the pattern is used with `re_matches` in order to extract a data table, and the `mutate` function is used for type conversion:

```
> re_matches(log.lines, rex.pattern) %>% mutate(
+   filetype = tolower(filetype),
+   time = as.POSIXct(time, format="%d/%b/%Y:%H:%M:%S %z"))

                time filetype
1 1995-07-05 10:51:39
2 1995-07-10 20:11:49      gif
3 1995-07-15 08:27:49      mpg
4 1995-07-13 08:02:50      xbm
```

Using the **namedCapture** package we begin by defining an analogous filetype pattern as a list containing literal regex strings and a type conversion function:

```
> namedCapture.filetype.pattern <- list(
+   "[^[:space:]]+[.]",
+   filetype='[^[:space:].?"]+', tolower)
```

We can then use that as a sub-pattern in a call to `str_match_variable`, which results in a data table with columns generated via the specified type conversion functions:

```
> namedCapture::str_match_variable(
+   log.lines,
+   "\\[",
+   time="[^]]*", function(x)as.POSIXct(x, format="%d/%b/%Y:%H:%M:%S %z"),
+   "\\]",
+   ' "GET ',
+   namedCapture.filetype.pattern, "?")

                time filetype
1 1995-07-05 10:51:39
2 1995-07-10 20:11:49      gif
3 1995-07-15 08:27:49      mpg
4 1995-07-13 08:02:50      xbm
```

Overall both **rex** and **namedCapture** provide good support for defining regular expresions using a verbose, readable, and thus user-friendly syntax. However there are two major differences:

- **namedCapture** assumes the user knows regular expressions and can write them as R string literals; **rex** assumes the user knows its functions, which generate regex strings. For example the capture group time, `none_of("]") %>% zero_or_more()` in rex gets translated to the regex string `[^]]*`. Thus **rex** code is a bit more verbose than **namedCapture**.
- In **namedCapture** type conversion functions can be specified on the same line as the capture group name/pattern, whereas in **rex** type conversions are specified as a post-processing step on the result of `re_matches`.

Comparing `namedCapture::df_match_variable` with other functions for data.frames

The **tidyr** and **rematch2** packages provide functionality similar to `namedCapture::df_match_variable`, which was introduced in Section "`df_match_variable` extracts new columns from character columns in a data.frame." Below we show how `tidyr::extract` can be used to compute a similar result as in that previous section, using the same data from the SLURM sacct command line program. We begin by defining a pattern which matches a range of integers in square brackets:

```
> tidyr.range.pattern <- "\\[([0-9]+)(?:-([0-9]+))?\\]"
> tidyr::extract(
+   sacct.df, "JobID", c("task1", "taskN"),
+   tidyr.range.pattern, remove=FALSE)

   Elapsed              JobID task1 taskN
1 07:04:42        13937810_25  <NA>  <NA>
2 07:04:42  13937810_25.batch  <NA>  <NA>
3 07:04:49 13937810_25.extern  <NA>  <NA>
4 00:00:00     14022192_[1-3]     1     3
5 00:00:00       14022204_[4]     4  <NA>
```

Note the pattern string includes un-named capture groups, because named capture is not supported. Names must therefore be specified in the third argument of `extract`. Next, we define a pattern which matches either a single task ID, or a range in square brackets:

```
> tidyr.task.pattern <- paste0("_(?:([0-9]+)|", tidyr.range.pattern, ")")
> tidyr::extract(sacct.df, "JobID", c("task", "task1", "taskN"),
+   tidyr.task.pattern, remove=FALSE)

   Elapsed              JobID task task1 taskN
1 07:04:42        13937810_25   25  <NA>  <NA>
2 07:04:42  13937810_25.batch   25  <NA>  <NA>
3 07:04:49 13937810_25.extern   25  <NA>  <NA>
4 00:00:00     14022192_[1-3] <NA>     1     3
5 00:00:00       14022204_[4] <NA>     4  <NA>
```

In the code below we define a pattern that matches the entire job string:

```
> tidyr.job.pattern <- paste0("([0-9]+)", tidyr.task.pattern, "(?:[.](.*))?")
> (job.df <- tidyr::extract(sacct.df, "JobID",
+   c("job", "task", "task1", "taskN", "type"), tidyr.job.pattern))

   Elapsed      job task task1 taskN   type
1 07:04:42 13937810   25  <NA>  <NA>   <NA>
2 07:04:42 13937810   25  <NA>  <NA>  batch
3 07:04:49 13937810   25  <NA>  <NA> extern
4 00:00:00 14022192 <NA>     1     3   <NA>
5 00:00:00 14022204 <NA>     4  <NA>   <NA>
```

Finally, we use another pattern to extract the components of the elapsed time. Note that `convert=TRUE` means to use `utils::type.convert` on the result of each extracted group.

```
> tidyr::extract(job.df, "Elapsed", c("hours", "minutes", "seconds"),
+   "([0-9]+):([0-9]+):([0-9]+)", convert=TRUE)

  hours minutes seconds      job task task1 taskN   type
1     7       4      42 13937810   25  <NA>  <NA>   <NA>
2     7       4      42 13937810   25  <NA>  <NA>  batch
3     7       4      49 13937810   25  <NA>  <NA> extern
4     0       0       0 14022192 <NA>     1     3   <NA>
5     0       0       0 14022204 <NA>     4  <NA>   <NA>
```

Below we show the same computation using `rematch2::bind_re_match`, which supports named capture. Note that we use `paste0` to define a regular expression with each named capture group on a separate line:

```
> rematch2.range.pattern <- paste0(
+   "\\[",
+   "(?P<task1>[0-9]+)",
+   "(?:-",
+   "(?P<taskN>[0-9]+)",
+   ")?\\]")
> rematch2::bind_re_match(sacct.df, JobID, rematch2.range.pattern)

   Elapsed              JobID task1 taskN
1 07:04:42        13937810_25  <NA>  <NA>
2 07:04:42  13937810_25.batch  <NA>  <NA>
3 07:04:49 13937810_25.extern  <NA>  <NA>
4 00:00:00     14022192_[1-3]     1     3
5 00:00:00       14022204_[4]     4
```

Above we extract a range of task IDs in square brackets, and below we optionally match a single task ID:

```
> rematch2.task.pattern <- paste0(
+   "_(?:",
+   "(?P<task>[0-9]+)",
+   "|", rematch2.range.pattern, ")")
> rematch2::bind_re_match(sacct.df, JobID, rematch2.task.pattern)
```

```
   Elapsed              JobID task task1 taskN
1 07:04:42        13937810_25   25
2 07:04:42  13937810_25.batch   25
3 07:04:49 13937810_25.extern   25
4 00:00:00     14022192_[1-3]          1     3
5 00:00:00       14022204_[4]          4
```

Below we additionally match the job ID and job type:

```
> rematch2.job.pattern <- paste0(
+   "(?P<job>[0-9]+)",
+   rematch2.task.pattern,
+   "(?:[.]",
+   "(?P<type>.*)",
+   ")?")
> (rematch2.job.df <- rematch2::bind_re_match(
+   sacct.df, JobID, rematch2.job.pattern))
```

```
   Elapsed              JobID      job task task1 taskN   type
1 07:04:42        13937810_25 13937810   25
2 07:04:42  13937810_25.batch 13937810   25              batch
3 07:04:49 13937810_25.extern 13937810   25             extern
4 00:00:00     14022192_[1-3] 14022192          1     3
5 00:00:00       14022204_[4] 14022204          4
```

Finally we call the function on the result from above with a new pattern for another column:

```
> transform(rematch2::bind_re_match(
+   rematch2.job.df, Elapsed,
+   "(?P<hours>[0-9]+):(?P<minutes>[0-9]+):(?P<seconds>[0-9]+)"),
+   hours.int=as.integer(hours),
+   minutes.int=as.integer(minutes),
+   seconds.int=as.integer(seconds))
```

```
   Elapsed              JobID      job task task1 taskN   type hours minutes
1 07:04:42        13937810_25 13937810   25                        07      04
2 07:04:42  13937810_25.batch 13937810   25              batch     07      04
3 07:04:49 13937810_25.extern 13937810   25             extern     07      04
4 00:00:00     14022192_[1-3] 14022192          1     3            00      00
5 00:00:00       14022204_[4] 14022204          4                  00      00
  seconds hours.int minutes.int seconds.int
1      42         7           4          42
2      42         7           4          42
3      49         7           4          49
4      00         0           0           0
5      00         0           0           0
```

Overall our comparison demonstrates that `tidyr::extract` and `rematch2::bind_re_match` function similarly to `namedCapture::df_match_variable`, with the following differences:

- Because `tidyr::extract` uses the ICU C library, which does not support named capture regular expressions, it requires specifying the group names in a separate argument. In contrast, **rematch2** supports specifying capture group names in regex string literals; **namedCapture** variable argument syntax supports specifying capture group names as R argument names on the same line as the corresponding sub-pattern.
- Since `rematch2::bind_re_match` returns character columns, conversion to numeric types must be accomplished in a post-processing step using a function such as `transform`. In contrast `tidyr::extract(convert=TRUE)` always uses `utils::type.convert` for type conversion, and `namedCapture::df_match_variable` supports arbitrary group-specific type conversion functions, which are specified on the same line as the corresponding name/pattern.
- Because `tidyr::extract` and `rematch2::bind_re_match` operate on one column in the subject data frame, they must be called twice (once for the Elapsed column, once for the JobID column). In contrast, one call to `namedCapture::df_match_variable` can be used to extract data from multiple columns in the subject data frame.

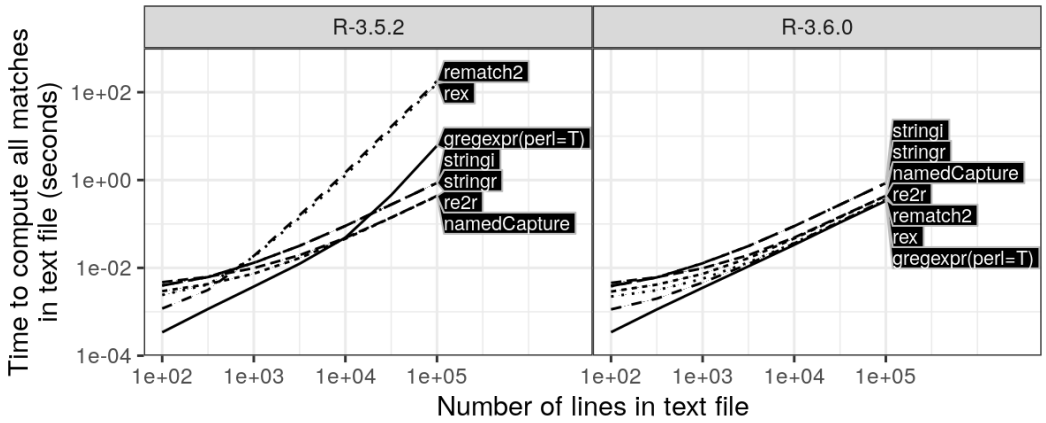


Figure 1: Computation time for finding all matches in a text file is plotted as a function of number of lines (median lines and quartile bands over 5 timings). Such timings are typical of real-world subjects/patterns. R-3.5.2 used quadratic time algorithms for `gregexpr/substring` (left), which were changed to linear time algorithms in R-3.6.0 due to this research (right).

Comparing computation times of R regex packages

In this section we compare the computation time of the proposed **namedCapture** package with other R packages. For all of the comparisons, we used the **microbenchmark** package to compute the computation times of each R package/function. We study how the empirical computation time scales as a function of subject/pattern size. The first three comparisons come from the real-world examples discussed earlier in this article; the last two comparisons are pathological examples used to show worst case time complexity.

The first example involves extracting all matches from a multi-line text file, as discussed in Section "Extract all matches from a multi-line text file via `str_match_all_variable`." Figure 1 shows comparisons with packages **re2r**, **stringr**, **stringi**, **rematch2**, **rex**. We expected small differences between the packages, on the order of constant factors. Using R-3.5.2 (left panel of Figure 1), the lines for the **rex** and **rematch2** packages have significantly larger slopes than the other packages (**namedCapture**, **stringr**, **stringi**, and **re2r**). This can be explained because **rex** and **rematch2** use the base `gregexpr` and `substring` functions, which are implemented using inefficient quadratic time algorithms in R-3.5.2. As a result of this research, this issue was reported on the R-devel email list, and R-core member Tomas Kalibera has fixed the problem. In R-3.6.0 (right panel of Figure 1), linear time algorithms are used.

The second example involves extracting the first match from each line of a log file, as discussed in Section "Comparing **namedCapture** variable argument syntax with **rex**." Figure 2 (left) shows comparisons with the previously discussed packages and `utils:strcapture`. We expected small differences between the packages, on the order of constant factors. In this comparison we observed only small constant factor differences, and linear time complexity for all packages.

The third example involves using a different regular expression to extract data for each of two columns of a data frame, as discussed in Section "Comparing `namedCapture::df_match_variable` with other functions for data.frames." Figure 2 (right) shows a comparison with **tidyr** and **rematch2**. Again we expected small differences between the packages, and we observed linear time complexity for **tidyr**, **rematch2**, and **namedCapture** (using either PCRE or RE2).

The fourth example shows the worst case time complexity, using a pathological regular expression of increasing size (with backreferences) on a subject of increasing size. For example with size $N = 2$ we use the regex `(a)?(a)?\1\1` on the pattern `aa`; the match time complexity is $O(2^N)$. Note that possessive quantifiers, `(a)?+`, could be used to avoid the exponential time complexity (but possessive quantifers are only supported in PCRE and ICU, not TRE nor RE2). Figure 3 (left) shows a comparison between ICU, PCRE, and TRE (RE2 is not included because it does not support backreferences). It is clear that all three libraries suffer from exponential time complexity. Although these timings are not typical, they illustrate the worst case time complexity that can be achieved. Such information should be considered along with other features (Table 1) when choosing a regex library. For example, guaranteed linear time complexity is essential for avoiding denial-of-service attacks in situations where potentially malicious users are permitted to define the regular expression pattern.

The final example involves using a pathological regular expression of increasing size (without backreferences) on a subject of increasing size. Figure 3 (right) shows a comparison between the previous libraries and additionally RE2. Is is clear that the fastest libraries are TRE and RE2, which

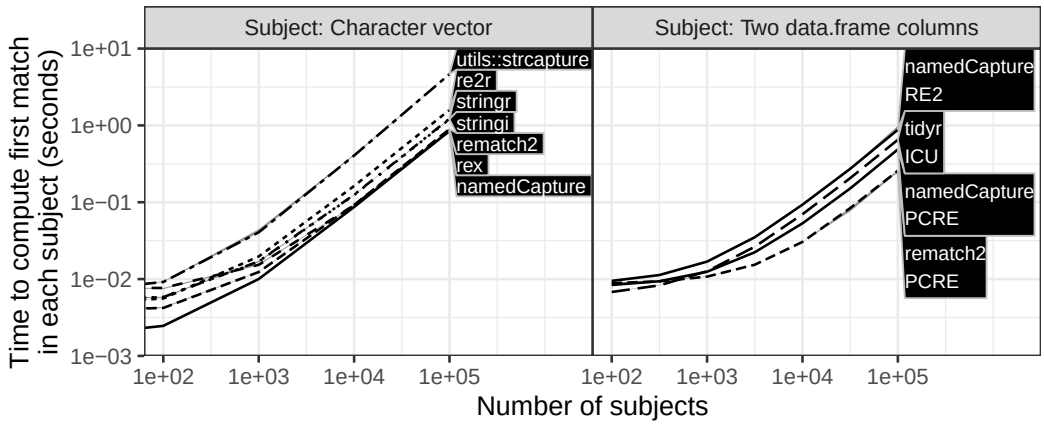


Figure 2: Computation time for finding the first match is plotted as a function of subject size (median lines and quartile bands over 5 timings). Such timings are typical for real-world subjects and patterns such as the two examples shown.

exhibit linear time complexity. The slowest algorithm is clearly ICU, which exhibits exponential time complexity. The PCRE library is exponential up to a certain pattern/subject size, after which it is constant, because of a default limit PCRE imposes on backtracking. If other libraries allow configuring a limit on backtracking, such an option could be used to avoid this exponential time complexity. Again these timings are on synthetic data which achieve the worst case time complexity, and are not typical of real data. Overall this comparison suggests that for guaranteed fast matching, RE2 must be used, via the **re2r** or **namedCapture** packages.

Discussion and conclusions

Our comparisons showed how similar operations can be performed by **namedCapture** and other R packages (e.g. **tidyr** and **rex**). Our empirical timings revealed an inefficient implementation of the `substring/gregexpr` functions in R-3.5.2, which was fixed in R-3.6.0 as a result of this research. After applying that fix, all packages were asymptotically linear in our empirical comparisons of time to compute matches using typical/real-world patterns and subjects. Finally, we studied the worst-case time complexity of matching on pathological patterns/subjects, and showed that RE2 must be used for guaranteed linear time complexity.

The article presented the **namedCapture** package, along with detailed comparisons with other R packages for regular expressions. A unique feature of the **namedCapture** package is its compact and readable syntax for defining regular expressions in R code. We showed how this syntax can be used to extract data tables from a variety of non-tabular text data. We also highlighted several other features of the **namedCapture** package, which include support for arbitrary type conversion functions, named output based on subject names and the `name` capture group, and two regex engines (PCRE and RE2). PCRE can be used for backreferences (e.g. for matching HTML tags), but otherwise RE2 should be preferred for guaranteed linear time complexity. The ICU library may be preferred for its strong unicode support (Table 1), so we are considering implementing ICU as another regex engine usable in **namedCapture**.

We thank a reviewer for a suggestion about other choices for the variable argument syntax for specifying type conversion functions. The current syntax uses a named R argument to specify the capture group name, then a character string literal to specify the capture group pattern, then a function name specify the type conversion. Other choices could use formulas or the `:=` operator to define type conversions. Overall we hope that the unique features of the **namedCapture** package will be useful and inspiring for other package developers.

Reproducible research statement. The source code for this article can be freely downloaded from `https://github.com/tdhock/namedCapture-article`

Bibliography

G. Csárdi. **rematch2***: Tidy Output from Regular Expression Matching*, 2017. URL `https://CRAN.R-project.org/package=rematch2`. R package version 2.0.1. [p329]

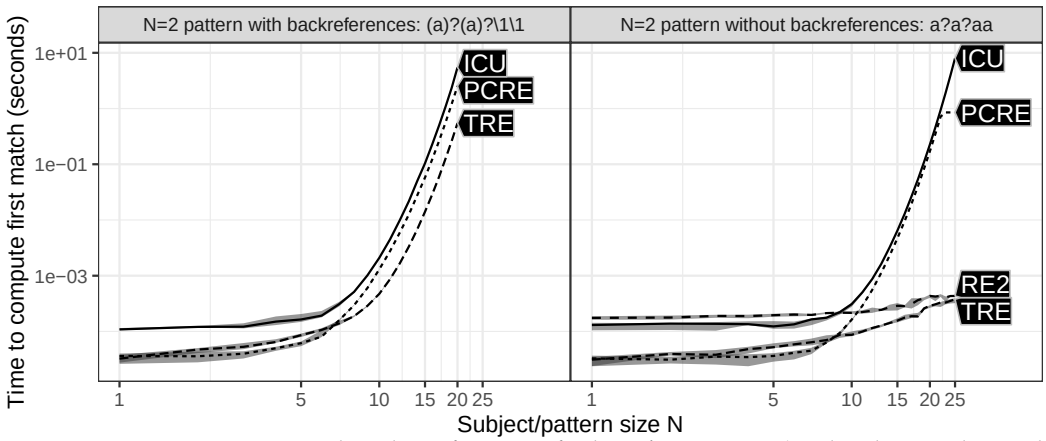


Figure 3: Computation time is plotted as a function of subject/pattern size (median lines and quartile bands over 10 timings). For $N = 2$ the subject is aa and the pattern is shown in the facet title. Such slow timings only result from pathological subject/pattern combinations.

J. E. F. Friedl. *Mastering Regular Expressions*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2 edition, 2002. [p328]

M. Gagolewski. *R package* **stringi***: Character string processing facilities*, 2018. URL http://www.gagolewski.com/software/stringi/. [p330]

P. Hazel. ChangeLog for PCRE, 2003. URL https://github.com/tdhock/regex-tutorial/blob/master/pcre1-changelog.txt. [p328]

T. D. Hocking. Bug 14518 - wishlist: named capture in regular expressions, 2011a. URL https://bugs.r-project.org/bugzilla3/show_bug.cgi?id=14518. [p329]

T. D. Hocking. Fast, named capture regular expressions in R 2.14. In *useR 2011 conference proceedings*, 2011b. URL http://web.warwick.ac.uk/statsdept/user-2011/TalkSlides/Lightening/2-StatisticsAndProg_3-Hocking.pdf. [p329]

ICU developers. Download ICU 55, 2015a. URL http://site.icu-project.org/download/55. [p330]

ICU developers. Named capture groups, 2015b. URL http://bugs.icu-project.org/trac/ticket/5312. [p330]

W. L. Johnson, J. H. Porter, S. I. Ackley, and D. T. Ross. Automatic generation of efficient lexical processors using finite state techniques. *Commun. ACM*, 11(12):805–813, Dec. 1968. ISSN 0001-0782. URL https://doi.org/10.1145/364175.364185. [p328]

S. C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956. URL http://www.diku.dk/hjemmesider/ansatte/henglein/papers/kleene1956.pdf. [p328]

V. Laurikari. *TRE: The free and portable approximate regex matching library*, 2019. URL https://laurikari.net/tre/. [p329]

Python developers. Python 1.5.2 history, 1997a. URL https://github.com/tdhock/regex-tutorial/blob/master/python-1.5.2-Misc-HISTORY.txt. [p328]

Python developers. Python documentation for built-in module re, 1997b. URL https://github.com/tdhock/regex-tutorial/blob/master/python-1.5-Doc-libre.tex. [p328]

R Core Team. News for the 0.x series, 1997. URL https://cloud.r-project.org/src/base/NEWS.0. [p329]

R Core Team. News for the 1.x series, 2002. URL https://github.com/tdhock/regex-tutorial/blob/master/R.NEWS.1.txt. [p329]

R Core Team. News for the 2.x series, 2009. URL https://cloud.r-project.org/src/base/NEWS.2. [p329]

K. Thompson. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11(6): 419–422, June 1968. ISSN 0001-0782. [p328]

K. Ushey, J. Hester, and R. Krzyzanowski. **rex**: *Friendly Regular Expressions*, 2017. URL https://CRAN.R-project.org/package=rex. R package version 1.1.2. [p330]

Q. Wenfeng. *pkgre2r: RE2 Regular Expression*, 2017. URL https://CRAN.R-project.org/package=re2r. R package version 0.2.0. [p330]

H. Wickham and L. Henry. **tidyr**: *Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2018. URL https://CRAN.R-project.org/package=tidyr. R package version 0.8.2. [p330]

Toby Dylan Hocking
School of Informatics, Computing, and Cyber Systems
Northern Arizona University
Flagstaff, Arizona
USA
toby.hocking@nau.edu

The Landscape of R Packages for Automated Exploratory Data Analysis

by Mateusz Staniak and Przemysław Biecek

Abstract The increasing availability of large but noisy data sets with a large number of heterogeneous variables leads to the increasing interest in the automation of common tasks for data analysis. The most time-consuming part of this process is the Exploratory Data Analysis, crucial for better domain understanding, data cleaning, data validation, and feature engineering.

There is a growing number of libraries that attempt to automate some of the typical Exploratory Data Analysis tasks to make the search for new insights easier and faster. In this paper, we present a systematic review of existing tools for Automated Exploratory Data Analysis (autoEDA). We explore the features of fifteen popular R packages to identify the parts of analysis that can be effectively automated with the current tools and to point out new directions for further autoEDA development.

Introduction

With the advent of tools for automated model training (autoML), building predictive models is becoming easier, more accessible and faster than ever. Tools for R such as mlrMBO (Bischl et al., 2017), parsnip (Kuhn and Vaughan, 2019); tools for python such as TPOT (Olson et al., 2016), auto-sklearn (Feurer et al., 2015), autoKeras (Jin et al., 2018) or tools for other languages such as H2O Driverless AI (H2O.ai, 2019; Cook, 2016) and autoWeka (Kotthoff et al., 2017) supports fully- or semi-automated feature engineering and selection, model tuning and training of predictive models.

Yet, model building is always preceded by a phase of understanding the problem, understanding of a domain and exploration of a data set. Usually, in the process of the data analysis much more time is spent on data preparation and exploration than on model tuning. This is why the current bottleneck in data analysis is in the exploratory data analysis (EDA) phase. Recently, a number of tools were developed to automate or speed up the part of the summarizing data and discovering patterns. Since the process of building predictive models automatically is referred to as autoML, we will dub the automation of data exploration autoEDA. The surge in interest in autoEDA tools[1] is evident in the Figure 1. Table 1 describes the popularity of autoEDA tools measured as the number of downloads from CRAN and usage statistics from Github[2].

There is an abundance of R libraries that provide functions for both graphical and descriptive data exploration. Here, we restrict our attention to packages that aim to automatize or significantly speed up the process of exploratory data analysis for tabular data. Such tools usually work with full data frames, which are processed in an automatic or semi-automatic manner, for example by guessing data types, and return summary tables, groups of plots or full reports. Currently, there is no CRAN Task View dedicated to packages for automated Exploratory Data Analysis and neither was there any repository that would catalogue them[3]. Here, we make a first attempt to comprehensively describe R tools for autoEDA. We chose two types of packages. The first group explicitly aims to automate EDA, as stated in the description of the package. These includes packages for *fast*, *easy*, *interactive* or *automated* data exploration. The second group contains packages that create data summaries. These packages were included, as long as they address at least two analysis goals listed in Table 2. We do not describe in detail packages that are either restricted to one area of application (for example **RBioPlot** (Zhang and Storey, 2016) package dedicated to biomolecular data or **intsvy** (Caro and Biecek, 2017) package focused on international large-scale assessments), designed for one specific task (for example creating tables), or in an early development phase. Some of the more task-specific packages are briefly discussed in the section Other packages. Some packages, such as **radiant** (Nijs, 2019) cover the full analysis pipeline and, as such, are too general for our purposes, even though they include an EDA module.

This paper has two main goals. First is to characterize existing R packages for automated Exploratory Data Analysis and compare their ranges of capabilities. To our best knowledge, this is first such a review. Previously, a smaller comparison of seven packages was done in Putatunda et al. (2019). Second is to identify areas, where automated data exploration could be improved. In particular, we are interested in gauging the potential of AI-assisted EDA tools.

The first goal is addressed in Sections R packages for automated EDA and Feature comparison

[1] Access the raw data with `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/aec9")`

[2] Access the data with `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/50a7")`

[3] The first author maintains a list of papers related to autoEDA and software tools in different languages at https://github.com/mstaniak/autoEDA-resources

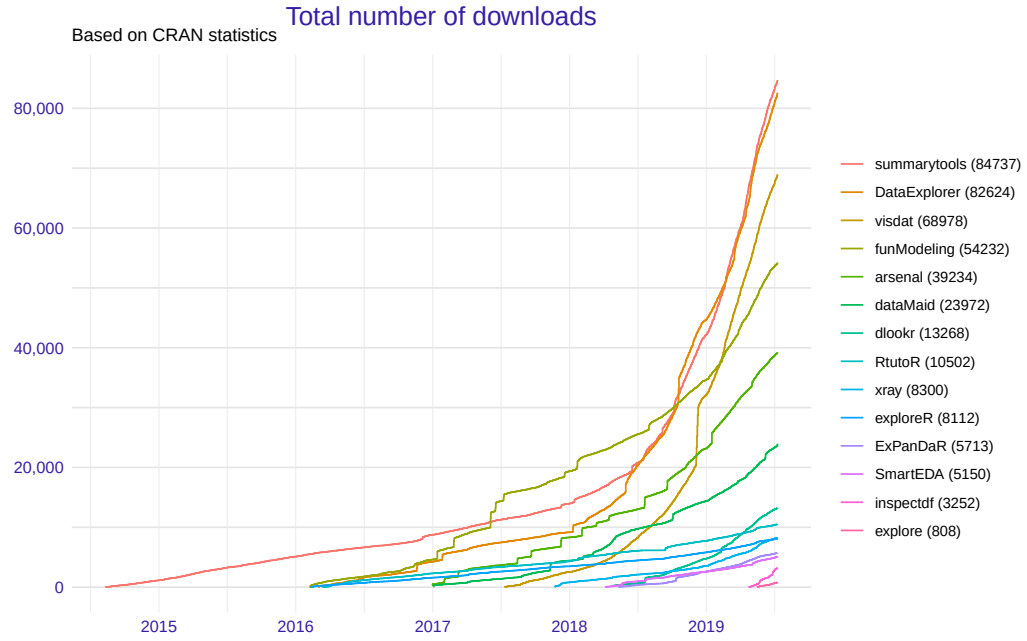


Figure 1: Trends in number of downloads of autoEDA packages available on CRAN since the first release. Data was gathered on 12.07.2019 with the help of the **cranlogs** package (Csardi, 2015).

where we first briefly describe each package and the compare, how are different EDA tasks are tackled by these packages. Then, in Section Discussion *Summary*, we compile a list of strong and weak points of the automated EDA software and detail some open problems.

The tasks of Exploratory Data Analysis

Exploratory Data Analysis is listed as an important step in most methodologies for data analysis (Biecek, 2019; Grolemund and Wickham, 2019). One of the most popular methodologies, the CRISP-DM (Wirth, 2000), lists the following phases of a data mining project:

1. Business understanding.
2. Data understanding.
3. Data preparation.
4. Modeling.
5. Evaluation.
6. Deployment.

Automated EDA tools aim to make the Data understanding phase as fast and as easy as possible. This part of a project can be further divided into smaller tasks. These include a description of a dataset, data exploration, and data quality verification. All these tasks can be achieved both by providing descriptive statistics and numerical summaries and by visual means. AutoEDA packages provide functions to deal with these challenges. Some of them are also concerned with simple variable transformations and data cleaning. Both these tasks belong in the Data preparation phase, which precedes and supports the model building phase. Let us notice that business understanding is affected by data understanding, which makes this part of the analysis especially important.

Goals of autoEDA tools are summarised in Table 2. The *Phase* and *Tasks* columns are taken from the CRISP-DM standard, while *Type* and *Examples* columns provide examples based on current functionalities of autoEDA packages.

Each task should be summarised in a report, which makes reporting another relevant problem of autoEDA. Uni- and bivariate data exploration is a part of the analysis that is most thoroughly covered by the existing autoEDA tools. The form of univariate summaries depends on the variable type. For numerical variables, most packages provide descriptive statistics such as centrality and dispersion measures. For categorical data, unique levels and associated counts are reported. Bivariate relationships descriptions display either dependency between one variable of interest and all other variables, which includes contingency tables, scatter plots, survival curves, plots of distribution by

	CRAN			GitHub				
package	downl.	debut	age	stars	commits	contrib.	issues	forks
arsenal	39234	2016-12-30	2y 6m	59	637	3	200	4
autoEDA	-	-	-	41	20	1	4	12
DataExplorer	82624	2016-03-01	3y 4m	235	187	2	121	44
dataMaid	23972	2017-01-02	2y 6m	68	473	2	45	18
dlookr	13268	2018-04-27	1y 2m	35	54	3	9	12
ExPanDaR	5713	2018-05-11	1y 2m	32	197	2	3	14
explore	808	2019-05-16	0y 1m	15	114	1	1	0
exploreR	8112	2016-02-10	3y 5m	1	1	1	0	0
funModeling	54232	2016-02-07	3y 5m	58	126	2	13	18
inspectdf	3252	2019-04-24	0y 2m	117	200	2	12	11
RtutoR	10502	2016-03-12	3y 3m	13	7	1	4	8
SmartEDA	5150	2018-04-06	1y 3m	4	4	1	1	2
summarytools	84737	2014-08-11	4y 11m	255	981	6	76	33
visdat	68978	2017-07-11	2y 0m	313	426	12	122	39
xray	8300	2017-11-22	1y 7m	63	33	4	10	5

Table 1: Popularity of R packages for autoEDA among users and package developers. First two columns summarise CRAN statistics, last five columns summarise package development at GitHub. When a repository owned by the author is not available, the data were collected from a CRAN mirror repository. Data was gathered on 12.07.2019.

values of a variable (histograms, bar plots, box plots), or between all pairs of variables (correlation matrices and plots), or chosen pairs of variables.

R packages for automated EDA

In this section, fifteen R libraries are shortly summarised. One of them is only available on GitHub (**autoEDA**), all others are available at CRAN. For each library, we include example outputs. The exact versions of packages that were used to create them can be found in the reference section. All examples are based on a subset of `typical_data`[4] dataset from **visdat** package. Whenever possible, **archivist** (Biecek and Kosinski, 2017) hooks are provided for easy access to the presented objects. When a function call only gives side-effects, a link is provided to the full result (PDF/PNG files). Tables were prepared with the **xtable** package (Dahl et al., 2018).

The arsenal package

The **arsenal** package (Heinzen et al., 2019) is a set of four tools for data exploration:

1. table of descriptive statistics and p-values of associated statistical tests, grouped by levels of a target variable (the so-called *Table 1*). Such a table can also be created for paired observation, for example longitudinal data (`tableby` and `paired` functions),
2. comparison of two data frames that can detect shared variables (`compare` function),
3. frequency tables for categorical variables (`freqlist` function),
4. fitting and summarizing simple statistical models (linear regression, Cox model, etc) in tables of estimates, confidence intervals and p-values (`modelsum` function).

Results of each function can be saved to a short report using the `write2` function. An example[5] can be found in Figure 2.

A separate vignette is available for each of the functions. `arsenal` is the most statistically-oriented package among reviewed libraries. It borrows heavily from SAS-style procedures used by the authors at the Mayo Clinic.

The autoEDA package

autoEDA package (Horn, 2018a) is a GitHub-based tool for univariate and bivariate visualizations and

[4] Access the data with `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/278c7")`
[5] Access the table with `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/d951")`

Phase	Task	Type	Examples
Data understanding	Data description	dimensions variables meta-data	variables number variable type size in RAM
	Data validity	invalid values missing values atypical values	typos NA count outliers
	Data exploration	univariate bivariate multivariate	histogram scatter plot Parallel coord. plot
Data preparation	Data cleaning	Imputation Outlier treatment	Impute mean Impute median
	Derived attributes	Dimension reduction Continuous Categorical	PCA Box-Cox transform Binning Merge rare factors

Table 2: Early phases of data mining project according to CRISP-DM standard, their specific goals and examples of how they are aided by autoEDA tools. (Wirth, 2000)

	FALSE (N=465)	TRUE (N=535)	Total (N=1000)	p value
Smokes				0.207
FALSE	384 (82.6%)	425 (79.4%)	809 (80.9%)	
TRUE	81 (17.4%)	110 (20.6%)	191 (19.1%)	
Race				0.359
N-Miss	52	55	107	
White	261 (63.2%)	318 (66.2%)	579 (64.8%)	
Hispanic	75 (18.2%)	71 (14.8%)	146 (16.3%)	
Black	56 (13.6%)	58 (12.1%)	114 (12.8%)	
Asian	14 (3.4%)	14 (2.9%)	28 (3.1%)	
Bi-Racial	5 (1.2%)	13 (2.7%)	18 (2.0%)	
Native	2 (0.5%)	5 (1.0%)	7 (0.8%)	
Other	0 (0.0%)	1 (0.2%)	1 (0.1%)	
Hawaiian	0 (0.0%)	0 (0.0%)	0 (0.0%)	

Figure 2: An example output from the `arsenal::tableby` function saved using `arsenal::write2` (**arsenal** v 2.0). `Smokes` and `Race` variables are compared by the levels of `Died` variable.

summaries. The `dataOverview` function returns a data frame that describes each feature by its type, number of missing values, outliers and typical descriptive statistics. Values proposed for imputation are also included. Two outlier detection methods are available: Tukey and percentile-based. A PDF report can be created using the `autoEDA` function. It consists of the plots of distributions of predictors grouped by outcome variable or distribution of outcome by predictors.

The package can be found on Xander Horn's GitHub page: https://github.com/XanderHorn/autoEDA. It does not include a vignette, but a short introduction article was published to LinkedIn (Horn, 2018b) and similar examples can be found in the readme of the project. Plots from a report[6] generated by **autoEDA** are displayed in Figure 3.

The DataExplorer package

DataExplorer (Cui, 2019) is a recent package that helps automatize EDA and simple data transformations. It provides functions for:

1. whole dataset summary: dimensions, types of variables, missing values, etc (`introduce` and

[6]Find the full report at https://github.com/mstaniak/autoEDA-resources/blob/master/autoEDA-paper/plots/autoEDA/autoEDA_report.pdf

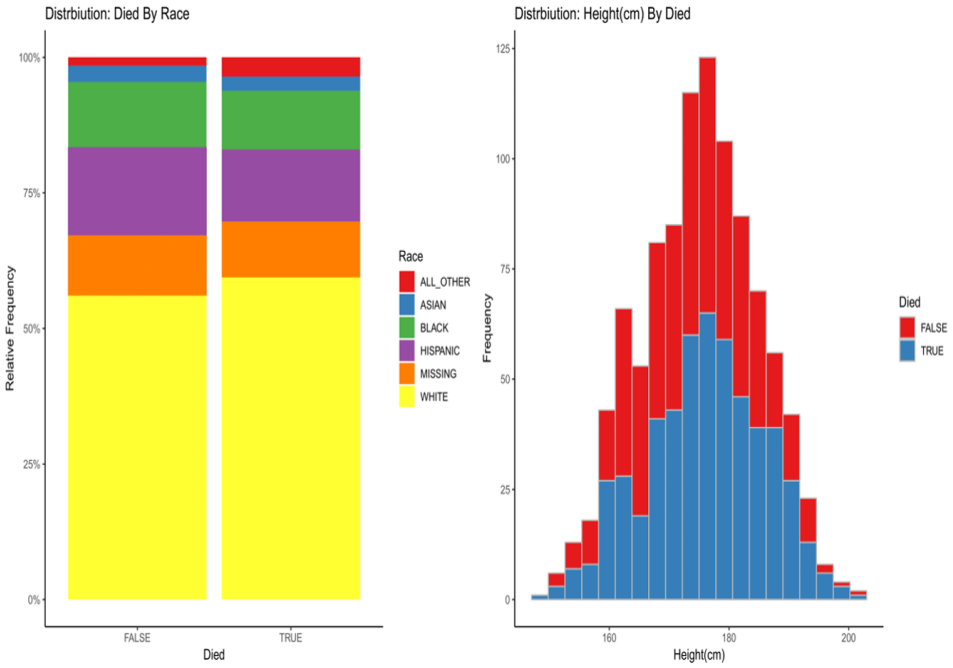


Figure 3: Sample pages from the report generated by the `autoEDA::autoEDA` function (**autoEDA** v. 1.0) displaying bivariate relationships between the target and explanatory variable.

`plot_intro` functions),

2. missing values profile as a plot of missing values fraction per column (`plot_missing` function) or summary statistics and suggested actions (`profile_missing` function),
3. plotting distributions of variables, separately numerical and categorical (`plot_histogram` and `plot_bar` functions),
4. QQ Plots (`plot_qq` function),
5. plotting correlation matrices (`plot_correlation` function),
6. visualizing PCA results by plotting percentage of explained variance and correlations with each original feature for every principal component (`plot_prcomp` function),
7. plotting relationships between the target variable and predictors - scatterplots and boxplots (`plot_scatterplot` and `plot_boxplot` functions),
8. data transformation: replacing missing values by a constant (`set_missing` function), grouping sparse categories (`group_category` function), creating dummy variables, dropping columns (`dummify`, `drop_features` functions) and modifying columns (`update_columns` function).

The `create_report` function generates a report. By default, it consists of all the above points except for data transformations and it can be further customized. An introductory vignette *Introduction to DataExplorer* that showcases all the functionalities is included in the package. It is noticeable that the package almost entirely relies on visual techniques. Plots taken from an example report[7] are presented in Figure 4.

The dataMaid package

The **dataMaid** (Petersen and Ekstrom, 2018) package has two central functions: the `check` function, which performs checks of data consistency and validity, and `summarize`, which summarizes each column. Another function, `makeDataReport`, automatically creates a report in PDF, DOCX or HTML format. The goal is to detect missing and unusual - outlying or incorrectly encoded - values. The report contains whole dataset summary: variables and their types, number of missing values, and univariate summaries in the form of descriptive statistics, histograms/bar plots and an indication of possible problems.

[7] Access the full report https://github.com/mstaniak/autoEDA-resources/blob/master/autoEDA-paper/plots/DataExplorer/dataexplorer_example.pdf

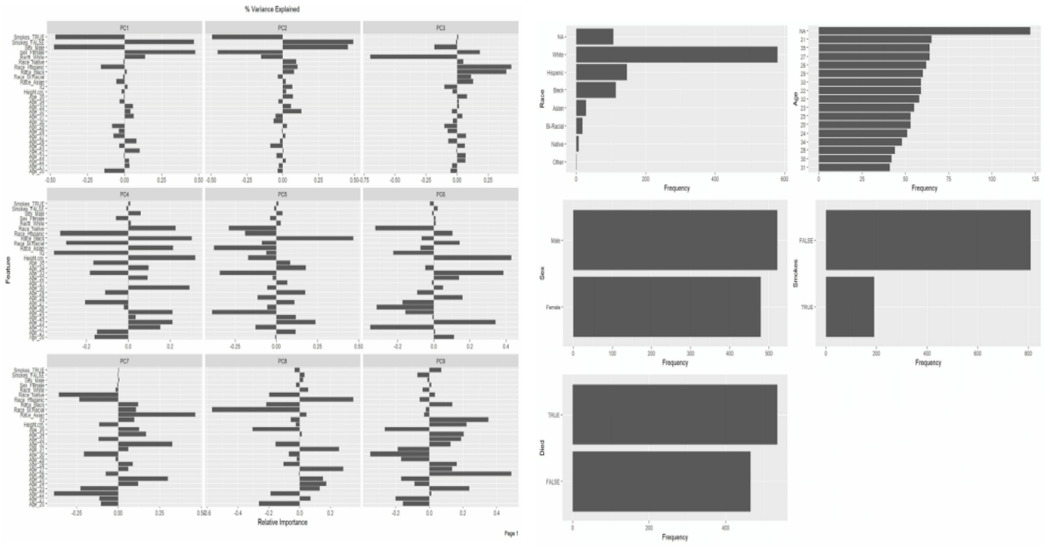


Figure 4: A visualization of PCA results - correlation with original variables for each principal component - and a *wall of bar plots* taken from a report generated by the `DataExplorer::create_report` function (**DataExplorer** v. 0.7).

User-defined checks and summaries can be also included in the analysis. The vignette *Extending dataMaid* explains how to define them. It is also possible to customize the report. In particular, variables for which no problems were detected can be ommited. An example report[8] can be found in Figure 5.

The dlookr package

The **dlookr** (Ryu, 2019) package provides tools for 3 types of analysis: data diagnosis including correctness, missing values, outlier detection; exploratory data analysis; and variable transformations: imputation, dichotomization, and transformation of continuous features. It can also automatically generate a PDF report for all these analyses.

For data diagnosis, types of variables are reported along with counts of missing values and unique values. Variables with a low proportion of unique values are described separately. All the typical descriptive statistics are provided for each variable. Outliers are detected and distributions of variables before and after outlier removal are plotted. Both missing values and outliers can be treated using `impute_na` and `impute_outlier` functions.

In the EDA report, descriptive statistics are presented along with normality tests, histograms of variables and their transformations that reduce skewness: logarithm and root square. Correlation plots are shown for numerical variables. If the target variable is specified, plots that show the relationship between the target and each predictor are also included.

A transformation report compares descriptive statistics and plots for each variable before and after imputation, skewness-removing transformation and binning. If the right transformation is found among the candidate transformations, it can be applied to the feature through one of the `binning`, `binning_by`, or `transform` functions.

Every operation or summary presented in the reports can also be performed manually. A dedicated vignette explains each of the main functionalities (*Data quality diagnosis*, *Data Transformation*, *Exploratory Data Analysis* vignettes). An example[9] taken from one of the reports can be found in Figure 6.

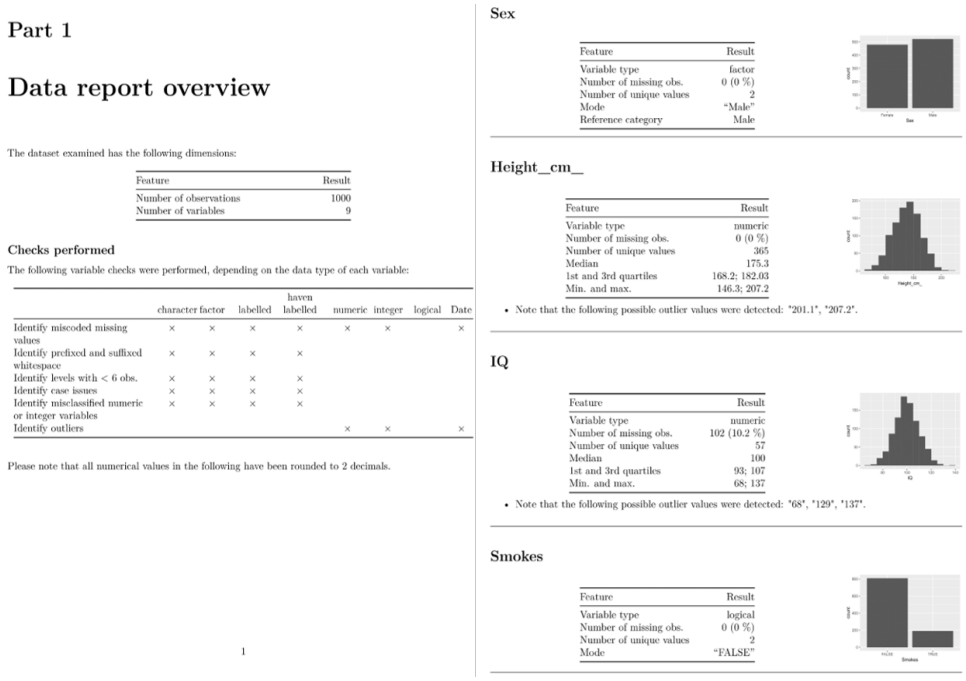
The ExPanDaR package

Notably, while the **ExPanDaR** package (Gassen, 2018) was designed for panel data exploration, it can also be used for standard EDA after adding an artificial constant time index. In this case, the package offers interactive **shiny** application for exploration. Several types of analysis are covered:

1. missing values and outlier treatment,

[8]Find the full report at https://github.com/mstaniak/autoEDA-resources/blob/master/autoEDA-paper/plots/dataMaid/dataMaid_report.pdf

[9]Access the full report at https://github.com/mstaniak/autoEDA-resources/blob/master/autoEDA-paper/plots/dlookr/dlookr_eda.pdf



Part 1

Data report overview

The dataset examined has the following dimensions:

Feature	Result
Number of observations	1000
Number of variables	9

Checks performed

The following variable checks were performed, depending on the data type of each variable:

	character	factor	labelled	haven labelled	numeric	integer	logical	Date
Identify miscoded missing values	×	×	×	×	×	×		×
Identify prefixed and suffixed whitespace	×	×	×	×				
Identify levels with < 6 obs.	×	×	×	×				
Identify case issues	×	×	×	×				
Identify misclassified numeric or integer variables	×	×	×	×				
Identify outliers					×	×		×

Please note that all numerical values in the following have been rounded to 2 decimals.

1

Sex

Feature	Result
Variable type	factor
Number of missing obs.	0 (0 %)
Number of unique values	2
Mode	"Male"
Reference category	Male

Height_cm_

Feature	Result
Variable type	numeric
Number of missing obs.	0 (0 %)
Number of unique values	365
Median	175.3
1st and 3rd quartiles	168.2; 182.03
Min. and max.	146.3; 207.2

- Note that the following possible outlier values were detected: "201.1", "207.2".

IQ

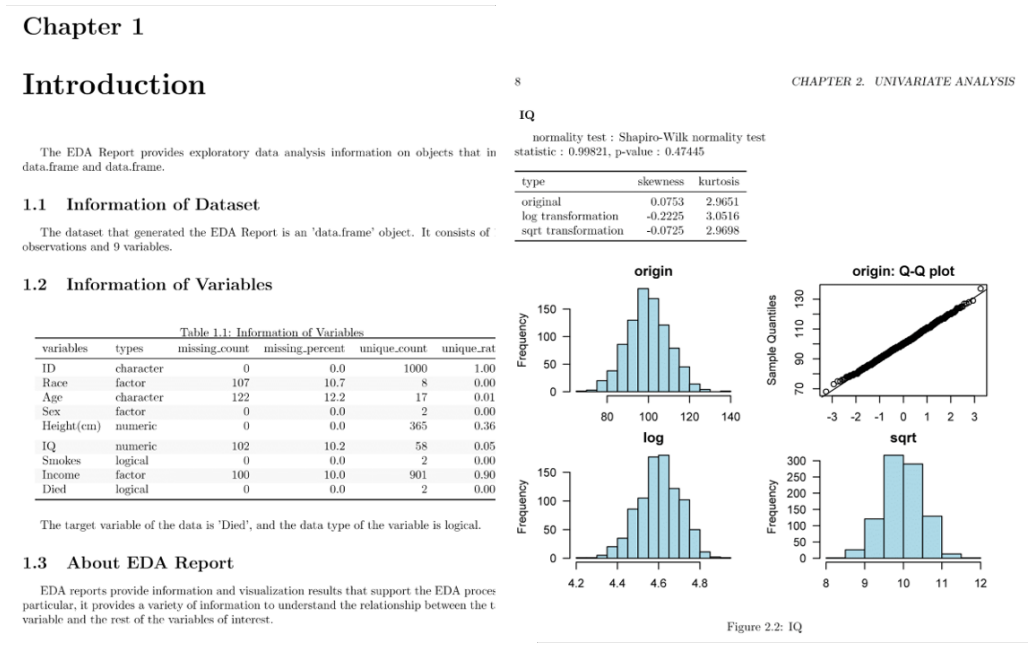
Feature	Result
Variable type	numeric
Number of missing obs.	102 (10.2 %)
Number of unique values	57
Median	100
1st and 3rd quartiles	93; 107
Min. and max.	68; 137

- Note that the following possible outlier values were detected: "68", "129", "137".

Smokes

Feature	Result
Variable type	logical
Number of missing obs.	0 (0 %)
Number of unique values	2
Mode	"FALSE"

Figure 5: Two pages from a data validity report generated using the `dataMaid::makeDataReport` function (**dataMaid** v. 1.2). Atypical values are listed under the variable summary.



Chapter 1

Introduction

The EDA Report provides exploratory data analysis information on objects that in data.frame and data.frame.

1.1 Information of Dataset

The dataset that generated the EDA Report is an 'data.frame' object. It consists of observations and 9 variables.

1.2 Information of Variables

Table 1.1: Information of Variables

variables	types	missing_count	missing_percent	unique_count	unique_rat
ID	character	0	0.0	1000	1.00
Race	factor	107	10.7	8	0.00
Age	character	122	12.2	17	0.01
Sex	factor	0	0.0	2	0.00
Height(cm)	numeric	0	0.0	365	0.36
IQ	numeric	102	10.2	58	0.05
Smokes	logical	0	0.0	2	0.00
Income	factor	100	10.0	901	0.90
Died	logical	0	0.0	2	0.00

The target variable of the data is 'Died', and the data type of the variable is logical.

1.3 About EDA Report

EDA reports provide information and visualization results that support the EDA proces particular, it provides a variety of information to understand the relationship between the t variable and the rest of the variables of interest.

8 CHAPTER 2. UNIVARIATE ANALYSIS

IQ

normality test : Shapiro-Wilk normality test
statistic : 0.99821, p-value : 0.47445

type	skewness	kurtosis
original	0.0753	2.9651
log transformation	-0.2225	3.0516
sqrt transformation	-0.0725	2.9698

Figure 2.2: IQ

Figure 6: Two pages from a report generated by the `dlookr::eda_report` function (**dlookr** v. 0.3.8). First, the dataset is summarised, than each variable is described. Optionally, plots of bivariate relationships can be added.

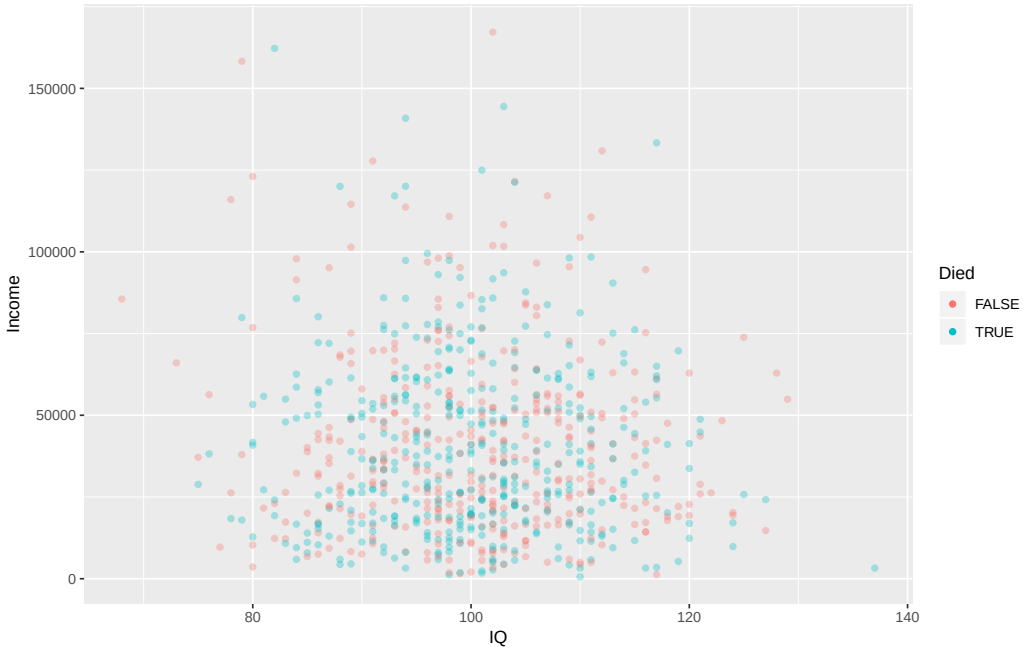


Figure 7: Scatter plot with of `Income` and `IQ` variables with `Died` variable denoted by the color. Created with the `prepare_scatter_plot` function (**ExPanDaR** v0.4.0).

2. univariate summaries (descriptive statistics) and plots (histograms/bar plots),
3. bivariate analysis via correlation matrices and plots. Interestingly, scatter plots can be enriched by associating size and color of points with variables,
4. multivariate regression analysis.

For each functionality of the application, there is a corresponding standalone function.

Three vignettes describe how the library can be used for data exploration (*Using the functions of the ExPanDaR package*), how to customize it (*Customize ExPanD*) and how to analyze panel data (*Using ExPanD for Panel Data Exploration*) Example instances of **ExPanDaR shiny** applications are available online. Links and other examples can be found in the GitHub repository of the package: `https://github.com/joachim-gassen/ExPanDaR`. An example of a scatter plot[10] created by the package can be found in Figure 7.

The explore package

The functionalities of the **explore** package (Krasser, 2019) can be accessed in three ways: through an interactive **shiny** (Chang et al., 2019) application, through an automatically generated HTML report or via standalone functions. In addition to data exploration, relationships with a binary target can be explored. The package includes functions for

1. full dataset summaries - dimensions, data types, missing values and summary statistics (`describe` function),
2. uni- and bivariate visualizations, including density plots, bar plots and boxplots (a family of `explore` functions, in particular `explore_all` function that creates plots for all variables),
3. simple modeling based on decision trees (`explain_tree` function) or logistic regression (`explain_logreg` function).

All result can be saved to HTML via the `report` function. Dataset and variable summaries can also be save to an MD file using the `data_dict_md` function[11]. The *explore* vignette includes a thorough description of the package. An example decision tree[12] can be found in Figure 8.

[10] Access the R object with the `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/9c5d")`

[11] Find examples at `https://github.com/mstaniak/autoEDA-resources/tree/master/autoEDA-paper/plots/explore`

[12] Access the R object with the `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/dc47")`.

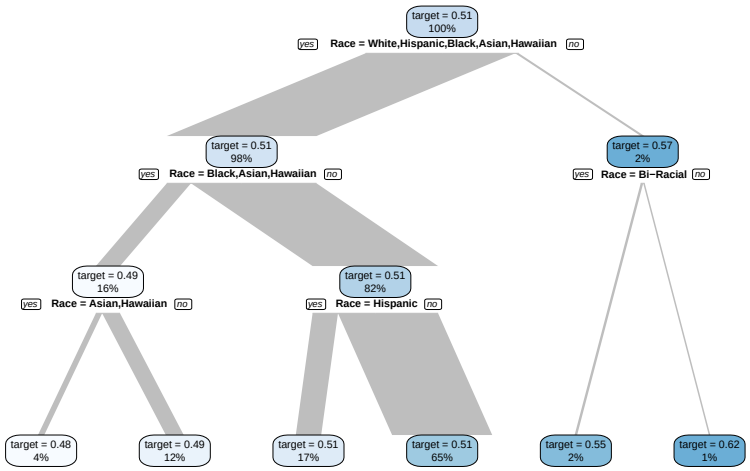


Figure 8: A decision tree fitted using the `explain_tree` function (**explore** v. 0.4.3). The tree can also be based on multiple explanatory variables.

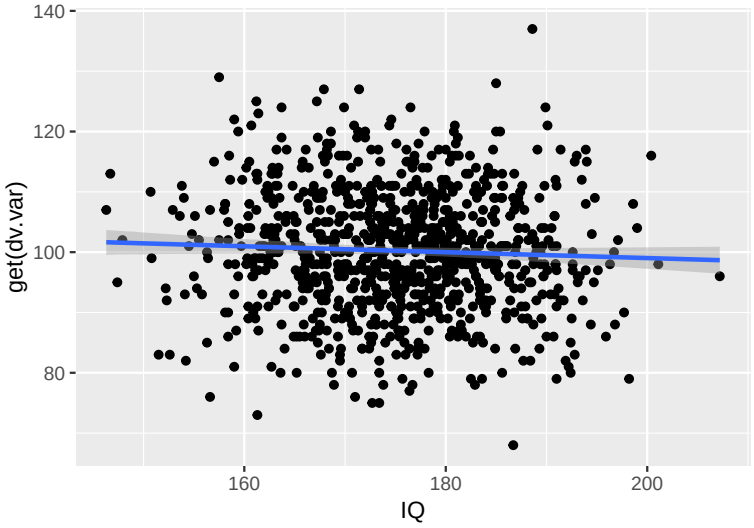


Figure 9: Univariate regression plot created using the `exploreR::massregplot` (**exploreR** v. 0.1).

The exploreR package

The **exploreR** package (Coates, 2016) takes a unique approach to data exploration compared to other packages. The analysis is based on linear regression. There are three functionalities:

1. fitting univariate regression model for each independent variable and summarizing the results in a table that consists of estimated parameters, p-values, and R^2 values (`masslm` function),
2. plotting target variable against each independent variable along with the fitted least squares line (`massregplot` function),
3. feature standardization by scaling to the interval $[0, 1]$ or subtracting mean and dividing by standard deviation.

Regression plots can be saved to a PDF file. A vignette called *The How and Why of Simple Tools* explains all the functions and provides examples. One of the regression plots[13] is presented in Figure 9.

[13] A PDF file with all the plots can be found at https://github.com/mstaniak/autoEDA-resources/blob/master/autoEDA-paper/plots/exploreR.pdf

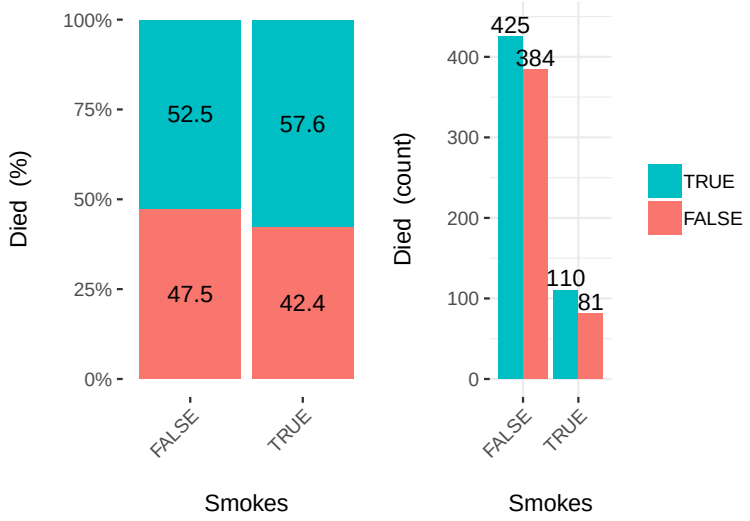


Figure 10: An example output from the `funModeling::cross_plot` function (**funModeling** v. 1.7). Such a plot is drawn for every variable in the dataset or for a specified subset of variables. Continuous features are discretized.

The funModeling package

The package **funModeling** (Casas, 2019) is a rich set of tools for EDA connected to the book Casas (2018). These tools include

1. dataset summary (`df_status` function),
2. plots and descriptive statistics for categorical and numerical variables (`plot_num`, `profiling_num` and `freq` functions),
3. classical and information theory-based correlation analysis for target variable vs other variables - (`correlation_table` function for numerical predictors, `var_rank_info` function for all predictors),
4. plots of distribution of target variables vs predictors (bar plots, box plots and histograms via `cross_plot` and `plotar` functions),
5. quantitative analysis for binary target variables (`categ_analysis` function),
6. different methods of binning continuous features (`discretize_df`, `convert_df_to_categoric` and `discretize_rgr` functions),
7. variable normalization by transforming to the $[0, 1]$ interval (`range01` function),
8. outlier treatment (`prep_outliers`, `tukey_outlier` and `hampel_outlier` functions),
9. gain and lift curves (`gain_lift` function).

It is the only library that encompasses visualizations related to predictive models and non-standard correlation analysis. The range of tools covered by **funModeling** is very wide. The package includes an exhaustive introduction vignette called **funModeling** *quick-start*. One of the bivariate visualizations[14] offered by the package can be found in Figure 10.

The inspectdf package

The **inspectdf** package (Rushworth, 2019) provides several tools for basic data exploration with a consistent interface. Each of the `inspect_*` functions returns a data frame with summaries (and additional attributes). The results can be then plotted using the `show_plot` function. The function are related to three aspects of EDA:

1. whole dataset can be summarised by numbers of missing values, number of variables of each type and memory used by each variable (`inspect_na`, `inspect_types` and `inspect_mem` functions),

[14] Find all the plots at https://github.com/mstaniak/autoEDA-resources/tree/master/autoEDA-paper/plots/funmodeling

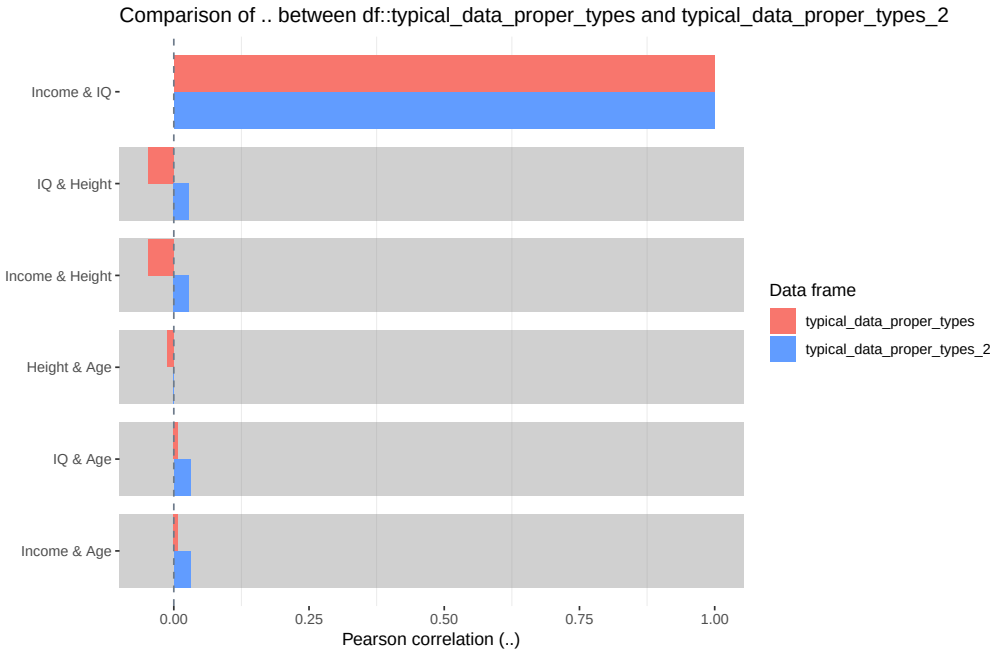


Figure 11: A comparison of correlations between numerical variables in two data frames. Plot created using the **inspectdf** package v. 0.0.3.

2. univariate analysis is done via summary statistics and histograms for numerical variables (`inspect_num` function), bar plots for categorical variables (`inspect_cat` function). Additionally, factors dominated by a single level can be found with the `inspect_imb` function,
3. bivariate relationships are described by Pearson correlation coefficient for numerical variables (`inspect_cor` function).

Notably, each function can take two data frames as parameters and return their comparison. An example of a correlation analysis plot comparing two data frames can be found in Figure 11 While the library does not include a vignette, extensive documentation with examples is provided on the GitHub webpage of the project: https://github.com/alastairrushworth/inspectdf.

The RtutoR package

The **RtutoR** package (Nair, 2018a) is a tool for automated reporting. There are three options for creating a report that contains univariate and bivariate data summaries:

1. plots can be created interactively in a **shiny** app (`launch_plotter` function),
2. the whole report can be generated from a **shiny** app that allows the user to tweak the report (`gen_exploratory_report_app` function),
3. the report can be created by a direct call to the `generate_exploratory_analysis_ppt` function.

The report is saved in the `PPTX` format. Notably, this package can identify the top `k` relevant variables based on a chosen criterion, for example, information gain, and display plots only for these variables. An example report can be found in the GitHub repository of the package[15]. The package was introduced in an R-Bloggers blog post (Nair, 2018b).

The SmartEDA package

The **SmartEDA** package (Ubrangala et al., 2018), is focused entirely on data exploration through graphics and descriptive statistics. It does not provide any functions which modify existing variables. The range of tools it includes is wide:

1. dataset summary (`ExpData` function),

[15]Find the report at https://github.com/anup50695/RtutoR/blob/master/titanic_exp_report_2.pptx

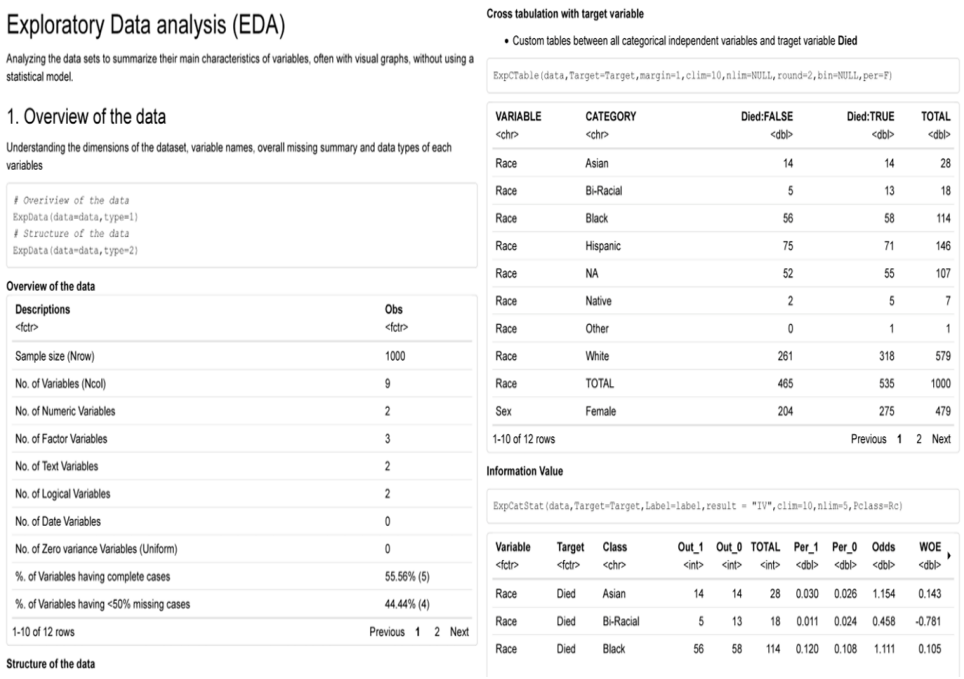


Figure 12: Sample pages from a report generated by the `SmartEDA::ExpReport` function (**SmartEDA** v. 0.3), including dataset overview and bivariate dependency for categorical variables.

2. descriptive statistics that may include correlation with target variable and density or bar plots (`ExpNumStat`, `ExpNumViz`, `ExpCatStat` and `ExpCatViz` functions). All visualizations may include the target variable,
3. QQ plots (`ExpOutQQ` function),
4. contingency tables (`ExpCTable` function),
5. information value and Weight of the Evidence coding (`ExpWoETable`, `ExpInfoValue` functions),
6. parallel coordinate plot for multivariate visualization (`ExpParcoord` function).

Plotting functions return grids of **ggplot2** object. The results can be written to a HTML report (`ExpReport` function). There are also additional functionalities dedicated to `data.table` objects from **data.table** package (Dowle and Srinivasan, 2019). An introductory vignette called *Explore data using SmartEDA (Intro)* is attached to the library. Another vignette *Custom summary statistics* describe customizing output tables. The package is also described in the Putatunda et al. (2019) paper. Examples[16] can be found in Figure 12.

The summarytools package

The **summarytools** package (Comtois, 2019) builds summary tables for whole datasets, individual variables, or pairs of variables. In addition, the output can be formatted to be included in **knitr**(Xie, 2015) or plain documents, HTML files and **shiny** apps (Chang et al., 2019). The are four main functionalities:

1. whole dataset summary including variable types and a limited number of descriptive statistics, counts of unique values and missing values and univariate plots within the output table (`dfSummary` function),
2. descriptive statistics, including skewness and kurtosis, for numerical variables, possibly grouped by levels of a factor (`descr`, `stby` functions),
3. counts and proportions for levels of categorical features (`freq` function),
4. contingency tables for pairs of categorical variables (`ctable` function).

All results can be saved and displayed in different formats. The package includes a vignette titled *Introduction to summarytools*. An example of univariate summaries[17] can be found in Figure 3.

[16] A full report is available at https://github.com/mstaniak/autoEDA-resources/blob/master/autoEDA-paper/plots/SmartEDA/smarteda_report_target.pdf

[17] Access the R object with `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/9e12")`.

	Height(cm)	IQ
Mean	175.09	100.23
Std.Dev.	9.83	10.03
Min	146.30	68.00
Q1	168.20	93.00
Median	175.30	100.00
Q3	182.05	107.00
Max	207.20	137.00
MAD	10.38	10.38
IQR	13.83	14.00
CV	0.06	0.10
Skewness	-0.08	0.08
SE.Skewness	0.08	0.08
Kurtosis	-0.30	-0.04
N.Valid	1000.00	898.00
% Valid	100.00	89.80

Table 3: An example table of descriptive statistics generated by the `summarytools::descr` function (**summarytools** v. 0.9.2).

The visdat package

The package **visdat** (Tierney, 2017) is maintained by rOpenSci. It consists of six functions that help visualize:

1. variables types and missing data (`vis_dat` function),
2. types of each value in each column (`vis_guess` function),
3. clusters of missing values (`vis_miss` function),
4. differences between the two datasets (`vis_compare` function),
5. where given conditions are satisfied in the data (`vis_expect` function),
6. correlation matrix for the numerical variables (`vis_cor` function).

Each of these functions returns a single **ggplot2** (Wickham, 2016) plot that shows a rectangular representation of the dataset where the expected information is denoted by colors. An example of this visualization[18] can be seen in Figure 13.

The package includes a vignette *Using visdat* that provides examples for all package options. Interestingly, it is the only package that use solely visual means of exploring the data.

The xray package

The **xray** (Seibelt, 2017) package has three functions for the analysis of data prior to statistical modeling:

1. detecting anomalies: missing data, zero values, blank strings, and infinite numbers (`anomalies` function),
2. drawing and printing univariate distributions of each variable through histograms, bar plots and quantile tables (`distributions` function),
3. drawing plots of variables over time for a specified time variable (`timebased` function).

Examples are presented in the readme file in the GitHub repository of the project (`https://github.com/sicarul/xray`), but no vignette is attached to it. Plots[19] generated by the package are presented in Figure 14.

Other packages

As mentioned before, there are numerous R packages that aim to make data exploration faster or the outputs more polished.

For table summaries of data that often include statistical tests, there are a few packages worth mentioning. The package **tableone** (Yoshida and Bohn., 2018) provides a `CreateTableOne` function to

[18] Access the plot object with `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/3cfd")`
[19] Access the associated table with `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/a3a3")`

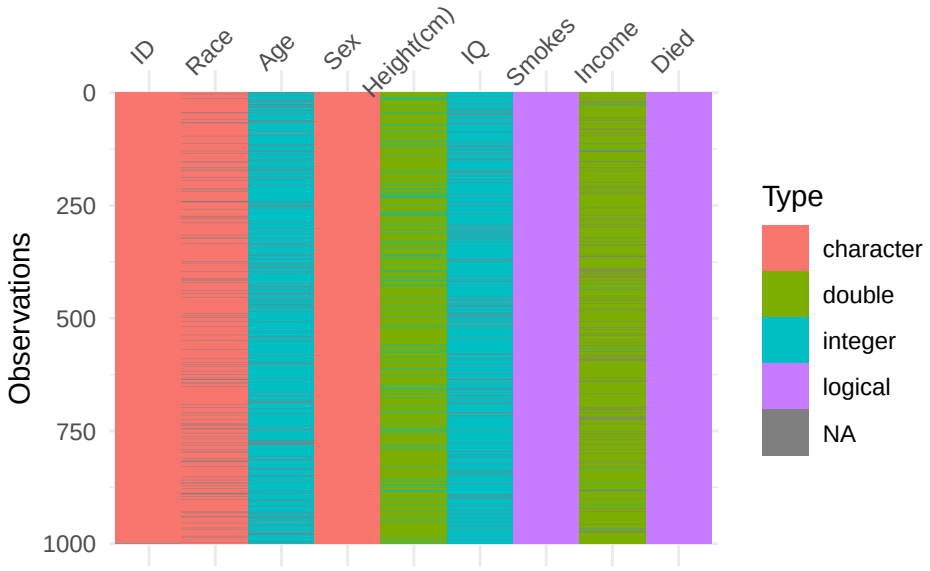


Figure 13: Example output of the `visdat::vis_guess` function (**visdat** v. 0.5.3), which displays types of each value in the data frame and the missing values. We can see that the `Age` variable consists of integer values, even though it is coded as a `character`.

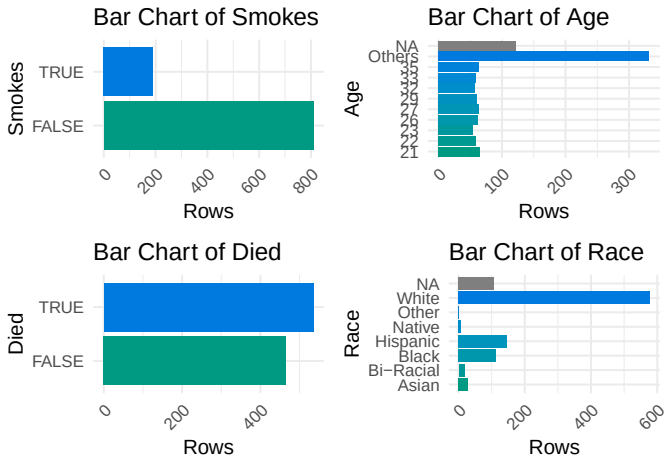


Figure 14: Example output from the `xray::distributions` function (**xray** v. 0.2). Such plots are created for each variable in the dataset along with a table of descriptive statistics.

make publication-ready tables referred to as *Table 1* - traditional name of tables that describe patients' characteristics, usually stratified and including p-values from significance tests. The `describe` function from **describer** package (Hendricks, 2015) prints a summary of a `data.frame` or a vector which includes data types, counts and descriptive statistics. Similarly, the **skimr** (Quinn et al., 2019) package summarises data frames, vectors and matrices. It can also handled grouped data frames. The summary consist of data dimensions, missing and complete value counts, typical descriptive statistics and simple histograms. A function of the same name from **prettyR** (Lemon and Grosjean, 2018) returns descriptive statistics for each column in a `data.frame`. This package is focused on improving the aesthetics of R statistical outputs. Similarly, the package **Hmisc** (Harrell Jr et al., 2019) includes a `describe` function that displays typical descriptive statistics and number of unique and missing values for each column. The `plot` method called on the result of the `describe` function returns a dot plot for each categorical and a spike histogram for each continuous column. The scope of this package is bigger than just Exploratory Data Analysis, as it includes many tools related to regression models.

There are also many packages related to data visualization. Two of them are particularly worth mentioning. The **ggfortify** package (Tang et al., 2016) serves as a uniform interface to plots of different statistical objects, including PCA results that can be used for data exploration and time series plots. The **autoplotly** library (Tang, 2018) was built on top of **ggfortify** to provide automatically generated, interactive visualizations of many statistical models. While these two packages are focused on statistical modeling, they can be helpful in exploratory analysis and exemplify the potential of quick and interactive visualization in R.

Two more packages are relevant to our interest. **gpairs** (Emerson and Green, 2014) and **GGally** (Schloerke et al., 2018) packages implement the generalized pairs plot (Emerson et al., 2013). This type of plot extends well known scatter plot matrices, that visualize bivariate relationships for many variables, by handling both numerical and categorical variables. It is helpful in data exploration and shares similarities to *walls of histograms* that can be found in automated EDA libraries.

Feature comparison

In this section, we compare how different packages address autoEDA tasks as described in Section The tasks of Exploratory Data Analysis. A quick overview of the functionalities of different packages can be found in Table 4.

Data description

Almost all packages contain functions for summarizing datasets. Tools that support data validity analysis are less common.

Whole dataset summaries

Most packages that provide a whole dataset summary take a similar approach and present names and types of variables, number of missing values and sometimes unique values or other statistics. This is true for **summarytools** (`dfSummary` function), **autoEDA** (`dataOverview` function), **dataMaid** (`makeDataReport` result), **funModeling** (`df_status` function), **explore** (`describe` function), **ExPanDaR** (`prepare_descriptive_table` function), and **DataExplorer** (`introduce` function). These outputs are sometimes mixed with univariate summaries. That is the case for one of the most popular summary-type functions: the `dfSummary` functions from the **summarytools** package. An example is given in Figure 15.

In the **dlookr** package, summaries for numerical variables and categorical variables are only presented separately in the report (`describe` function).

The **visdat** package introduces the most original summaries of full dataset. The drawback of this approach is that it is not well suited for high dimensional data. But for a smaller number of variables, it gives a good overview of the dataset.

Data validity

Some packages can perform automated checks for the data, including at least outlier detection. The **dataMaid** package's main purpose is to find inconsistencies and errors in the data. It finds possible outliers, missing values, low-frequency and possibly miscoded factor levels. All this information can be summarised in a quality report. The **dlookr** package covers similar functionality. There are two main differences: the report does not describe possibly miscoded factors, but outlier analysis is

Data Frame Summary

Dimensions: 1000 x 2
Duplicates: 93

No	Variable	Stats / Values	Freqs (% of Valid)	Graph	Valid	Missing
1	Race [factor]	1. White	579 (64.8%)	IIIIIIIIIII	893	107
		2. Hispanic	146 (16.4%)	III	(89.3%)	(10.7%)
		3. Black	114 (12.8%)	II		
		4. Asian	28 (3.1%)			
		5. Bi-Racial	18 (2.0%)			
		6. Native	7 (0.8%)			
		7. Other	1 (0.1%)			
		8. Hawaiian	0 (0.0%)			
2	Income [numeric]	Mean (sd) : 40589.8 (27221.7)	900 distinct values	:	900	100
		min < med < max:		: : :	(90%)	(10%)
		592.1 < 35656.5 < 167203.3		: : :		
		IQR (CV) : 34818.5 (0.7)		: : : :		
				: : : : :		

Figure 15: An example of whole data frame description that includes univariate summary and simple graphics. Created with the `dfSummary function` (**summarytools** package v. 0.9.3).

supplemented with plots showing variable distribution before and after removing the outliers. In all cases, the analysis is rather simple, for example in zero-inflated variables non-zero values are treated as outliers (**dlookr**). The **ExPanDaR** packages handles outliers by providing function that calculate winsorized or trimmed mean. Other packages only provide information about the number of missing values/outliers and identify columns that consist of a single value.

Data exploration

While multivariate analysis is rarely supported, there are many tools for descriptive and graphical exploration of uni- and bivariate patterns in the data.

Univariate statistics

All the tools that support univariate analysis take a similar approach to univariate analysis. For categorical variables, counts are reported and bar plots are presented, while histogram or boxplots and typical descriptive statistics (including quantiles, sometimes skewness) are used for continuous variables.

In **dataMaid** and **dlookr** packages, these plots are presented variable-by-variable in the report. In other packages (**DataExplorer**, **funModeling**, **SmartEDA**, **inspectdf**) groups of plots of the same type are shown together - as a wall of histograms or bar plots. Similarly, the **explore** package present all the plots at once. The **ExPanDaR** package allows user to choose variables to display in a **shiny** applications. Notably, **dlookr** reports skewness of variables and in case a skewed variable is found, it shows the distribution after some candidate transformations to reduce the skewness have been applied. This library also reports normality. The **SmartEDA** package also reports skewness and displays QQ plots against normal distribution, but it does not provide any means of reducing skewness.

Bivariate statistics

The **funModeling** and **SmartEDA** packages only support calculating correlations between variables and a specified target. **DataExplorer** and **visdat** packages can plot correlation matrices. They differ in categorical variables treatment. Some packages require only numerical features (**visdat**). Interestingly, in **DataExplorer**[20], low-cardinality categorical features are converted to 0-1 variables and plotted alongside numerical variables, as seen in Figure 16.

The **arsenal** package only presents variable summaries by levels of a chosen categorical variable. The report from the **autoEDA** package consists of a limited number of bar plots/boxplots with target variable as one of the dimensions. Similarly, in **DataExplorer**, **dlookr**, **funModeling** and **SmartEDA**, scatter plots and box plots or histograms with a specified target variable on one of the axis can be plotted. Additionally, **funModeling** and **dlookr** draw histograms/densities of continuous features by the target. In **shiny** applications provided by **ExPanDaR** and **explore** packages, the user can choose target variables and explanatory variables to display bivariate plots. Interestingly, scatter plots provided by the **ExPanDaR** package can be extended to display multivariate dependencies

[20] Access the plot with `archivist::aread("mstaniak/autoEDA-resources/autoEDA-paper/0526")`

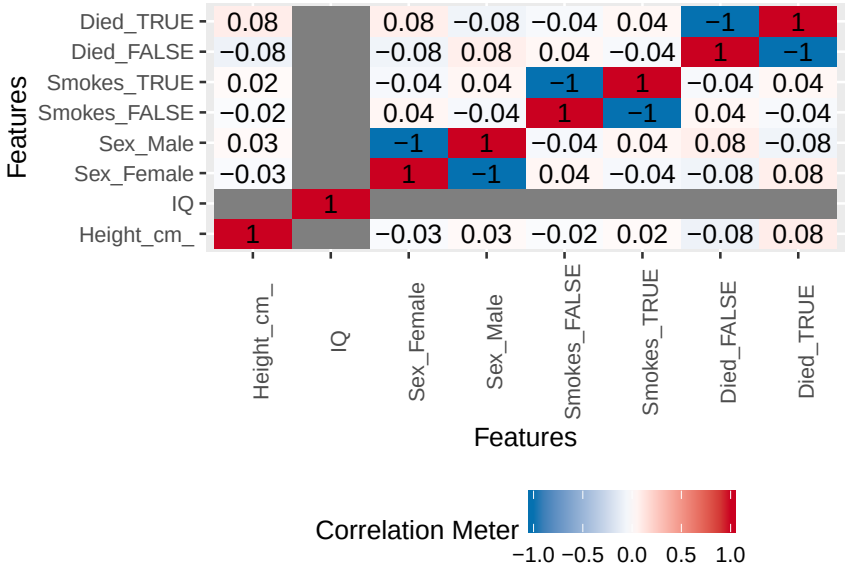


Figure 16: Correlation plot as returned by the `DataExplorer::plot_correlation` function.

by mapping variables to size and color of the points. The **funModeling** package also has unique options: drawing bar plots of discretized variables by the target and quantitative analysis for binary outcome based on representativeness and accuracy. **arsenal**, **summarytools** and **SmartEDA** also feature contingency tables. Moreover, **exploreR** and **ExPanDaR** packages use linear regression plots and statistics to find relationships between the target and other variables. The **explore** package can only handle binary targets, but it allows user to fit and plot a decision tree model.

Data cleaning and data transformation

The **dataMaid** package assumes that every decision regarding the data should be made by the analyst and does not provide any tools for data manipulation after diagnosis. Most of the packages only provide exploration tools. Exceptions are **dlookr**, **funModeling**, **DataExplorer** and **exploreR**. **DataExplorer** provides tools for normalization, imputation by a constant, merging levels of factors, creating dummy variables and transforming columns.

The **dlookr** package can create a report that presents different possible transformations of features. Missing values can be imputed by mean/median/mode and distributions of variables before and after the procedure can be compared. The same is done for imputation of outliers. Logarithmic and root square transforms are proposed for skewed variables. Different methods of binning continuous variables are also presented, including Weight of the Evidence.

The **funModeling** package can perform discretization of a variable using an equal frequency criterion or gain ratio maximization. It can also scale variables to the interval $[0, 1]$. Outliers can be treated using the Tukey or Hampel method.

Reporting

DataExplorer, **dlookr**, **dataMaid**, **SmartEDA**, **explore** and **RtutoR** have an option of generating a report and saving it to a file. Such a report usually consists of all or most possible outputs of the package. The plots and summaries are organized by the exploration task (for example univariate, then bivariate analysis) and either simply variable-by-variable (**dataMaid**, **dlookr**) or grouped by variable type (**DataExplorer**, **SmartEDA**). The **autoEDA** package generates a minimal report with bivariate plots. Packages **arsenal**, **funModeling**, **xray**, **summarytools** and **exploreR** have an option of saving outputs - plots or tables - to files.

Discussion

Automated EDA can be either directed towards a general understanding of a particular dataset or be more model-oriented, serving as a foundation for good modeling. While presented packages include

some tools related to simple variable transformations, they are more focused on data understanding. For this task, they have many advantages. In this section, we summarize the strong points of existing tools and point out some possible improvements and new directions for autoEDA.

Strengths of autoEDA packages

1. The packages **dlookr**, **dataMaid**, **DataExplorer**, **SmartEDA** are capable of creating good quality reports.
2. **DataExplorer** has very good visualizations for PCA.
3. **DataExplorer** handles categorical variables on correlation plots by creating dummy features, which is a unique idea compared to other packages.
4. The **visdat** package, while probably not the best choice for high dimensional data, features interesting take on initial whole dataset exploration.
5. The **dlookr** package is capable of selecting skewed variables and proposing transformations. Some of the other packages display binned continuous variables, which can also help in seeing visualizing dependencies.
6. **dataMaid** is a good tool for finding problems in the data. Thanks to the structure of `check` and `summarize` functions results, discovered issues can be treated effectively.
7. For datasets with a moderate number of features, **DataExplorer**, **funModeling**, **dlookr** and **SmartEDA** give a reasonable insight into variables distributions and simple relationships.
8. **SmartEDA** package provides a method of visualizing multivariate relationships - parallel coordinate plot.
9. The **exploreR** package provides usefuls tool for assessing bivariate relationship through linear regression.

We can see that tasks related to data quality and whole dataset summary are well by the existing libraries. Getting the big picture of the data and finding possible data quality problems is easy, especially with the **dataMaid** package. For classical applications, for example, statistical analyses in medicine, the current tools provide very good tables, such as the ones from **tableone** or **arsenal** packages, and uni-/bivariate plots. The **inspectdf** and **summarytools** packages can also provide quick insights into a dataset. Univariate analysis can be performed either variable-after-variable (**dlookr**, **dataMaid**), where we can see the statistical properties of each variable, or as groups of plots based on variable type (**DataExplorer**, **funModeling**). Both ways can be useful for a reasonable number of predictors. While multivariate tools are scarce, the available tools, PCA in **DataExplorer** and PCP in **SmartEDA**, are very well done. Notably, the **ExPanDaR** package provides very high flexibility thanks to the possibility of interactively choosing variables to display, adding new variables on-the-fly and customizing plots in the **shiny** application.

Future directions and possible improvements

The field of autoEDA is growing. New packages are being developed rapidly - there are recent additions from April and May. Features are added to existing packages and bugs are corrected, as new issues are suggested by users on GitHub. At this moment, we can identify the following problems and challenges.

All the presented tools can fail in situations with imperfect data. In particular, they are usually not robust to issues like zero-variance/constant variables. Such problems are expected to be solved in the nearest future, as suggested for example by issues in the GitHub repo of the **DataExplorer** package. In general, error messages can be uninformative. Moreover, in some situations, they lack flexibility. For example, in **DataExplorer** arguments can be passed to `cor` function, but not to `corrplot` function.

In case of *walls of histograms* (or bar plots), no selection is being done and no specific order is chosen to promote most interesting distributions. The same is true for automatically created reports. This problem is only addressed by the **RtutoR** package, which allows to select top `k` relevant variables. Moreover, for high-dimensional data or high-cardinality factors, the plots often become unreadable or impractical. Partial solutions to this problems are applied, for example **DataExplorer** removes too large factors from the panels. More generally, many GitHub issues for the described packages are related to customizing and improving plots and output tables. It is a challenging task due to the diversity of possible input data and a major concern for developers of autoEDA packages.

Typical EDA tasks are limited to exploring bivariate relationships. Searching for higher dimensional dependencies would be interesting, for example by adding color and size dimensions to the plots, which was already done in the **ExPanDaR** package. For *wall of plots* type of display, such an

addition would result in a large number of new plots. Thus, it would require a proper method of finding the most relevant visualization. Interactivity partially helps address this issue. PCA, parallel coordinate plots and model summaries are supported, but each by a separate package. It is evident that there is a shortage of multivariate tools. Univariate regression models can be plotted by the **exploreR** package. The **explore** package plots decision trees for binary target variables. In other cases, exploration based on simple statistical models (such as scatter plot smoothing) is not an option. Using regression models and feature transformations to identify and measure relevant relationships could improve bivariate or multivariate analyses supported by automated EDA.

Regarding variable transformation, only one of the packages addresses the issue of skewed variables. Proposing transformations of continuous features other than binning would be helpful and could improve visualizations, for example, scatter plots with skewed variables. Missing data imputation more advanced than imputing a constant is delegated to other packages, although, it is known that imputation by a constant is usually not the best method of missing values treatment. Some of the above issues limit the packages' usefulness in iterative work. Though, the comparisons of transform and original features and the possibility of applying discovered transformations to data in **dlookr** package are steps in the right direction.

Support for time-varying variables and non-classical (not IID) problems such as survival analysis is limited or non-existent. For survival analysis, the automation level is low, but there are two notable tools for summarizing dependencies. First is the recognized package **survminer** (Kassambara and Kosinski, 2018), which helps visualize survival curves, while also displaying survival tables and other information. The other tool is the **cr17** package (Młynarczyk and Biecek, 2017), which includes `summarizeCR` function that returns several tables and plots for competing risks analysis. More tools for fast visualization of at least bivariate relationships in such problems would be a big help for analysts. Cluster analysis is sometimes regarded a part of the EDA process, but it is not available in any of the packages.

The tools available in R have similar range to other languages' libraries, for example from Python. Python packages such as Dora (Epstein, 2017) or lens (Zabalza and Engineers, 2018) also cover feature-by-feature descriptive statistics and plots, bivariate visualizations of the relationships between predictors and target variable, contingency tables, basic data transformations, and imputation. Tools for visual data exploration supports also tools for visual model exploration like **DALEX** (Biecek, 2018) or **iml** (Casalicchio et al., 2018). In both cases visual summaries help to quickly grasp key relations between variables or between input features and model predictions.

Since EDA is both closely connected to feature engineering and based on visual insights, automated EDA can draw from existing tools for automated feature extraction like SAFE ML (Gosiewska et al., 2019) or TPOT (Olson et al., 2016) and visualization recommendations. When it comes to aiding visual exploration of a dataset, standalone software carries possibilities beyond what we can expect from R packages or analogous libraries in other languages. A recent notable example is DIVE (Hu et al., 2018). It is an example of a growing number of tools for visual data exploration that aim to distinguish between relevant and irrelevant visualization and help the analyst find the most interesting plots. DIVE is one of the *mixed-initiative visualization systems*, meaning it uses both statistical properties of the dataset and user interactions to find the relevant plots. Building recommendation systems into autoEDA tools can help address the issue of dealing with high-dimensional data and multivariate dependencies by letting the ML-based system deal with the complexity of a large number of candidate visualizations. AI-assisted data exploration can be even faster and more efficient.

As autoEDA tools are still maturing, the efforts in the field are somewhat fragmented. Many packages try to achieve similar goals, but they can be quite inconsistent. It is especially visible in the multiplicity of names for the `summary`-type function to describe a whole data frame. As the libraries develop, new standards and conventions should be proposed.

Acknowledgement

This work was financially supported by the NCN Opus grant 2016/21/B/ST6/02176.

Bibliography

P. Biecek. DALEX: Explainers for Complex Predictive Models in R. *Journal of Machine Learning Research*, 19(84):1–5, 2018. URL http://jmlr.org/papers/v19/18-416.html. [p365]

P. Biecek. Model Development Process. *arXiv e-prints*, 2019. URL https://arxiv.org/abs/1907.04461. [p348]

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. [p349]

B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. *mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions*, 2017. [p347]

D. H. Caro and P. Biecek. intsvy: An R package for analyzing international large-scale assessment data. *Journal of Statistical Software*, 81(7):1–44, 2017. URL https://doi.org/10.18637/jss.v081.i07. [p347]

G. Casalicchio, C. Molnar, and B. Bischl. Visualizing the feature importance for black box models, 2018. URL https://arxiv.org/abs/1804.06620 [p365]

P. Casas. Data Science Live Book. https://livebook.datascienceheroes.com/, 2018. Retrieved on 14 March 2019. [p356]

P. Casas. *funModeling: Exploratory Data Analysis and Data Preparation Tool-Box Book*, 2019. URL https://CRAN.R-project.org/package=funModeling. R package version 1.7. [p356]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *Shiny: Web Application Framework for R*, 2019. URL https://CRAN.R-project.org/package=shiny. R package version 1.3.2. [p354, 358]

M. Coates. *exploreR: Tools for Quickly Exploring Data*, 2016. URL https://CRAN.R-project.org/package=exploreR. R package version 0.1. [p355]

D. Comtois. *Summarytools: Tools to Quickly and Neatly Summarize Data*, 2019. URL https://CRAN.R-project.org/package=summarytools. R package version 0.9.2. [p358]

D. Cook. *Practical Machine Learning with H2O: Powerful, Scalable Techniques for Deep Learning and AI*. O'Reilly Media, 2016. [p347]

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. [p348]

B. Cui. *DataExplorer: Automate Data Exploration and Treatment*, 2019. URL https://CRAN.R-project.org/package=DataExplorer. R package version 0.8.0. [p350]

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. [p349]

M. Dowle and A. Srinivasan. *Data.table: Extension of 'data.frame'*, 2019. URL https://CRAN.R-project.org/package=data.table. R package version 1.12.0. [p358]

J. W. Emerson and W. A. Green. *Gpairs: Gpairs: The Generalized Pairs Plot*, 2014. URL https://CRAN.R-project.org/package=gpairs. R package version 1.2. [p361]

J. W. Emerson, W. A. Green, B. Schloerke, J. Crowley, D. Cook, H. Hofmann, and H. Wickham. The generalized pairs plot. *Journal of Computational and Graphical Statistics*, 22(1):79–91, 2013. URL https://doi.org/10.1080/10618600.2012.694762. [p361]

N. Epstein. Dora: Exploratory data analysis toolkit for python, 2017. URL https://github.com/NathanEpstein/Dora. Python library version 0.0.2. [p365]

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015. [p347]

J. Gassen. *ExPanDaR: Explore Panel Data Interactively*, 2018. URL https://CRAN.R-project.org/package=ExPanDaR. R package version 0.3.0. [p352]

A. Gosiewska, A. Gacek, P. Lubon, and P. Biecek. SAFE ML: Surrogate Assisted Feature Extraction for Model Learning. *arXiv e-prints*, 2019. URL https://arxiv.org/abs/1902.11035. [p365]

G. Grolemund and H. Wickham. *R for Data Science*, 2019. URL https://r4ds.had.co.nz/. [p348]

H2O.ai. *H2O*, 2019. version 3.22.1.6. [p347]

F. E. Harrell Jr, with contributions from Charles Dupont, and many others. *Hmisc: Harrell Miscellaneous*, 2019. URL https://CRAN.R-project.org/package=Hmisc. R package version 4.2-0. [p361]

E. Heinzen, J. Sinnwell, E. Atkinson, T. Gunderson, and G. Dougherty. *Arsenal: An Arsenal of 'R' Functions for Large-Scale Statistical Summaries*, 2019. URL https://CRAN.R-project.org/package=arsenal. R package version 2.0.0. [p349]

P. Hendricks. *Describer: Describe Data in R Using Common Descriptive Statistics*, 2015. URL https://CRAN.R-project.org/package=describer. R package version 0.2.0. [p361]

X. Horn. *autoEDA: Automated Univariate and Bivariate Exploratory Data Analysis*, 2018a. R package version 1.0. [p349]

X. Horn. Automated exploratory data analysis in r. https://www.linkedin.com/pulse/automated-exploratory-data-analysis-r-xander-horn/, 2018b. Retrieved on 14 March 2019. [p350]

K. Hu, D. Orghian, and C. Hidalgo. Dive: A mixed-initiative system supporting integrated data exploration workflows. In *ACM SIGMOD Workshop on Human-In-the-Loop Data Analytics (HILDA)*. ACM, 2018. URL https://doi.org/10.1145/3209900.3209910. [p365]

H. Jin, Q. Song, and X. Hu. Auto-Keras: Efficient Neural Architecture Search with Network Morphism. *arXiv e-prints*, 2018. [p347]

A. Kassambara and M. Kosinski. *Survminer: Drawing Survival Curves Using 'ggplot2'*, 2018. URL https://CRAN.R-project.org/package=survminer. R package version 0.4.3. [p365]

L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18 (1):826–830, 2017. [p347]

R. Krasser. *Explore: Simplifies Exploratory Data Analysis*, 2019. URL https://CRAN.R-project.org/package=explore. R package version 0.4.3. [p354]

M. Kuhn and D. Vaughan. *parsnip: A Common API to Modeling and Analysis Functions*, 2019. URL https://CRAN.R-project.org/package=parsnip. R package version 0.0.2. [p347]

J. Lemon and P. Grosjean. *prettyR: Pretty Descriptive Stats*, 2018. URL https://CRAN.R-project.org/package=prettyR. R package version 2.2-2. [p361]

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. [p]

M. Młynarczyk and P. Biecek. *Cr17: Testing Differences Between Competing Risks Models and Their Visualisations*, 2017. URL https://CRAN.R-project.org/package=cr17. R package version 0.1.0. [p365]

A. Nair. *RtutoR: Shiny Apps for Plotting and Exploratory Analysis*, 2018a. URL https://CRAN.R-project.org/package=RtutoR. R package version 1.2. [p357]

A. Nair. Automating basic eda. https://www.r-bloggers.com/automating-basic-eda/, 2018b. Retrieved on 25 March 2019. [p357]

V. Nijs. *Radiant: Business Analytics Using R and Shiny*, 2019. URL https://CRAN.R-project.org/package=radiant. R package version 0.9.9.1. [p347]

R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pages 123–137. Springer-Verlag, 2016. URL https://doi.org/10.1007/978-3-319-31204-0_9. [p347, 365]

A. H. Petersen and C. T. Ekstrom. *dataMaid: A Suite of Checks for Identification of Potential Errors in a Data Frame as Part of the Data Screening Process*, 2018. URL https://CRAN.R-project.org/package=dataMaid. R package version 1.2.0. [p351]

S. Putatunda, K. Rama, D. Ubrangala, and R. Kondapalli. SmartEDA: An R Package for Automated Exploratory Data Analysis. *arXiv e-prints*, art. arXiv:1903.04754, 2019. [p347, 358]

M. Quinn, A. McNamara, E. Arino de la Rubia, H. Zhu, and S. Ellis. *Skimr: Compact and Flexible Summaries of Data*, 2019. URL https://CRAN.R-project.org/package=skimr. R package version 1.0.7. [p361]

A. Rushworth. *Inspectdf: Inspection, Comparison and Visualisation of Data Frames*, 2019. URL https://CRAN.R-project.org/package=inspectdf. R package version 0.0.3. [p356]

C. Ryu. *Dlookr: Tools for Data Diagnosis, Exploration, Transformation*, 2019. URL https://CRAN.R-project.org/package=dlookr. R package version 0.3.8. [p352]

B. Schloerke, J. Crowley, D. Cook, F. Briatte, M. Marbach, E. Thoen, A. Elberg, and J. Larmarange. *GGally: Extension to 'ggplot2'*, 2018. URL https://CRAN.R-project.org/package=GGally. R package version 1.4.0. [p361]

P. Seibelt. *Xray: X Ray Vision on Your Datasets*, 2017. URL https://CRAN.R-project.org/package=xray. R package version 0.2. [p359]

Y. Tang. Autoplotly: An r package for automatic generation of interactive visualizations for statistical results. *Journal of Open Source Software*, 3, 2018. URL https://doi.org/10.21105/joss.00657. [p361]

Y. Tang, M. Horikoshi, and W. Li. ggfortify: Unified Interface to Visualize Statistical Results of Popular R Packages. *The R Journal*, 8(2):474–485, 2016. URL https://doi.org/10.32614/rj-2016-060. [p361]

N. Tierney. Visdat: Visualising whole data frames. *JOSS*, 2(16):355, 2017. URL https://doi.org/10.21105/joss.00355. [p359]

D. Ubrangala, K. Rama, and R. Kondapalli. *SmartEDA: Summarize and Explore the Data*, 2018. URL https://CRAN.R-project.org/package=SmartEDA. R package version 0.3.0. [p357]

H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2016. ISBN 978-3-319-24277-4. URL https://doi.org/10.1007/978-0-387-98141-3. [p359]

R. Wirth. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pages 29–39, 2000. [p348, 350]

Y. Xie. *Dynamic Documents with R and Knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL https://doi.org/10.18637/jss.v056.b02. [p358]

K. Yoshida and J. Bohn. *Tableone: Create 'Table 1' to Describe Baseline Characteristics*, 2018. URL https://github.com/kaz-yos/tableone. R package version 0.9.3. [p359]

V. Zabalza and F. Engineers. lens, 2018. URL https://doi.org/10.5281/zenodo.2593337. Python library version 0.4.5. [p365]

J. J. Zhang and K. B. Storey. RBioplot: An Easy-to-Use R Pipeline for Automated Statistical Analysis and Data Visualization in Molecular Biology and Biochemistry. *PeerJ*, 2016(9), 2016. URL https://doi.org/10.7717/peerj.2436. [p347]

Mateusz Staniak
Faculty of Mathematics and Information Science
Warsaw University of Technology
Poland
mtst@mstaniak.pl

Przemysław Biecek
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
Poland
Samsung R&D Institute Poland (SRPOL)
ORCiD: 0000-0001-8423-1823
przemyslaw.biecek@gmail.com

Task type	Task	a	aE	DE	dM	d	EPD	e	eR	fM	i	R	SE	s	v	x
Dataset	Variable types		x	x	x	x		x		x	x		x	x	x	
	Dimensions		x	x	x	x	x			x	x		x		x	
	Other info			x							x				x	
	Compare datasets	x								x	x				x	
Validity	Missing values		x	x	x	x	x	x		x	x		x	x	x	x
	Redundant col.		x		x	x		x		x			x	x	x	
	Outliers		x		x	x	x			x						
	Atypical values				x					x						x
	Level encoding				x											
Univar.	Descriptive stat.		x		x	x	x	x		x	x	x	x	x		x
	Histograms		x	x	x	x	x	x		x	x	x	x	x		
	Other dist. plots			x			x									
	Bar plots		x	x	x	x	x	x		x	x	x	x	x		
	QQ plots			x		x							x			
Bivar.	Descriptive stat.	x				x		x				x	x	x		
	Correlation matrix			x		x	x				x				x	
	1 vs each corr.		x						x	x			x			
	Time-dependency	x					x									x
	Bar plots by target		x	x		x	x	x		x		x	x			
	Num. plots by target		x			x	x	x		x			x			
	Scatter plots			x			x		x			x	x			
	Contigency tables	x				x							x	x		
	Other stats. (factor)									x	x		x			
Multivar.	PCA			x												
	Stat. models	x					x	x								
	PCP												x			
Transform.	Imputation			x		x		x								
	Scaling					x			x	x						
	Skewness					x										
	Outlier treatment					x	x	x		x						
	Binning			x		x				x						
	Merging levels			x						x						
Reporting	Reports		x	x	x	x		x				x	x			
	Saving outputs	x							x	x				x		x

Table 4: Overview of functionalities of all described packages. Package names were shortened to make the table as compact as possible. **a** denotes **arsenal**, **aE** - **autoEDA**, **DE** - **DataExplorer**, **dM** - **dataMaid**, **d** - **dlookr**, **EPD** - **ExPanDaR**, **e** - **explore**, **eR** - **exploreR**, **fM** - **funModeling**, **i** - **inspectdf**, **R** - **RtutoR**, **SE** - **SmartEDA**, **s** - **summarytools**, **v** - **visdat**, **x** denotes **xray**. *Num. plots by target* refers to either histogram, density, violin or box plot.

HCmodelSets: An R Package for Specifying Sets of Well-fitting Models in High Dimensions

by Henrique Hoeltgebaum and Heather Battey

Abstract In the context of regression with a large number of explanatory variables, Cox and Battey (2017) emphasize that if there are alternative reasonable explanations of the data that are statistically indistinguishable, one should aim to specify as many of these explanations as is feasible. The standard practice, by contrast, is to report a single effective model for prediction. This paper illustrates the R implementation of the new ideas in the package **HCmodelSets**, using simple reproducible examples and real data. Results of some simulation experiments are also reported.

Introduction

In a recent paper Cox and Battey (2017) outline a procedure for regression analysis when there are more explanatory features than study individuals, a situation that arises particularly in genomics. Their emphasis is on understanding the true data-generating mechanism rather than prediction. The distinction is important. For prediction there may be several models that are essentially equally effective and any choice between them is rather arbitrary. On the other hand, since different well-fitting models typically have different subject-matter implications, it is insufficient, and often misleading, to report an arbitrary one. Even if the immediate goal is prediction, a causal explanation is likely to produce more stable and more generalizable predictions. A key message of Cox and Battey (2017) is that if there are several models that fit the data essentially equally well, one should aim to specify as many as is feasible. This view is in contraposition to that implicit in the use of the lasso (Tibshirani, 1996) and other variable selection methods, which produce a single model effective for prediction.

The methods of Cox and Battey (2017) are summarized in Section Methodology. Software implementing these ideas in R has been written by Hoeltgebaum (2018) and is available in the **HCmodelSets** package. The software supports most widely used models of dependency including the linear model, the linear logistic model for binary data (Cox, 1958), and the proportional hazards model fitted by partial likelihood (Cox, 1972, 1975b). The present article aims to provide a detailed guide to usage based on simple examples.

Methodology

Suppose that data are available on n units, for each of which an outcome y is observed along with a vector x of d potential explanatory variables, where d is much larger than n. For progress an assumption of sparsity is needed, and the most explicit and interpretable such assumption is that relatively few of the potential explanatory features have a real effect, an assumption central to the formulation of the lasso and similar penalized regression procedures.

Cox and Battey (2017) suggest a different approach whose aim is essentially a confidence set of models. There are three stages to a discussion, and conditionally on the first two, the resulting set of models can be made to have the formal statistical properties associated with confidence sets.

In the first stage, an initial reduction is made in which a large number of variables are discarded on the basis that they have no explanatory power, or that any explanatory power that they appear to have is explained away by other variables. The assessment is made by fitting a suitable low-dimensional regression model several times to each variable, each time alongside a different set of k companion variables. A variable is retained for further study if it satisfies a particular criterion in at least half of the analyses in which it appears. The sets of variables to be considered together are specified by a partially balanced incomplete block design (Yates, 1936) in which variable indices are arranged in a hypercube of appropriate dimension. This initial dimension is determined by d and a constraint on k to mitigate the effect of dependence between p-values, or the associated test statistics, in any single analysis. Ideally k will be between 10 and 15; see §7 of Battey and Cox (2018) for a discussion of this choice. Successive reductions are made using arrangements in successively lower-dimensional hypercubes, where the criterion for retaining variables in each stage is guided by the theoretical discussion of Battey and Cox (2018), the need to keep the number of rows, columns, etc., of successive hypercubes ideally ≤ 15, and the requirement for a degree of stability over rerandomization of the variable indices in the hypercube. Thus, judgement is required at various stages.

On the resulting set of variables, of which there will be roughly 10-20 by construction, an exploratory analysis is performed, of the kind that is standard in much statistical work. For instance, inspection of interaction plots or probability plots of t statistics. The objective is to detect possible nonlinearities or outliers.

All variables retained through the reduction phase and any squared or interaction terms suggested at the exploratory phase comprise the so called comprehensive model. All low dimensional subsets of the comprehensive model are then tested for their compatibility with the data using a likelihood ratio test, and all models that pass this test are reported. If, among this set, there are models that contain interaction terms without the corresponding main effects, the main effects are added.

For the resulting sets of models to have the formal statistical properties associated with confidence sets, conditional on the first two phases, it is necessary to either split the sample, see Cox (1975a) for a discussion, or to adjust standard tests of model adequacy in account of the alternative hypothesis being selected in the light of the data.

Illustration of usage: a simple reproducible example

Some simple data generating processes

We illustrate the functions available in **HCmodelSets**, and their appropriate usage, using simple examples. These functions include `DGP`, which can be used to reproduce the simulation study of Battey and Cox (2018) and to explore further sensitivities.

```
library(HCmodelSets)
dgp = DGP(s=5, a=3, sigStrength=1, rho=0.9, n=100,
                intercept=5, noise=1, var=1, d=1000, DGP.seed = 2018)
```

This generates normally distributed responses as $Y_i = \mu + x_i^T\beta + \varepsilon_i$ $(i = 1, \ldots, n)$ where, in the present example `n = 100`, μ `= intercept = 5`, the ε_i are independently standard normally distributed and the x_i are realizations of a `d = 1000` dimensional normal random vector of mean zero and covariance matrix Σ. The vector of regression coefficients β is sparse in the sense that only `s = 5` entries are non-zero, equal to `sigStrength = 1`, and Σ is such that a correlation `rho = 0.9` is induced between the corresponding entries of x_i, the so called signal variables, and among `a = 3` of the remaining variables. All potential explanatory variables have variance `var = 1`. The results of the subsequent analysis can be reproduced by setting `DGP.seed = 2018` as above, but this argument is not needed.

With the appropriate modification to its arguments, the `DGP` function also generates survival times from a proportional hazards model with Weibull baseline hazard. The hazard function for the ith individual is

$$h_i(t;\beta) = \kappa\tau(\tau t)^{\kappa-1}\exp\{x_i^T\beta\},$$

where $h(t) = \kappa\tau(\tau t)^{\kappa-1}$ is the Weibull hazard function. From this, the density and distribution functions of survival times conditional on x_i are obtained as

$$\begin{aligned} f_i(t;\beta) &= \kappa\tau(\tau t)^{\kappa-1}\exp\{x_i^T\beta\}\exp\{-e^{x_i^T\beta}(\tau t)^{\kappa}\}, \\ F_i(t;\beta) &= \exp\{-e^{x_i^T\beta}(\tau t)^{\kappa}\}. \end{aligned}$$

Thus, given covariates x_i, uncensored survival times from the above proportional hazards model are generated as $T_i = \{-\log U/(\tau^\kappa e^{x_i^T\beta})\}^{1/\kappa}$, where U is a uniform random variable on $(0,1)$.

In the section entitled Illustration of performance in some idealized settings a minor modification to the previous code is given to generate (potentially censored) survival time data from this model. Simulation results for the procedure fitted to both types of data are also reported in the same section.

Reduction phase

Based on the previous output, typical usage of the function `Reduction.Phase` is:

```
out = Reduction.Phase(X = dgp$X,Y = dgp$Y,
                family = gaussian, seed.HC = 1012)
```

In particular, this arranges the indices of the columns of `dgp$X` in a hypercube of appropriate dimension, and fits a linear regression model to each set of variables indexed by the rows, columns, etc., of the hypercube. Other choices of the argument `family` are illustrated in section entitled Real example. The arrangement of the variable indices in the hypercube is at random. However, `seed.HC = 1012` allows

the results of the analyses reported here to be reproduced. If the argument `dmHC` is left unspecified (the recommendation), as in this example, the dimension of the initial hypercube is set to be the smallest dimension such that the number of rows, columns etc., is no greater than 15. Thus, the present example initially has the 1000 variable indices arranged in a $10 \times 10 \times 10$ cube.

Because the comprehensive model obtained from the full data achieves better fit than an arbitrary model embedding the one to be tested, a test of adequacy of the smaller model rejects too often in hypothetical repeated application. It is therefore usually sensible to split the sample in two and use, say 70%, for the reduction and exploratory phases, and the remaining 30% for construction of the conditional confidence sets of models. The appropriate modification to the previous code, so that only the first 70 observations are used for the reduction phase, is:

```
outSplit1 = Reduction.Phase(X = dgp$X[1:70,],
                Y = dgp$Y[1:70], family = gaussian, seed.HC = 1012)
```

If the initial sample size is rather small and the model to be fitted is non-Gaussian, the sample size available for the final phase of the procedure is likely to be too small for the distribution of the maximum likelihood estimator to be well-approximated by its asymptotic distribution. Correspondingly, the coverage probability of the confidence sets of models conditional on the reduction phase is likely to differ from the nominal value. This could be mitigated through a Bartlett correction to the likelihood ratio statistic, but this has not been implemented in the current version of the package. See Bartlett (1937); Barndorff-Nielsen and Cox (1984) and Barndorff-Nielsen and Cox (1994) (p133, p152–53) for a discussion of the theory of Bartlett correction.

A strong reassurance over the security of one's conclusions is given if the set of retained variables does not alter much upon rerandomization of the arrangement of the variable indices in the (hyper)cube, and this is a suitably cautious check in practice. Indeed, if the answers so obtained differ appreciably, the suggestion is that too severe a reduction has been performed. Thus we consider also the outcome `outSplit2`, obtained when no argument `seed.HC` is provided, so that variable indices are arranged in their original order. Some variables will appear in all or almost all analyses.

```
outSplit2 = Reduction.Phase(X = dgp$X[1:70,],
                Y = dgp$Y[1:70], family = gaussian)
```

The outcomes `outSplit1` and `outSplit2` of the previous two analyses are two lists of variable indices from each successive reduction. Only the latter reductions are of ultimate interest, but the intermediate reductions should be inspected to ensure that the number of variables retained is not so large as to be detrimental to the subsequent stage of the reduction. In the present example, the final lists of variables are arrived at by an implementation of the default decision rules, to some extent guided by the analysis of Battey and Cox (2018). These are to retain variables if they are among the two most significant in at least half the analyses in which they appear in the first stage reduction, and if they are significant at the 1% level in at least half the analysis in which they appear in subsequent reductions. The 1% threshold is arbitrary and judgement should be exercised if the output of such an analysis is unreasonable, for instance if too many variables are retained in any stage of the reduction. This is particularly important when the initial number of variables is very large, so that variables are initially arranged in a four or five dimensional hypercube. The Real example illustrates appropriate use of judgement through the optional argument `vector.signif` of the `Reduction.Phase` function. The objective of the reduction phase is to reduce the number of candidate signal variables to ideally not more than 20, to be subjected to more detailed joint analysis.

The sorted lists of retained variables using the default decision rules and the two initial arrangements of variables indices in the cube are:

```
v1=sort(outSplit1$List.Selection$`Hypercube with dim 2`$numSelected1)
v2=sort(outSplit2$List.Selection$`Hypercube with dim 2`$numSelected1)
v1
#> [1] 46 51 66 156 229 263 272 319 423 496 531 559 735 804 827 897 929 984 1000
v2
#> [1] 46 156 272 291 319 397 531 559 642 827 897 929 984
```

Of these variables, ten are in common, an appreciable overlap. The indices of the five true signal variables are contained in `v1` and `v2` (and their intersection). These are

```
dgp$TRUE.idx
#> [1] 46 531 559 897 929
```

Usage of the other functions in the package is illustrated using the output of the second analysis, i.e., the variables in `v2`.

An alternative to the reduction phase is to use a deliberately undertuned lasso fit. The lasso is typically fitted by the coordinate descent algorithm in general regression settings, or by the least angle regression algorithm in the linear model. Thus, the practical implementation of the lasso is essentially forward selection. By contrast, the reduction phase of Cox and Battey (2017) is a version of backward elimination. Both forward selection and backward elimination are likely to be effective in many cases, although a theoretical elucidation of the conditions on the design matrix to ensure this has not been attempted. If the objective is to obtain a superset of the comprehensive model, as here, backward elimination has advantages in simpler settings. See Illustration of performance in some idealized settings for an empirical comparison in idealized examples.

The lasso, fitted by coordinate descent as implemented in the R package **glmnet**, and undertuned to produce at least the same number of variables as in `v2`, is obtained by

```
library(glmnet)
lasso.fit = glmnet(x = dgp$X[1:70,],y = dgp$Y[1:70])
n.coefs = apply(coef(lasso.fit), 2, function(x) length(which(x!=0)))
idx.coefs = which(n.coefs == length(v2))
if(length(idx.coefs)==0){
        idx.coefs = min(which(n.coefs >= length(v2)))}
lasso.var = which(coef(lasso.fit)[,idx.coefs[1]]!=0)
```

In the present example, the associated variables excluding the intercept, are

```
lasso.var[-1]-1
#> [1] 40 46  161 341 384 511 531  559  827 897  929  984
```

Seven of these are in common with `v1` and `v2`, including the five signal variables.

Exploratory phase

The analysis discussed by Cox and Battey (2017) is intended to be largely exploratory, and a key aspect of the procedure is that it allows informal checks, standard in much statistical work. The function `Exploratory.Phase` automates some, but by no means all, of what would typically take place in an exploratory data analysis, and is provided as a rough guide. Usage of the `silent` argument is illustrated in the Real example section, in which `silent = FALSE` forces a certain degree of judgement to be exercised.

The following code detects potentially important squared or interaction terms among the variables whose indices are stored in `v2`.

```
out.exp.phase = Exploratory.Phase(X = dgp$X[1:70,],
                Y = dgp$Y[1:70], list.reduction = v2,
                family = gaussian, signif = 0.01)
```

Neither squared terms nor interaction terms are suggested as potentially important.

Model selection phase

The final stage of the procedure is to test all low-dimensional subsets of the comprehensive model for compatibility with the data. The comprehensive model is that containing all variables from the reduction phase and any squared or interaction terms suggested at the exploratory phase, of which there are none in the present example. Usage is:

```
out.MS = ModelSelection.Phase(X = dgp$X[71:100,],
                Y = dgp$Y[71:100], list.reduction = v2, signif = 0.01)
```

The appropriate modification to the arguments of this function when squared or interaction terms are to be considered is illustrated in the Real example section.

The above finds all models of dimension 5 or smaller whose likelihood ratio test against the comprehensive is not rejected at the `signif = 0.01` significance level. The optional argument `modelSize` specifies the maximum size of the models to be searched over. The true model appears in the set of all well-fitting models identified, i.e., in the list of models displayed by `out.MS$goodModels$` `` `Model Size 5` ``. All models that are found to be compatible with the data should be reported. Specifically, the output of the function `ModelSelection.Phase` should be used to produce (sometimes large) tables like those appearing in the supplementary file of Cox and Battey (2017), available at:
https://www.pnas.org/content/pnas/suppl/2017/07/20/1703764114.DCSupplemental

Provided that the sample is split as in the example above, such tables constitute a conditional confidence set of models. Conditional on the reduction phase, these have, in principle, exact nominal coverage in the linear regression model and asymptotically nominal coverage in more general regression models fitted by maximum likelihood.

Illustration of performance in some idealized settings

The present section explores empirical sensitivities of the procedure to modifications to the data generating mechanism. Several aspects are of interest: sensitivity of the reduction phase as described by Cox and Battey (2017) (a version of backward elimination) and of the undertuned lasso (a version of forward selection) in terms of retaining the true model in its entirety; efficacy of the model confidence sets in terms of their coverage probabilities and size. Full sample and split sample properties of both approaches are considered.

It is an open problem to elucidate the conditions on the design matrix and signal strength in order for the procedure based on traversal of successively lower dimensional hypercubes to retain a reasonably sized superset of the true set of signal variables with quantifiable high probability. Some related discussion for the undertuned lasso is given by Bühlmann and Van De Geer (2011) chapter 7 and Belloni and Chernozhukov (2013).

In the tables below, $\mathcal{S}$ is the true set of signal variables, $\widehat{\mathcal{S}}$ is the set of variables surviving the reduction phase, $\mathcal{M}$ is the set of low-dimensional models whose likelihood ratio test against the comprehensive model is not rejected at the 1% level. In all the simulation experiments considered, the first stage of the reduction phase arranges the 1000 variables in a $10 \times 10 \times 10$ cube and retains variables if they are among the two most significant in at least two of the three analyses in which they appear. The second stage reduction is tuned so that approximately 10-20 variables are retained through the reduction phase, however the associated threshold for the significance tests is fixed across Monte Carlo replications so that the number of retained variables is random. Results for the linear model with a sample of size $n = 100$ are reported in Table 1, where 'CB' is the procedure of Cox and Battey (2017) implemented using **HCmodelSets** package. The threshold of the second-stage significance test is 0.1%.

				$\text{pr}(\mathcal{S} \subseteq \widehat{\mathcal{S}})$				$\text{pr}(\mathcal{S} \in \mathcal{M})$		$\mathbb{E}\lvert\mathcal{M}\backslash\mathcal{S}\rvert$	
v_{S0}	v_{C0}	ρ	signal / noise	undertuned lasso (full)	undertuned lasso (split)	CB (full)	CB (split)	CB (full)	CB (split)	CB (full)	CB (split)
1	1	0.9	1	1.00 (0.04)	0.99 (0.08)	1.00 (0.00)	1.00 (0.00)	0.57 (0.49)	0.98 (0.14)	6.16 (7.25)	16.3 (28.6)
1	1	0.9	0.6	0.94 (0.24)	0.84 (0.37)	0.98 (0.15)	0.83 (0.37)	0.41 (0.49)	0.83 (0.38)	4.93 (4.82)	16.1 (31.5)
1	1	0.5	1	0.92 (0.27)	0.87 (0.34)	1.00 (0.00)	1.00 (0.00)	0.56 (0.50)	0.98 (0.13)	4.63 (5.89)	8.73 (18.7)
1	1	0.5	0.6	0.85 (0.36)	0.75 (0.43)	0.99 (0.11)	0.85 (0.35)	0.39 (0.49)	0.84 (0.36)	2.29 (2.73)	8.68 (19.2)
1	3	0.9	1	1.00 (0.04)	0.99 (0.08)	1.00 (0.04)	0.99 (0.08)	0.62 (0.49)	0.96 (0.19)	24.6 (24.8)	77.2 (126)
1	3	0.9	0.6	0.92 (0.27)	0.79 (0.41)	0.97 (0.16)	0.81 (0.39)	0.48 (0.50)	0.79 (0.41)	22.7 (18.4)	44.6 (78.1)
1	3	0.5	1	0.97 (0.16)	0.92 (0.27)	1.00 (0.00)	1.00 (0.00)	0.59 (0.49)	0.98 (0.13)	10.9 (14.0)	14.3 (36.1)
1	3	0.5	0.6	0.89 (0.31)	0.82 (0.39)	0.97 (0.17)	0.87 (0.34)	0.36 (0.48)	0.85 (0.35)	3.66 (5.01)	10.3 (25.0)
5	1	0.9	1	0.98 (0.15)	0.95 (0.21)	1.00 (0.00)	1.00 (0.00)	0.94 (0.23)	0.98 (0.13)	7.15 (7.19)	80.4 (85.7)
5	1	0.9	0.6	0.79 (0.40)	0.57 (0.50)	1.00 (0.04)	0.98 (0.15)	0.89 (0.31)	0.96 (0.19)	40.9 (35.7)	146 (149)
5	1	0.5	1	1.00 (0.04)	0.98 (0.15)	1.00 (0.00)	1.00 (0.00)	0.96 (0.20)	0.99 (0.11)	0.01 (0.10)	8.64 (13.0)
5	1	0.5	0.6	0.99 (0.10)	0.96 (0.21)	1.00 (0.00)	0.99 (0.10)	0.88 (0.32)	0.98 (0.15)	1.18 (2.04)	51.6 (64.2)
5	3	0.9	1	0.99 (0.11)	0.95 (0.22)	1.00 (0.00)	1.00 (0.00)	0.94 (0.24)	0.98 (0.15)	16.7 (18.4)	212 (202)
5	3	0.9	0.6	0.77 (0.42)	0.51 (0.50)	1.00 (0.06)	0.96 (0.20)	0.86 (0.35)	0.94 (0.24)	101 (88.2)	418 (351)
5	3	0.5	1	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.96 (0.20)	0.98 (0.14)	0.01 (0.10)	20.0 (35.0)
5	3	0.5	0.6	1.00 (0.06)	0.98 (0.13)	1.00 (0.00)	0.99 (0.12)	0.89 (0.32)	0.96 (0.20)	2.85 (4.03)	123 (137)

Table 1: Monte Carlo estimates and their estimated standard errors (in parentheses) from 500 Monte Carlo draws from the linear model with parameter combinations as displayed. In the split sample case, 70 observations are used for reduction and 30 for construction of the confidence sets of models.

The same experiment is performed on survival time data, generated according to a proportional hazards model with Weibull baseline hazard as described in the section entitled Some simple data generating processes. The survival times are censored, with the censoring times generated from an exponential distribution of rate 0.1. In particular, our previous code fragment is modified so that in each Monte Carlo replication, data are generated as:

```
dgp = DGP(s=Vs0, a=Vc0, sigStrength=sigNoise, rho=corr, n=150, intercept=0,
                DGP.seed = (n.rand + cont.error),
                var=1, d=1000, type.response = "S", scale=1, shape=1, rate=0.1)
```

adopting the values `c(1,5)` for `Vs0`, `c(1,3)` for `Vc0`, `c(0.9,0.5)` for `corr` and `c(1,0.6)` for `sigNoise` as previously done for the linear theory model in Table 1.

In the notation of Section Some simple data generating processes, the parameters of the Weibull distribution are set as τ = `scale` = 1, and κ = `shape` = 1. Knowledge of the baseline hazard is ignored and the data are fit by partial likelihood as implemented in the `coxph` function of the **survival** package. Summary statistics over 500 Monte Carlo replications are reported in Table 2. The threshold of the second-stage significance test is 0.25%.

				$\text{pr}(\mathcal{S} \subseteq \widehat{\mathcal{S}})$				$\text{pr}(\mathcal{S} \in \mathcal{M})$		$\mathbb{E}\lvert\mathcal{M}\backslash\mathcal{S}\rvert$	
v_{S0}	v_{C0}	ρ	$\frac{\text{signal}}{\text{noise}}$	undertuned lasso (full)	undertuned lasso (split)	CB (full)	CB (split)	CB (full)	CB (split)	CB (full)	CB (split)
1	1	0.9	1	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.03 (0.17)	0.95 (0.23)	54.1 (92.2)	1273 (1490)
1	1	0.9	0.6	0.99 (0.12)	0.94 (0.24)	1.00 (0.04)	0.97 (0.17)	0.00 (0.04)	0.89 (0.31)	15.6 (57.3)	1863 (2264)
1	1	0.5	1	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.04 (0.21)	0.95 (0.21)	57.4 (97.4)	962 (1085)
1	1	0.5	0.6	1.00 (0.00)	0.98 (0.13)	0.99 (0.08)	0.96 (0.20)	0.00 (0.00)	0.90 (0.31)	13.0 (31.6)	1734 (2374)
1	3	0.9	1	1.00 (0.00)	0.99 (0.09)	1.00 (0.00)	1.00 (0.04)	0.07 (0.25)	0.95 (0.22)	102 (209)	2468 (2738)
1	3	0.9	0.6	0.97 (0.18)	0.90 (0.30)	0.98 (0.13)	0.95 (0.21)	0.01 (0.09)	0.91 (0.29)	45.0 (98.5)	3182 (3700)
1	3	0.5	1	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.07 (0.25)	0.95 (0.22)	105 (158)	1094 (1090)
1	3	0.5	0.6	1.00 (0.00)	1.00 (0.06)	1.00 (0.00)	0.97 (0.17)	0.00 (0.04)	0.91 (0.28)	18.6 (51.1)	1955 (2859)
5	1	0.9	1	0.98 (0.15)	0.90 (0.30)	1.00 (0.00)	1.00 (0.04)	0.78 (0.41)	0.91 (0.29)	30.9 (46.5)	916 (1165)
5	1	0.9	0.6	0.79 (0.41)	0.52 (0.50)	1.00 (0.00)	0.99 (0.08)	0.59 (0.49)	0.94 (0.24)	136 (180)	2216 (2390)
5	1	0.5	1	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.80 (0.40)	0.91 (0.28)	0.00 (0.09)	59.0 (118)
5	1	0.5	0.6	1.00 (0.00)	0.99 (0.12)	1.00 (0.00)	1.00 (0.04)	0.54 (0.50)	0.90 (0.31)	1.46 (4.22)	382 (572)
5	3	0.9	1	0.98 (0.13)	0.86 (0.35)	1.00 (0.00)	1.00 (0.04)	0.80 (0.40)	0.86 (0.35)	46.4 (66.2)	1383 (1682)
5	3	0.9	0.6	0.71 (0.45)	0.48 (0.50)	1.00 (0.00)	0.99 (0.11)	0.63 (0.48)	0.90 (0.30)	242 (310)	2846 (2603)
5	3	0.5	1	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	0.83 (0.38)	0.87 (0.34)	0.09 (1.03)	73.4 (175)
5	3	0.5	0.6	1.00 (0.00)	0.99 (0.11)	1.00 (0.00)	1.00 (0.04)	0.59 (0.49)	0.90 (0.30)	2.35 (5.25)	575 (925)

Table 2: Monte Carlo estimates and their estimated standard errors (in parentheses) from 500 Monte Carlo draws from the Weibull proportional hazards model with parameter combinations as displayed.

The results are qualitatively similar to those for the linear model. The main difference is that the coverage probability of the confidence sets of models, conditional on all variables being retained through the first stage reduction, is lower than the 0.99 nominal level. The reason is that the distribution theory underpinning the associated likelihood ratio tests is, in principle, exact for the linear model and is at best asymptotically valid for most other types of regression model. This could be mitigated through a Bartlett correction to the likelihood ratio statistic, but this has not been implemented in the current version of the package. The results of Table 2 are for $n = 150$ with 100 observations used for reduction and 50 for construction of confidence sets of models in the split sample case. As mentioned previously, adjustments to the likelihood ratio statistic to improve the χ^2 approximation to its distribution are possible, but these have not been implemented in **HCmodelSets**.

Real example

We now illustrate the use of **HCmodelSets** to construct conditional confidence sets of models for the survival times of lymphoma patients. The data[1] are from the study of Alizadeh et al. (2000) and also used by Simon et al. (2011). There are measurements on $d = 7399$ genetic variants for $n = 240$ patients. The indices of these variables are arranged in a 4 dimensional hypercube, which is the default starting dimension. As before, the data are divided into those to be used in the reduction and exploratory phases and those to be used in the model selection phases.

```
data(LymphomaData)
x = t(patient.data$x)
y = patient.data$time
status = patient.data$status
# Data Splitting
X.in = x[1:168,]
Y.in = y[1:168]
status.in = status[1:168]
Y = cbind(Y.in,status.in)
X.out = x[169:240,]
Y.out = y[169:240]
status.out = status[169:240]
```

The first stage decision rule is to retain all variables that are among the two most significant in at least two of the three analyses in which they appear. The decision rules for the remaining reduction stages are specified by the argument `vector.signif` in the `Reduction.Phase` function:

[1] Avaliable at https://www.jstatsoft.org/article/view/v039i05

```
library(HCmodelSets)
out.1 = Reduction.Phase(X = X.in,Y = Y,Cox.Hazard = TRUE,
                vector.signif = c(2,0.0025,0.001), seed.HC = 2)
```

The choice `vector.signif = c(2,0.0025,0.001)` means that the second stage decision rule retains variables if they are significant at the 0.25% level in at least two of the three analyses in which they appear and the third stage decision rule retains variables if they are significant at the 0.1% level in at least one of the two analyses in which they appear. This choice was determined by checking that the numbers of variables retained through each stage of the reduction is sensible, that the number of variables ultimately retained is within the target range, and that the outcome is not too sensitive to changes to the original arrangement of the variable indices in the hypercube. The set of variables ultimately retained is

```
v1 = out.1$List.Selection$`Hypercube with dim 2`$numSelected1
sort(v1)
#> [1] 1188 1660 1825 2437 2879 2902 3172 3177 3800 3814 3822 3824 5027 6134 6706 6896 7357
```

Rerandomizing the variable indices in the hypercube produces the set of variables

```
out.2 = Reduction.Phase(X = X.in,Y = Y,Cox.Hazard = TRUE,
                vector.signif = c(2,0.0025,0.001), seed.HC = 11)
v2 = out.2$List.Selection$`Hypercube with dim 2`$numSelected1
sort(v2)
#> [1] 1188 1675 1714 1825 1984 2437 2900 3172 3800 3811 3818 3819 3822 3833  4126
        5027  6134  6706  7069  7357
```

Ten of the variables in the original list of 17 are also in the second list. The lasso, undertuned to select at least the same number of variables as in `v1` produces an overlap of 8 variables with `v1`, namely,

```
library(glmnet)
lasso.fit = glmnet(X.in, Surv(Y.in,status.in),
                family = "cox", alpha = 1)
n.coefs = apply(coef(lasso.fit), 2, function(x) length(which(x!=0)))
idx.coefs = which(n.coefs == length(v1))
if(length(idx.coefs)==0){
        idx.coefs = min(which(n.coefs >= v1))}
lasso.var = which(coef(lasso.fit)[,idx.coefs[1]]!=0)
lasso.var
#> [1] 394 1072 1188 1456 1662 1681 1825 2902 3172 3180 3801
        3822 4882 5027 6134 6896 7357
```

The variable 3801, found by the lasso, has empirical correlation greater than 0.9 with variable 3800 in `v1` and `v2`.

The exploratory phase now uses significance tests as an informal guide to suggesting potential squared or interaction terms. For each of the variables in `v1`, a regression is fitted by partial likelihood with its squared term added. Extreme t statistics on squared terms suggest a potentially important effect. The linear interactions of pairs of variables are checked in a similar way, with `silent = FALSE` in `Exploratory.Phase` producing plots of the response variable as a function of pairs of variables for any interaction suggested as potentially important. Example usage is

```
out.exp.phase =  Exploratory.Phase(X=X.in,Y=Y,
                list.reduction = v1, silent = FALSE,
                Cox.Hazard = TRUE, signif=0.01)
```

which produces a sequence of plots and questions of the form

```
Discard interaction term? [Y/N].
```

For illustrative purposes, we answer `N` (no) to the questions for which the plots are displayed in Figure 1, although the suggestion from an interaction plot ought to be much stronger to justify an interaction's inclusion. See Cox and Battey (2017) for an example.

Thus we have 20 variables in all, the seventeen variables contained in `v1`, one squared term contained in `out.Exploratory.Phase$mat.select.SQ` and two interaction terms given by the rows of `out.Exploratory.Phase$mat.select.INTER`. The analysis proceeds as follows:

```
sq.terms = out.exp.phase$mat.select.SQ
in.terms = out.exp.phase$mat.select.INTER
```

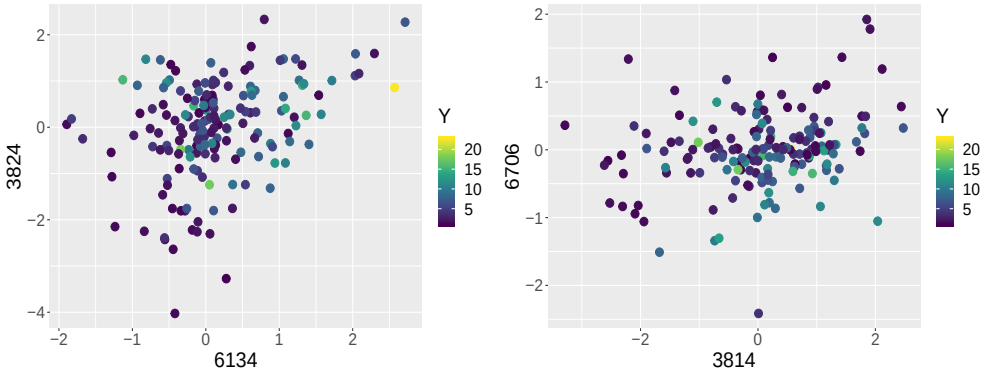


Figure 1: Interaction plots of variables (6134, 3824) and (3814, 6706)

A	B																			
	12	14	19	10	4	15	20	3	7	9	18	1	11	17	13	5	6	16	2	8
12		0.26	0.24	0.87	0.81	0.84	0.76	0.75	0.77	0.74	0.72	0.73	0.73	0.72	0.72	0.71	0.71	0.71	0.71	0.70
14	0.22		0.20	0.85	0.80	0.83	0.75	0.74	0.75	0.73	0.71	0.72	0.71	0.70	0.71	0.70	0.69	0.69	0.69	0.69
19	0.00	0.00		0.82	0.75	0.79	0.69	0.67	0.70	0.65	0.63	0.65	0.63	0.62	0.63	0.61	0.61	0.61	0.61	0.60
10	0.75	0.73	0.73		0.47	0.35	0.37	0.45	0.46	0.44	0.47	0.47	0.48	0.47	0.47	0.48	0.48	0.48	0.47	0.48
4	0.62	0.63	0.63	0.45		0.41	0.48	0.48	0.40	0.46	0.47	0.43	0.43	0.45	0.44	0.43	0.44	0.44	0.45	0.44
15	0.68	0.68	0.68	0.31	0.40		0.31	0.42	0.43	0.43	0.42	0.43	0.43	0.44	0.44	0.44	0.44	0.45	0.44	0.44
20	0.48	0.47	0.48	0.28	0.43	0.26		0.40	0.40	0.40	0.40	0.39	0.39	0.39	0.40	0.39	0.39	0.39	0.39	0.39
3	0.44	0.44	0.44	0.34	0.41	0.36	0.38		0.38	0.39	0.37	0.37	0.37	0.37	0.36	0.37	0.37	0.37	0.38	0.37
7	0.48	0.48	0.48	0.37	0.32	0.37	0.38	0.38		0.39	0.37	0.37	0.38	0.38	0.38	0.38	0.37	0.37	0.36	0.37
9	0.38	0.38	0.38	0.30	0.35	0.34	0.35	0.35	0.35		0.35	0.34	0.32	0.34	0.34	0.34	0.33	0.34	0.34	0.34
18	0.33	0.34	0.32	0.33	0.37	0.32	0.33	0.33	0.32	0.34		0.34	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33
1	0.35	0.35	0.35	0.32	0.31	0.32	0.32	0.32	0.32	0.32	0.33		0.32	0.32	0.33	0.32	0.32	0.32	0.32	0.32
11	0.32	0.31	0.30	0.32	0.28	0.29	0.30	0.30	0.31	0.29	0.30	0.31		0.30	0.30	0.30	0.31	0.30	0.30	0.30
17	0.29	0.29	0.28	0.30	0.30	0.30	0.29	0.30	0.30	0.30	0.29	0.30	0.30		0.29	0.30	0.30	0.30	0.30	0.30
13	0.30	0.30	0.28	0.30	0.29	0.30	0.31	0.28	0.30	0.30	0.30	0.31	0.30	0.29		0.29	0.30	0.29	0.30	0.30
5	0.25	0.25	0.24	0.28	0.26	0.28	0.27	0.27	0.28	0.28	0.27	0.27	0.28	0.28	0.27		0.28	0.27	0.27	0.27
6	0.24	0.23	0.22	0.28	0.26	0.27	0.26	0.26	0.26	0.26	0.27	0.27	0.27	0.27	0.27	0.27		0.27	0.27	0.27
16	0.23	0.23	0.22	0.27	0.25	0.28	0.26	0.26	0.25	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26		0.26	0.26
2	0.23	0.23	0.21	0.27	0.27	0.27	0.26	0.27	0.25	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26		0.26
8	0.20	0.20	0.18	0.26	0.24	0.26	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.25	0.24	0.24	0.25	0.25	

Table 3: The proportion of the models in the confidence set not containing variable B that contain variable A, i.e. $|\mathcal{M}(A \cap \neg B)|/|\mathcal{M}(\neg B)|$, where $\mathcal{M}(\neg B)$ is the set of models in the confidence set that do not contain variable B.

```
out.MS = ModelSelection.Phase(X = X.out,
                Y = cbind(Y.out,status.out),
                list.reduction = v1,Cox.Hazard = TRUE, sq.terms = sq.terms,
                in.terms = in.terms,signif = 0.05, modelSize = 7)
```

Sets of well-fitting models of different sizes, up to `modelSize = 7`, are contained in the list `out.MS$goodModels`. For instance, `out.MS$goodModels$`Model Size 2`` produces a list of well-fitting models of size 2. If there are models for which an interaction term is present without the corresponding main effects, the main effects are added. Thus, there are 23 models of size 2, statistically indistinguishable from the comprehensive model at the 5% significance level.

Of all the well-fitting models identified 72% involve the variable 3824 and 70% involve the variable 6134. A very small proportion of models contain neither 3814 nor 6134. Indeed variables 3814 and 6134 occur frequently, but rarely together. Table 3 reports the proportion of models containing variable A, given that they do not contain variable B, say. While one should be cautious over overinterpretting the output, these give an indication of which variables might be substitutes for one another. The variables have been ordered, from left to right and from top to bottom, in order of their frequency of appearance in the sets of models. For typographical reasons their indices have been recoded as: 1=1188; 2=1660; 3=1825; 4=2437; 5=2879; 6=2902; 7=3172; 8=3177; 9=3800; 10=3814; 11=3822; 12=3824; 13=5027; 14=6134; 15=6706; 16=6896; 17=7357; 18=squared term on 3814; 19=interaction between 6134 and 3824; 20=interaction between 3814 and 6706. For an example of other summary tables of potential interest, see the supplementary file of Cox and Battey (2017).

Summary

In the context of regression with a large number of potential explanatory variables Cox and Battey (2017) emphasize that if there are several statistically indistinguishable explanations of the data, one should aim to specify as many as is feasible, a view that is in contraposition to that implicit in the use of the lasso and similar methods. The approach of Cox and Battey (2017) entails reducing the set of variables to those that potentially have an individual effect on the response, followed by more detailed joint exploration, requiring judgement at various stages. We have discussed the R implementation of these new ideas in **HCmodelSets**. Matlab code is also available at http://wwwf.imperial.ac.uk/~hbattey/softwareCube.html.

Acknowledgments

We thank D. R. Cox for valuable comments and M. Avella Medina for a helpful reference. The work of H. H. Hoeltgebaum was fully supported by the National Council for Research and Development, CNPq, Ministry of Science and Technology, Brazil. The development of **HCmodelSets** and the work of H. S. Battey was supported by a UK Engineering and Physical Sciences Research Fellowship (grant number EP/P002757/1). We also would like to thank the anonymous referees for helpful comments.

Bibliography

A. A. Alizadeh, M. B. Eisen, R. E. Davis, C. Ma, I. S. Lossos, A. Rosenwald, J. C. Boldrick, H. Sabet, T. Tran, X. Yu, J. I. Powell, L. Yang, G. E. Marti, T. Moore, J. Hudson Jr, L. Lu, D. B. Lewis, R. Tibshirani, G. Sherlock, W. C. Chan, T. C. Greiner, D. D. Weisenburger, J. O. Armitage, R. Warnke, R. Levy, W. Wilson, M. R. Grever, J. C. Byrd, D. Botstein, P. O. Brown, and L. M. Staudt. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503, 2000. [p375]

O. E. Barndorff-Nielsen and D. R. Cox. Bartlett adjustments to the likelihood ratio statistic and the distribution of the maximum likelihood estimator. *J. of the Royal Statistical Society B*, 46(3):483–495, 1984. [p372]

O. E. Barndorff-Nielsen and D. R. Cox. *Inference and Asymptotics*, volume 13. Chapman & Hall London, 1994. [p372]

M. S. Bartlett. Properties of sufficiency and statistical tests. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 160(901):268–282, 1937. [p372]

H. Battey and D. R. Cox. Large numbers of explanatory variables: a probabilistic assessment. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474, 2018. [p370, 371, 372]

A. Belloni and V. Chernozhukov. Least squares after model selection in high-dimensional sparse models. *Bernoulli*, 19(2):521–547, 2013. [p374]

P. Bühlmann and S. Van De Geer. *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer-Verlag, 2011. [p374]

D. R. Cox. The regression analysis of binary sequences. *J. of the Royal Statistical Society B*, 20(2):215–232, 1958. [p370]

D. R. Cox. Regression models and life tables (with discussion). *J. of the Royal Statistical Society*, 34(2): 187–220, 1972. [p370]

D. R. Cox. A note on data-splitting for the evaluation of significance levels. *Biometrika*, 62(2):441–444, 1975a. [p371]

D. R. Cox. Partial likelihood. *Biometrika*, 62:269–276, 1975b. [p370]

D. R. Cox and H. Battey. Large numbers of explanatory variables, a semi-descriptive analysis. *Proceedings of the National Academy of Sciences*, 114(32):8592–8595, 2017. [p370, 373, 374, 376, 377, 378]

H. H. Hoeltgebaum. *HCmodelSets: Regression with a Large Number of Potential Explanatory Variables*, 2018. URL https://CRAN.R-project.org/package=HCmodelSets. R package version 1.0.2. [p370]

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, 2011. [p375]

R. Tibshirani. Regression shrinkage and selection via the lasso. *J. of the Royal Statistical Society B*, 58(1): 267–288, 1996. [p370]

F. Yates. A new method of arranging variety trials involving a large number of varieties. *The Journal of Agricultural Science*, 26(3):424–455, 1936. [p370]

Henrique Helfer Hoeltgebaum
Electrical Engineering Department, Pontifical Catholic University of Rio de Janeiro
R. Marquês de São Vicente, 225 - Gávea,
Rio de Janeiro - RJ, 22451-900, Brazil
Department of Mathematics, Imperial College London
180 Queen's Gate, South Kensington,
London, SW7 2AZ, UK
hh3015@ic.ac.uk

Heather S. Battey
Department of Mathematics, Imperial College London
180 Queen's Gate, South Kensington,
London, SW7 2AZ, UK
h.battey@imperial.ac.uk

Resampling-Based Analysis of Multivariate Data and Repeated Measures Designs with the R Package MANOVA.RM

by Sarah Friedrich, Frank Konietschke and Markus Pauly

Abstract Nonparametric statistical inference methods for a modern and robust analysis of longitudinal and multivariate data in factorial experiments are essential for research. While existing approaches that rely on specific distributional assumptions of the data (multivariate normality and/or equal covariance matrices) are implemented in statistical software packages, there is a need for user-friendly software that can be used for the analysis of data that do not fulfill the aforementioned assumptions and provide accurate p value and confidence interval estimates. Therefore, newly developed nonparametric statistical methods based on bootstrap- and permutation-approaches, which neither assume multivariate normality nor specific covariance matrices, have been implemented in the freely available R package **MANOVA.RM**. The package is equipped with a graphical user interface for plausible applications in academia and other educational purpose. Several motivating examples illustrate the application of the methods.

Introduction

Nowadays, a large amount of measurements are taken per experimental unit or subject in many experimental studies—requiring inferential methods from multivariate analysis in a unified way. Here we distinguish between *two cases*:

1. If the same quantity is measured under different treatment conditions or at different time points, a *repeated measures* (RM) design is present. Therein, observations are measured on the same scale and are *combinable*. This is also the case if the measuring instrument produces multiple responses, e.g., microarrays in bioinformatics.
2. If different quantities are measured on the same unit or subject, a *multivariate analysis of variance* (MANOVA) design is apparent. In such a situation, data is measured on different scales and not combinable (e.g., height and weight).

These two different definitions do not only lead to different questions of interest but also require different inference procedures as outlined below. In particular, the main difference between the two approaches is that in repeated measures designs comparisons between the response variables are meaningful. This means that also hypotheses regarding sub-plot or within subject factors (e.g., time) are of interest. On the other hand, MANOVA settings are usually designed to detect effects of the observed factors (and interactions thereof) on the multivariate outcome vectors, thus allowing —in contrast to multiple univariate ANOVA analyses—to evaluate the *combined* changes of the outcome variables with respect to the factor levels.

Despite their differences, MANOVA- and RM-type techniques share the same advantages over classical univariate endpoint-wise—ANOVA-type—analyses:

- They provide joint inference and take the dependency across the endpoints into account, thus leading to possibly larger power to detect underlying effects.
- They allow for testing of additional factorial structures and
- can easily be equipped with a closed testing procedure for subsequently detecting local effects in specific components, i.e. to perform post-hoc analyses.

Focusing on metric data and mean-based procedures, MANOVA and RM models are typically inferred by means of "classical" procedures such as Wilks' Lambda, Lawley-Hotelling, Roy's largest root (Davis, 2002; Johnson and Wichern, 2007; Anderson, 2001) or (generalized) linear mixed models with generalized estimating equations. For the classical one-way layout, these methods are implemented in R within the `manova` function in the **stats** package, where one can choose between the options `'Pillai'`, `'Wilks'`, `'Hotelling-Lawley'` and `'Roy'`. Nonparametric rank-based methods for null hypotheses formulated in distribution functions are implemented within the packages **npmv** for one- and two-way MANOVA (Burchett et al., 2017) and **nparLD** for several repeated measures designs

(Noguchi et al., 2012). In case of fixed block effects, the **GFD** package (Friedrich et al., 2017b), which implements a permutation Wald-type test in the univariate setting, can also be used.

(Generalized) linear mixed models are implemented in the `lm` and the `glm` function (package **stats**) for univariate data as well as in the **SCGLR** package for Generalized Linear Model estimation in the context of multivariate data (Cornu et al., 2018). The `Anova` and `Manova` function in the **car** package (Fox and Weisberg, 2011) calculate type-II and type-III analysis-of-variance tables for objects produced by, e.g., `lm`, `glm` or `manova` in the univariate and multivariate context, respectively. In the MANOVA context, repeated measures designs can be included as well.

Furthermore, the packages **flip** (Finos et al., 2018) and **ffmanova** (Øyvind Langsrud and Mevik, 2019) contain interesting permutation and rotation tests, which, however, require certain invariances resulting in model restrictions (see, e.g., the discussion in Huang et al., 2006; Chung and Romano, 2013) .

Most of these procedures, however, rely on specific distributional assumptions (such as multivariate normality) and/or specific covariance or correlation structures (e.g., homogeneity between groups or, for RM, compound symmetry; possibly implying equal correlation between measurements) which may often not be justifiable in real data. In particular, with decreasing sample sizes and increasing dimensions, such presumptions are almost impossible to verify in practice and may lead to inflated type-I-errors, cf. Vallejo et al. (2001); Lix and Keselman (2004); Vallejo Seco et al. (2007); Livacic-Rojas et al. (2010). To this end, several alternative procedures have been developed that tackle the above problems and have been compared in extensive simulation studies, see amongst others Brunner (2001); Lix and Lloyd (2007); Gupta et al. (2008); Zhang (2011); Harrar and Bathke (2012); Konietschke et al. (2015); Xiao and Zhang (2016); Bathke et al. (2018); McFarquhar et al. (2016); Friedrich et al. (2017a); Livacic-Rojas et al. (2017); Friedrich and Pauly (2018) and the references cited therein. Here, we focus on nonparametric statistical methods that are valid in the multivariate Behrens-Fisher situation—equal covariance matrices across the groups are not assumed—and provide accurate inferential results in terms of p value estimates and confidence intervals for the parameters of interest. In particular, we implemented bootstrap- and permutation-based approaches to approximate the distribution of the test statistics in a robust way. Simulation studies comparing these approaches to the traditional methods mentioned above can, e. g. be found in the main papers and the supplements of Friedrich et al. (2017a) and Bathke et al. (2018).

More precisely, we focus on nonparametric methods for testing main and interaction effects of *fixed* factors in repeated measures designs and multivariate data. In particular, general Wald-type test statistics (for MANOVA and RM), ANOVA-type statistics (for RM) and modified ANOVA-type tests (for MANOVA) are implemented in **MANOVA.RM** (Friedrich et al., 2019) because

- they can be used to test hypotheses in various factorial designs in a flexible way,
- their sampling distribution can be approximated by resampling techniques, even allowing their application for small sample sizes,
- and they are appropriate methods in the Behrens-Fisher situation.

To make the methods freely accessible we have provided the R package **MANOVA.RM** for routine statistical analyses. It is available from the R Archive at

`https://CRAN.R-project.org/package=MANOVA.RM`

The main functions `RM` (for RM designs) and `MANOVA` (for MANOVA designs) are developed in style of the well known ANOVA functions `lm` or `aov`. Its user-friendly application not only provides the p values and test statistics of interest but also a descriptive overview together with component-wise two-sided confidence intervals. Moreover, the `MANOVA` function even allows for an easy calculation and confidence ellipsoid plots for specified multivariate contrasts as described in Friedrich and Pauly (2018).

Specifically, for testing multivariate main- and interaction effects in one-, two- and higher-way MANOVA models, the `MANOVA` function provides

- the Wald-type statistic (WTS) proposed by Konietschke et al. (2015) using a parametric bootstrap, a wild bootstrap or its asymptotic χ^2-distribution for p value computations, and
- the modified ANOVA-type statistic (MATS) proposed by Friedrich and Pauly (2018) using a parametric or wild bootstrap procedure for p value computations.

In addition to multivariate group-wise effects, the `RM` function also allows to test hypotheses formulated across within subject factors. The implemented test statistics are

- the ANOVA-type statistic (ATS) using an F-approximation as considered in Brunner (2001) as well as a parametric and a wild bootstrap approach and

- the Wald-type statistic (WTS) using the asymptotic χ^2-distribution (Brunner, 2001), the permutation technique proposed in Friedrich et al. (2017a) as well as a parametric (Bathke et al., 2018) and a wild bootstrap approach for p value estimation.

The paper is organized as follows: In Section Statistical model and inference methods the multivariate statistical model as well as the implemented inference procedures are described. The application of the R package **MANOVA.RM** is exemplified on several Repeated Measures and MANOVA Examples in Section Examples. Finally, the paper closes with a discussion in Section Discussion and Outlook.

Throughout the paper we use the subsequent notation from multivariate linear models: For $a \in \mathbb{N}$ we denote by $\boldsymbol{P}_a = \boldsymbol{I}_a - \frac{1}{a}\boldsymbol{J}_a$ the a-dimensional centering matrix, by $\boldsymbol{I}_a$ the a-dimensional identity matrix and by $\boldsymbol{J}_a$ the $a \times a$ matrix of 1's, i.e., $\boldsymbol{J}_a = \mathbf{1}_a\mathbf{1}_a'$, where $\mathbf{1}_a = (1,\ldots,1)'$ is the a-dimensional column vector of 1's.

Statistical model and inference methods

For both the RM and the MANOVA design equipped with an arbitrary number of fixed factors, we consider the general linear model given by d-variate random vectors

$$\mathbf{v}^\top \mathcal{X}_{ik} \;=\; (X_{ijk})_{j=1}^{d} \quad = \boldsymbol{\mu}_i + \boldsymbol{\epsilon}_{ik}. \tag{1}$$

Here, $k = 1,\ldots,n_i$ denotes the experimental unit or subject in group $i = 1,\ldots,a$. Note, that a higher-way factorial structure on the groups/between subject or within subject factors can be achieved by sub-indexing the indices i (group/between subject factors) or j (within subject factors) into $i_1,\ldots,i_p$ or $j_1,\ldots,j_q$. In this model $\boldsymbol{\mu}_i = (\mu_{i1},\ldots,\mu_{id})' \in R^d$ is the mean vector in group $i = 1,\ldots,a$ and for each fixed i it is assumed that the error terms $\boldsymbol{\epsilon}_{ik}, k = 1,\ldots,n_i$, are independent and identically distributed d-variate random vectors with mean $E(\boldsymbol{\epsilon}_{i1}) = 0$ and existing variances $0 < \sigma_{ij}^2 = \mathrm{var}(X_{ijk}) < \infty$, $j = 1,\ldots,d$. For the WTS-type procedures we additionally assume positive definite covariance matrices $\mathrm{cov}(\boldsymbol{\epsilon}_{i1}) = \boldsymbol{V}_i > 0$ and existing finite fourth moments $E(||\boldsymbol{\epsilon}_{i1}||^4) < \infty$.

Within this framework, hypotheses for RM or MANOVA can be formulated by means of an adequate contrast hypothesis matrix $\boldsymbol{H}$ by

$$H_0 : \boldsymbol{H}\boldsymbol{\mu} = \mathbf{0},$$

where $\boldsymbol{\mu} = (\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_a)'$.

Let $\overline{\boldsymbol{X}}_\bullet = (\overline{\boldsymbol{X}}_{1\cdot}',\ldots,\overline{\boldsymbol{X}}_{a\cdot}')'$ denote the vector of pooled group means $\overline{\boldsymbol{X}}_{i\cdot} = \frac{1}{n_i}\sum_{k=1}^{n_i}\mathbf{v}^\top\mathcal{X}_{ik}, i = 1,\ldots,a$ and $\widehat{\boldsymbol{\Sigma}}_N = N \cdot \mathrm{diag}\{\widehat{\boldsymbol{V}}_1/n_1,\ldots,\widehat{\boldsymbol{V}}_a/n_a\}$ the estimated covariance of $\sqrt{N}\overline{\boldsymbol{X}}_\bullet$. Here, $N = \sum_i n_i$ and $\widehat{\boldsymbol{V}}_i = \frac{1}{n_i-1}\sum_{k=1}^{n_i}(\mathbf{v}^\top\mathcal{X}_{ik} - \overline{\boldsymbol{X}}_{i\cdot})(\mathbf{v}^\top\mathcal{X}_{ik} - \overline{\boldsymbol{X}}_{i\cdot})'$. In this set-up Konietschke et al. (2015) propose a statistic of Wald-type (WTS)

$$T_N = T_N(\mathbf{v}^\top\mathcal{X}) = N\overline{\boldsymbol{X}}_\bullet'\boldsymbol{T}(\boldsymbol{T}\widehat{\boldsymbol{\Sigma}}_N\boldsymbol{T})^+\boldsymbol{T}\overline{\boldsymbol{X}}_\bullet, \tag{2}$$

for testing H_0, where $\boldsymbol{T} = \boldsymbol{H}'(\boldsymbol{H}\boldsymbol{H}')^+\boldsymbol{H}$, $\mathbf{v}^\top\mathcal{X} = \{\mathbf{v}^\top\mathcal{X}_{11},\ldots,\mathbf{v}^\top\mathcal{X}_{an_a}\}$, and $\boldsymbol{A}^+$ denotes the Moore-Penrose inverse of the matrix $\boldsymbol{A}$. Since its asymptotic $\chi^2_{rank(\boldsymbol{T})}$ null distribution provides a poor finite sample approximation, they propose the following asymptotic model-based bootstrap approach: Given the data $\mathbf{v}^\top\mathcal{X}$ let $\mathbf{v}^\top\mathcal{X}_{ik}^\star \sim N(\mathbf{0},\widehat{\boldsymbol{V}}_i), i = 1,\ldots,a, k = 1,\ldots,n_i$, be independent random vectors that are used for recalculating the test statistic as $T_N^\star = T_N(\mathbf{v}^\top\mathcal{X}^\star)$, where $\mathbf{v}^\top\mathcal{X}^\star = \{\mathbf{v}^\top\mathcal{X}_{11}^\star,\ldots,\mathbf{v}^\top\mathcal{X}_{an_a}^\star\}$. Denoting by $c^\star$ the corresponding $(1-\alpha)$-quantile of the (conditional) distribution of $T_N^\star$ the test rejects H_0 if $T_N > c^\star$. The validity of this procedure (also named parametric bootstrap WTS) is proven in Konietschke et al. (2015).

This procedure is not only applicable for MANOVA but also for RM designs (Bathke et al., 2018). However, Friedrich et al. (2017a) proposed a more favourable technique for Repeated Measurements. It is based on an at first blush chaotic resampling method: Wild permutation of all pooled components without taking group membership or possible dependencies into account. Denoting the resulting permuted data set as $\mathbf{v}^\top\mathcal{X}^\pi$ their permutation test for RM models rejects H_0 if $T_N > c^\pi$. Here c^π denotes the $(1-\alpha)$-quantile of the (conditional) distribution of the permutation version of the test statistic $T_N^\pi = T_N(\mathbf{v}^\top\mathcal{X}^\pi)$. As shown in extensive simulations in Friedrich et al. (2017a) and the corresponding supporting information this 'wild' permutation WTS method controls the type-I error rate very well. Note that this procedure is only applicable for RM due to the commensurate nature of their components. In MANOVA set-ups the permutation would stir different scalings making comparisons meaningless.

In addition to these WTS procedures two other statistics are considered as well. For RM the

well-established ANOVA-type statistic (ATS)

$$Q_N = N\overline{\boldsymbol{X}}_\bullet' \boldsymbol{T} \overline{\boldsymbol{X}}_\bullet \tag{3}$$

by Brunner (2001) is implemented together with the enhanced F-approximation of the statistic proposed in Brunner et al. (1997, 2002) and implemented in the SAS `PROC Mixed` procedure. Although known to be rather conservative it has the advantage (over the WTS) of being applicable in case of eventually singular covariance matrices $\boldsymbol{V}_i$ or $\widehat{\boldsymbol{V}}_i$ since it waives the Moore-Penrose inverse involved in Equation 2.

Similar to the permuted WTS the ATS given in Equation 3 is only applicable for RM since it is not invariant under scale transformations (e.g., change of units) of the univariate components. To nevertheless provide a robust method for MANOVA settings which is also applicable in case of singular $\boldsymbol{V}_i$ or $\widehat{\boldsymbol{V}}_i$, Friedrich and Pauly (2018) have recently proposed the novel MATS (modified ATS)

$$M_N = M_N(\mathbf{v}^\top \mathcal{X}) = N\overline{\boldsymbol{X}}_\bullet' \boldsymbol{T} (\boldsymbol{T}\widehat{\boldsymbol{D}}_N \boldsymbol{T})^+ \boldsymbol{T}\overline{\boldsymbol{X}}_\bullet.$$

Here, the involved diagonal matrix $\widehat{\boldsymbol{D}}_N = \oplus_{1\le i\le a, 1\le s\le d} N\widehat{\sigma}_{is}^2/n_i$ of the empirical variances $\widehat{\sigma}_{is}^2$ of component s in group i, deduces an invariance under component-wise scale transformations of the MATS for null hypotheses as described in Section Special designs and hypotheses, i.e., of the form $\boldsymbol{T} = \boldsymbol{M} \otimes \boldsymbol{I}_d$, see Friedrich and Pauly (2018) for details. To obtain an accurate finite sample performance, it is also equipped with an asymptotic model based bootstrap approach. That is, MATS rejects H_0 if $M_N > \tilde{c}^\star$, where $\tilde{c}^\star$ is the $(1-\alpha)$-quantile of the (conditional) distribution of the bootstrapped statistic $M_N^\star = M_N(\mathbf{v}^\top \mathcal{X}^\star)$. In addition, we implemented a wild bootstrap approach, which is based on multiplying the centered data vectors $(\mathbf{v}^\top \mathcal{X}_{ik} - \mathbf{v}^\top \mathcal{X}_{i\cdot})$ with random weights W_{ik} fulfilling $E(W_{ik}) = 0, \mathrm{var}(W_{ik}) = 1$ and $\sup_{i,k} E(W_{ik}^4) < \infty$. In the package, we implemented Rademacher distributed weights, i.e., random signs. Extensive simulations in Friedrich and Pauly (2018) not only confirm its applicability in case of singular covariance matrices but also disclose a very robust behaviour that even seems to be advantageous over the parametrically bootstrapped WTS of Konietschke et al. (2015). However, both procedures, as well as the 'usual' asymptotic WTS are displayed within the MANOVA functions. All of the aforementioned procedures are applicable in various factorial designs in a unified way, i.e., when more than one factor may impact the response. The specific models and the hypotheses being tested will be discussed in the next section.

Special designs and hypotheses

In order to provide a general overview of different statistical designs and layouts that can be analyzed with **MANOVA.RM** we exemplify few designs that occur frequently in practical applications and discuss the model, hypotheses and limitations. All of the methods implemented in **MANOVA.RM** are even applicable in higher-way layouts than being presented here; and the list should not be seen as the limited application of the package. The models are derived by sub-indexing the index i in Equation 1 in the following ways:

- **One-Way (A):** Writing $\boldsymbol{\mu}_i = \boldsymbol{\nu} + \boldsymbol{\alpha}_i$ we have $\mathbf{v}^\top \mathcal{X}_{ik} = \boldsymbol{\nu} + \boldsymbol{\alpha}_i + \boldsymbol{\epsilon}_{ik}$ with $\sum_{i=1}^a \boldsymbol{\alpha}_i = \mathbf{0}$ and obtain the null hypothesis of 'no group' or 'factor A' effect as

$$\begin{aligned} H_0(A) : \{(\boldsymbol{P}_a \otimes \boldsymbol{I}_d)\boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\mu}_1 = \cdots = \boldsymbol{\mu}_a\} \\ &= \{\boldsymbol{\alpha}_1 = \cdots = \boldsymbol{\alpha}_a = \mathbf{0}\}. \end{aligned}$$

  In case of $a = 2$ this includes the famous *multivariate Behrens-Fisher problem* as, e.g., analyzed in Yao (1965); Nel and Van der Merwe (1986); Christensen and Rencher (1997); Krishnamoorthy and Yu (2004) or Yanagihara and Yuan (2005).

- **Crossed Two-Way ($A \times B$):** Splitting the index into two and writing $\boldsymbol{\mu}_{ij} = \boldsymbol{\nu} + \boldsymbol{\alpha}_i + \boldsymbol{\beta}_j + \boldsymbol{\gamma}_{ij}$ we obtain the model $\mathbf{v}^\top \mathcal{X}_{ijk} = \boldsymbol{\nu} + \boldsymbol{\alpha}_i + \boldsymbol{\beta}_j + \boldsymbol{\gamma}_{ij} + \boldsymbol{\epsilon}_{ijk}$, $1 \le i \le a$, $1 \le j \le b$, $1 \le k \le n_{ij}$ with $\sum_i \boldsymbol{\alpha}_i = \sum_j \boldsymbol{\beta}_j = \sum_i \boldsymbol{\gamma}_{ij} = \sum_j \boldsymbol{\gamma}_{ij} = \mathbf{0}$. The corresponding null hypotheses of no main effects in A or B and no interaction effect between A and B can be written as:

$$\begin{aligned} H_0(A) : \{(\boldsymbol{P}_a \otimes b^{-1}\boldsymbol{J}_b \otimes \boldsymbol{I}_d)\,\boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\alpha}_1 = \cdots = \boldsymbol{\alpha}_a = \mathbf{0}\}, \\ H_0(B) : \{(a^{-1}\boldsymbol{J}_a \otimes \boldsymbol{P}_b \otimes \boldsymbol{I}_d)\,\boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\beta}_1 = \cdots = \boldsymbol{\beta}_b = \mathbf{0}\}, \\ H_0(AB) : \{(\boldsymbol{P}_a \otimes \boldsymbol{P}_b \otimes \boldsymbol{I}_d)\,\boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\gamma}_{11} = \cdots = \boldsymbol{\gamma}_{ab} = \mathbf{0}\}. \end{aligned}$$

  Note that the interpretation of main effects is complicated by the presence of significant interaction effects and further analyses are necessary to determine the direction of the effects.

- **Hierarchically nested Two-Way ($B(A)$):** A fixed subcategory B within factor A can be introduced via the model $\mathbf{v}^\top \mathcal{X}_{ijk} = \nu + \boldsymbol{\alpha}_i + \boldsymbol{\beta}_{j(i)} + \boldsymbol{\epsilon}_{ijk}, 1 \leq i \leq a,\ 1 \leq j \leq b_i\ , 1 \leq k \leq n_{ij}$ with $\sum_i \boldsymbol{\alpha}_i = \sum_j \boldsymbol{\beta}_{j(i)} = \mathbf{0}$. Here, the hypotheses of no main effect A or no sub-category main effect B can be written as

$$\begin{aligned} H_0(A) : \{(\boldsymbol{P}_a \widetilde{\boldsymbol{J}}_b \otimes \boldsymbol{I}_d)\, \boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\alpha}_1 = \cdots = \boldsymbol{\alpha}_a = \mathbf{0}\}, \\ H_0(B(A)) : \{(\widetilde{\boldsymbol{P}}_b \otimes \boldsymbol{I}_d)\, \boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\beta}_{j(i)} = 0 \forall\ 1 \leq i \leq a, 1 \leq j \leq b_i\} \end{aligned}$$

with $\widetilde{\boldsymbol{P}}_b = \bigoplus_{j=1}^{a} \boldsymbol{P}_{b_j}$, $\widetilde{\boldsymbol{J}}_b = \bigoplus_{j=1}^{a} b_j^{-1} \mathbf{1}'_{b_j}$ and $\boldsymbol{\mu} := (\boldsymbol{\mu}'_{11}, ..., \boldsymbol{\mu}'_{1b_1}, \boldsymbol{\mu}'_{21}, ..., \boldsymbol{\mu}'_{2b_2}, ..., \boldsymbol{\mu}'_{ab_a})'$.
We only implemented balanced designs, i.e., $b_i = b$ for all $i = 1, \ldots, a$. Hierarchically nested three-way designs or arbitrary crossed higher-way layouts can be introduced similarly and are implemented as well.

- **Repeated Measures and Split Plot Designs** are covered by setting $d = t$, where even hypotheses about within subject factors can be formulated. We exemplify this for profile analyses in the special case of a one-sample RM design with $a = 1$

$$H_0(\text{Time}) : \{\boldsymbol{P}_t\, \boldsymbol{\mu} = \mathbf{0}\} = \{\mu_{11} = \cdots = \mu_{1t}\},$$

as well as for a two-sample RM design with $a = 2$:

$$\begin{aligned} H_0(\text{Parallel}) : \{\boldsymbol{T}_P\, \boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 = \gamma\, \mathbf{1}_t \ \text{ for some } \ \gamma \in \mathbb{R}\} \\ H_0(\text{Flat}) : \{\boldsymbol{T}_F\, \boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\mu}_{1s} + \boldsymbol{\mu}_{2s} = \bar{\boldsymbol{\mu}}_{1\cdot} + \bar{\boldsymbol{\mu}}_{2\cdot} \ \text{ for all } \ s\} \\ H_0(\text{Identical}) : \{\boldsymbol{T}_I\, \boldsymbol{\mu} = \mathbf{0}\} &= \{\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2\} \end{aligned}$$

with $\boldsymbol{T}_F = \boldsymbol{P}_t\, (\boldsymbol{I}_t \quad \boldsymbol{I}_t)$, $\boldsymbol{T}_P = (\mathbf{1}_{t-1} \quad -\boldsymbol{I}_{t-1})\, \boldsymbol{T}_I$ and $\boldsymbol{T}_I = (\boldsymbol{I}_t \quad -\boldsymbol{I}_t)$.

parallel flat identical

Note, that we could also employ more complex factorial structures on the repeated measurements (i.e., more within subject factors) by splitting up the index j.

Examples

To demonstrate the use of the `RM` and the `MANOVA` function, we provide several examples for both repeated measures designs and multivariate data in the following. Furthermore, the **MANOVA.RM** package is equipped with an optional GUI (graphical user interface), based on **RGtk2** (Lawrence and Temple Lang, 2010), which will be explained in detail in Section Graphical user interface below.

Both functions are structured similarly: The main input parameters are the `formula` specifying the outcome on the left hand side and the factor variables of interest on the right, the `data` and the `resampling` approach. The latter varies according to the design: the user can choose between a parametric and a wild bootstrap and in the RM design, additionally a permutation approach for the WTS is implemented.

Repeated Measures Designs

The function `RM` is built as follows:

```
R> RM(formula, data, subject, no.subf = 1, iter = 10000, alpha = 0.05,
+     resampling = "Perm", CPU, seed, CI.method = "t-quantile", dec = 3)
```

Data need to be provided in long format, i.e., one row per measurement. Here, `subject` specifies the column name of the subjects variable in the data frame, while `no.subf` denotes the number of within subject factors considered. Note that in a setting with more than one between subjects factor, the subject ids in the different groups need to be different. Otherwise the program will erroneously assume that these measurements belong to the same subject. The number of cores used for parallel computing as well as a random seed can optionally be specified using `CPU` and `seed`, respectively. For calculating the confidence intervals, the user can choose between t-quantiles (the default) and the quantiles based on the resampled WTS. The results are rounded to `dec` digits.

The function RM returns an object of class "RM" from which the user may obtain plots and summaries of the results using plot(), print() and summary(), respectively. Here, print() returns a short summary of the results, i.e., the values of the test statistics along with degrees of freedom and corresponding p values whereas summary() also displays some descriptive statistics such as the means, sample sizes and confidence intervals for the different factor level combinations. Plotting is based on **plotrix** (Lemon, 2006). For two- and higher-way layouts, the factors for plotting can be additionally specified in the plot call, see the examples below.

Example 1: One between subject and two within subject factors

For illustration purposes, we consider the data set o2cons, which is included in **MANOVA.RM**. This data set contains measurements of the oxygen consumption of leukocytes in the presence and absence of inactivated staphylococci at three consecutive time points. Due to the study design, both time and staphylococci are within subject factors while the treatment (Verum vs. Placebo) is a between subject factor (see Friedrich et al., 2017a, for more details).

```
R> data("o2cons")
R> model1 <- RM(O2 ~ Group * Staphylococci * Time, data = o2cons,
+               subject = "Subject",  no.subf = 2, iter = 1000,
+               resampling = "Perm", seed = 1234)
R> summary(model1)

Call:
O2 ~ Group * Staphylococci * Time
A repeated measures analysis with  2 within-subject and  1 between-subject factors.

Descriptive:
   Group Staphylococci Time  n Means Lower 95 % CI Upper 95 % CI
1      P             0    6 12 1.322         1.150         1.493
5      P             0   12 12 2.430         2.196         2.664
9      P             0   18 12 3.425         3.123         3.727
3      P             1    6 12 1.618         1.479         1.758
7      P             1   12 12 2.434         2.164         2.704
11     P             1   18 12 3.527         3.273         3.781
2      V             0    6 12 1.394         1.201         1.588
6      V             0   12 12 2.570         2.355         2.785
10     V             0   18 12 3.677         3.374         3.979
4      V             1    6 12 1.656         1.471         1.840
8      V             1   12 12 2.799         2.500         3.098
12     V             1   18 12 4.029         3.802         4.257

Wald-Type Statistic (WTS):
                   Test statistic       df    p-value
Group                   "11.167"       "1" "0.001"
Staphylococci           "20.401"       "1" "<0.001"
Group:Staphylococci     "2.554"        "1" "0.11"
Time                    "4113.057"     "2" "<0.001"
Group:Time              "24.105"       "2" "<0.001"
Staphylococci:Time      "4.334"        "2" "0.115"
Group:Staphylococci:Time "4.303"       "2" "0.116"

ANOVA-Type Statistic (ATS):
                    Test statistic     df1      df2       p-value
Group                   "11.167"       "1"     "316.278" "0.001"
Staphylococci           "20.401"       "1"     "Inf"     "<0.001"
Group:Staphylococci     "2.554"        "1"     "Inf"     "0.11"
Time                    "960.208"      "1.524" "Inf"     "<0.001"
Group:Time              "5.393"        "1.524" "Inf"     "0.009"
Staphylococci:Time      "2.366"        "1.983" "Inf"     "0.094"
Group:Staphylococci:Time "2.147"       "1.983" "Inf"     "0.117"

p-values resampling:
                     Perm (WTS)
Group                   "0.005"
```

```
Staphylococci             "0.001"
Group:Staphylococci       "0.145"
Time                      "<0.001"
Group:Time                "<0.001"
Staphylococci:Time        "0.144"
Group:Staphylococci:Time "0.139"
```

The output consists of four parts: `model1$Descriptive` gives an overview of the descriptive statistics: The number of observations, mean and confidence intervals are displayed for each factor level combination. Second, `model1$WTS` contains the results for the Wald-type test: The test statistic, degree of freedom and p values based on the asymptotic χ^2-distribution are displayed. Note that the χ^2-approximation is highly anti-conservative for small sample sizes, cf. Konietschke et al. (2015); Friedrich et al. (2017a). The corresponding results based on the ATS are contained within `model1$ATS`. This test statistic tends to rather conservative decisions in case of small sample sizes and is even asymptotically only an approximation, thus not providing an asymptotic level α test, see Brunner (2001); Friedrich et al. (2017a). Finally, `model1$resampling` contains the p values based on the chosen resampling approach. For the ATS, the permutation approach is not feasible since it would result in an incorrect covariance structure, and is therefore not implemented. Due to the above mentioned issues for small sample sizes, the respective resampling procedure is recommended for such situations.

In this example, we find significant effects of all factors as well as a significant interaction between group and time.

Example 2: Two within subject and two between subject factors

We consider the data set `EEG` from the **MANOVA.RM** package: At the Department of Neurology, University Clinic of Salzburg, 160 patients were diagnosed with either Alzheimer's Disease (AD), mild cognitive impairment (MCI), or subjective cognitive complaints without clinically significant deficits (SCC), based on neuropsychological diagnostics (Bathke et al., 2018). This data set contains z-scores for brain rate and Hjorth complexity, each measured at frontal, temporal and central electrode positions and averaged across hemispheres. In addition to standardization, complexity values were multiplied by -1 in order to make them more easily comparable to brain rate values: For brain rate we know that the values decrease with age and pathology, while Hjorth complexity values are known to increase with age and pathology. The three between subject factors considered were sex (men vs. women), diagnosis (AD vs. MCI vs. SCC), and age (< 70 vs. ≥ 70 years). Additionally, the within subject factors region (frontal, temporal, central) and feature (brain rate, complexity) structure the response vector.

Note that due to the small number of subjects in some groups (e.g., only 2 male patients aged < 70 were diagnosed with AD) we restrict our analyses to two between subject factors at a time. However, more complex factorial designs can also be analyzed with **MANOVA.RM** as outlined above.

```
R> data("EEG")
R> EEG_model <- RM(resp ~ sex * diagnosis * feature * region, data = EEG,
+                  subject = "id", no.subf = 2, resampling = "WildBS",
+                  iter = 1000, alpha = 0.01, CPU = 4, seed = 123)
R> summary(EEG_model)

Call:
resp ~ sex * diagnosis * feature * region
A repeated measures analysis with  2 within-subject and  2 between-subject factors.

Descriptive:
 sex diagnosis    feature   region   n  Means    Lower 99 % CI  Upper 99 % CI
1    M        AD  brainrate  central 12 -1.010        -4.881         2.861
13   M        AD  brainrate  frontal 12 -1.007        -4.991         2.977
25   M        AD  brainrate temporal 12 -0.987        -4.493         2.519
7    M        AD complexity  central 12 -1.488       -10.053         7.077
19   M        AD complexity  frontal 12 -1.086        -6.906         4.735
31   M        AD complexity temporal 12 -1.320        -7.203         4.562
3    M       MCI  brainrate  central 27 -0.447        -1.591         0.696
15   M       MCI  brainrate  frontal 27 -0.464        -1.646         0.719
27   M       MCI  brainrate temporal 27 -0.506        -1.584         0.572
9    M       MCI complexity  central 27 -0.257        -1.139         0.625
21   M       MCI complexity  frontal 27 -0.459        -1.997         1.079
33   M       MCI complexity temporal 27 -0.490        -1.796         0.816
```

```
5    M       SCC  brainrate  central 20  0.459        -0.414         1.332
17   M       SCC  brainrate  frontal 20  0.243        -0.670         1.156
29   M       SCC  brainrate temporal 20  0.409        -1.210         2.028
11   M       SCC complexity  central 20  0.349        -0.070         0.767
23   M       SCC complexity  frontal 20  0.095        -1.037         1.227
35   M       SCC complexity temporal 20  0.314        -0.598         1.226
2    W        AD  brainrate  central 24 -0.294        -1.978         1.391
14   W        AD  brainrate  frontal 24 -0.159        -1.813         1.495
26   W        AD  brainrate temporal 24 -0.285        -1.776         1.206
8    W        AD complexity  central 24 -0.128        -1.372         1.116
20   W        AD complexity  frontal 24  0.026        -1.212         1.264
32   W        AD complexity temporal 24 -0.194        -1.670         1.283
4    W       MCI  brainrate  central 30 -0.106        -1.076         0.863
16   W       MCI  brainrate  frontal 30 -0.074        -1.032         0.885
28   W       MCI  brainrate temporal 30 -0.069        -1.064         0.925
10   W       MCI complexity  central 30  0.094        -0.464         0.652
22   W       MCI complexity  frontal 30  0.131        -0.768         1.031
34   W       MCI complexity temporal 30  0.121        -0.652         0.895
6    W       SCC  brainrate  central 47  0.537        -0.049         1.124
18   W       SCC  brainrate  frontal 47  0.548        -0.062         1.159
30   W       SCC  brainrate temporal 47  0.559        -0.015         1.133
12   W       SCC complexity  central 47  0.384         0.110         0.659
24   W       SCC complexity  frontal 47  0.403        -0.038         0.845
36   W       SCC complexity temporal 47  0.506         0.132         0.880

Wald-Type Statistic (WTS):
                        Test statistic     df  p-value
sex                           "9.973"       "1" "0.002"
diagnosis                     "42.383"      "2" "<0.001"
sex:diagnosis                 "3.777"       "2" "0.151"
feature                       "0.086"       "1" "0.769"
sex:feature                   "2.167"       "1" "0.141"
diagnosis:feature             "5.317"       "2" "0.07"
sex:diagnosis:feature         "1.735"       "2" "0.42"
region                        "0.07"        "2" "0.966"
sex:region                    "0.876"       "2" "0.645"
diagnosis:region              "6.121"       "4" "0.19"
sex:diagnosis:region          "1.532"       "4" "0.821"
feature:region                "0.652"       "2" "0.722"
sex:feature:region            "0.423"       "2" "0.81"
diagnosis:feature:region      "7.142"       "4" "0.129"
sex:diagnosis:feature:region "2.274"        "4" "0.686"

ANOVA-Type Statistic (ATS):
                      Test statistic       df1     df2       p-value
sex                           "9.973"       "1"     "657.416" "0.002"
diagnosis                     "13.124"      "1.343" "657.416" "<0.001"
sex:diagnosis                 "1.904"       "1.343" "657.416" "0.164"
feature                       "0.086"       "1"     "Inf"     "0.769"
sex:feature                   "2.167"       "1"     "Inf"     "0.141"
diagnosis:feature             "1.437"       "1.562" "Inf"     "0.238"
sex:diagnosis:feature         "1.031"       "1.562" "Inf"     "0.342"
region                        "0.018"       "1.611" "Inf"     "0.965"
sex:region                    "0.371"       "1.611" "Inf"     "0.644"
diagnosis:region              "1.091"       "2.046" "Inf"     "0.337"
sex:diagnosis:region          "0.376"       "2.046" "Inf"     "0.691"
feature:region                "0.126"       "1.421" "Inf"     "0.81"
sex:feature:region            "0.077"       "1.421" "Inf"     "0.864"
diagnosis:feature:region      "0.829"       "1.624" "Inf"     "0.415"
sex:diagnosis:feature:region "0.611"        "1.624" "Inf"     "0.51"

p-values resampling:
                        WildBS (WTS)   WildBS (ATS)
sex                           "<0.001"     "<0.001"
```

```
diagnosis                    "<0.001"     "<0.001"
sex:diagnosis                "0.119"      "0.124"
feature                      "0.798"      "0.798"
sex:feature                  "0.152"      "0.152"
diagnosis:feature            "0.067"      "0.249"
sex:diagnosis:feature        "0.445"      "0.362"
region                       "0.967"      "0.98"
sex:region                   "0.691"      "0.728"
diagnosis:region             "0.182"      "0.338"
sex:diagnosis:region         "0.863"      "0.814"
feature:region               "0.814"      "0.926"
sex:feature:region           "0.881"      "0.951"
diagnosis:feature:region     "0.098"      "0.519"
sex:diagnosis:feature:region "0.764"      "0.683"
```

We find significant effects at level $\alpha = 0.01$ of the between subject factors sex and diagnosis, while none of the within subject factors or interactions become significant.

Plotting

The `RM()` function is equipped with a plotting option, displaying the calculated means along with $(1 - \alpha)$ confidence intervals based on t-quantiles. The plot function takes an `RM` object as argument. In addition, the factor of interest may be specified. If this argument is omitted in a two- or higher-way layout, the user is asked to specify the factor for plotting. Furthermore, additional graphical parameters can be used to customize the plots. The optional argument `legendpos` specifies the position of the legend in higher-way layouts, whereas `gap` (default 0.1) is the distance introduced between error bars in a higher-way layout. Additionally, the parameter `CI.info` can be set to `TRUE` in order to output the means and confidence intervals for the desired interaction.

```
R> plot(EEG_model, factor = "sex", main = "Effect of sex on EEG values")
R> plot(EEG_model, factor = "sex:diagnosis", legendpos = "topleft",
+       col = c(4, 2),  ylim = c(-1.8, 0.8), CI.info = TRUE)

$mean
     AD         MCI       SCC
M -1.1496245 -0.43724352 0.3114978
W -0.1723434  0.01624054 0.4897902

$lower
     AD        MCI       SCC
M -1.6714757 -0.6251940 0.1732354
W -0.3874841 -0.1290226 0.3848516

$upper
      AD        MCI       SCC
M -0.62777339 -0.2492930 0.4497601
W  0.04279732  0.1615037 0.5947288

R> plot(EEG_model, factor = "sex:diagnosis:feature",
+       legendpos = "bottomright", gap = 0.05)
```

The resulting plots are displayed in Figure 1 and Figure 2, respectively.

MANOVA Design

For the analysis of multivariate data, the functions `MANOVA` and `MANOVA.wide` are implemented. The difference between the two functions is that the response must be stored in long and wide format for using `MANOVA` or `MANOVA.wide`, respectively. The structure of both functions is very similar. They both calculate the WTS for multivariate data in a design with crossed or nested factors. Additionally, the modified ANOVA-type statistic (MATS) is calculated which has the additional advantage of being applicable to designs involving singular covariance matrices and is invariant under scale transformations of the data (Friedrich and Pauly, 2018). The resampling methods provided are a parametric bootstrap approach and a wild bootstrap using Rademacher weights. Note that only balanced nested designs (i.e., the same number of factor levels b for each level of the factor A) with

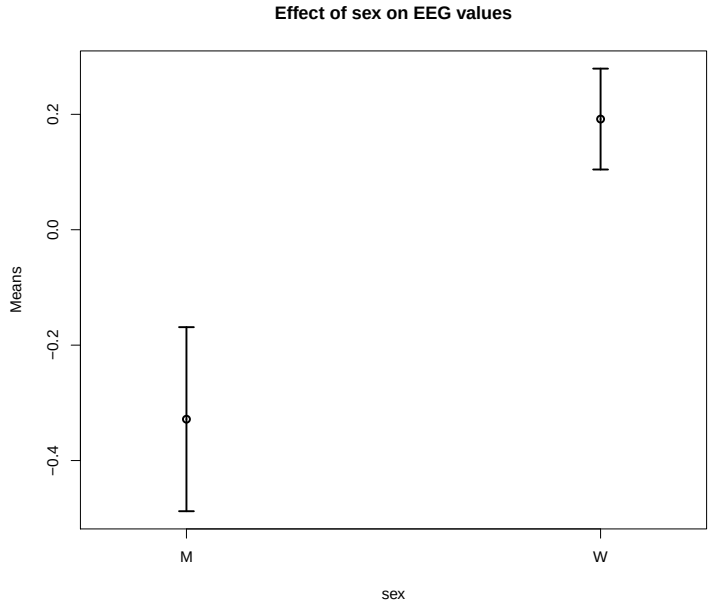


Figure 1: Plot for factor "sex" in the RM model of the EEG data example.

up to three factors are implemented. Designs involving both crossed and nested factors are not implemented. Note that in nested designs, the levels of the nested factor usually have the same labels for all levels of the main factor, i.e., for each level $i = 1, \ldots, a$ of the main factor A the nested factor levels are labeled as $j = 1, \ldots, b_i$. If the levels of the nested factor are named uniquely, this has to be specified by setting the parameter `nested.levels.unique` to `TRUE`.

```
R> MANOVA(formula, data, subject, iter = 10000, alpha = 0.05,
+         resampling = "paramBS", CPU, seed,
+         nested.levels.unique = FALSE, dec = 3)
R> MANOVA.wide(formula, data, iter = 10000, alpha = 0.05,
+              resampling = "paramBS", CPU, seed,
+              nested.levels.unique = FALSE, dec = 3)
```

The only difference between `MANOVA` and `MANOVA.wide` in the function call except from the different shape of the formula (see examples below) is the `subject` variable, which needs to be specified for `MANOVA` only.

Data Example MANOVA: Two crossed factors

We again consider the data set `EEG` from the **MANOVA.RM** package, but now we ignore the within subject factor structure. Therefore, we are now in a multivariate setting with 6 measurements per patient and three crossed factors sex, age and diagnosis. Due to the small number of subjects in some groups we restrict our analyses to two factors at a time. The analysis of this example is shown below.

```
R> data("EEG")
R> EEG_MANOVA <- MANOVA(resp ~ sex * diagnosis, data = EEG, subject = "id",
+                       resampling = "paramBS", iter = 1000,  alpha = 0.01,
+                       CPU = 1, seed = 987)
R> summary(EEG_MANOVA)

Call:
resp ~ sex * diagnosis

Descriptive:
   sex diagnosis n   Mean 1 Mean 2 Mean 3 Mean 4 Mean 5 Mean 6
1   M        AD 12 -0.987 -1.007 -1.010 -1.320 -1.086 -1.488
3   M       MCI 27 -0.506 -0.464 -0.447 -0.490 -0.459 -0.257
5   M       SCC 20  0.409  0.243  0.459  0.314  0.095  0.349
2   W        AD 24 -0.285 -0.159 -0.294 -0.194  0.026 -0.128
```

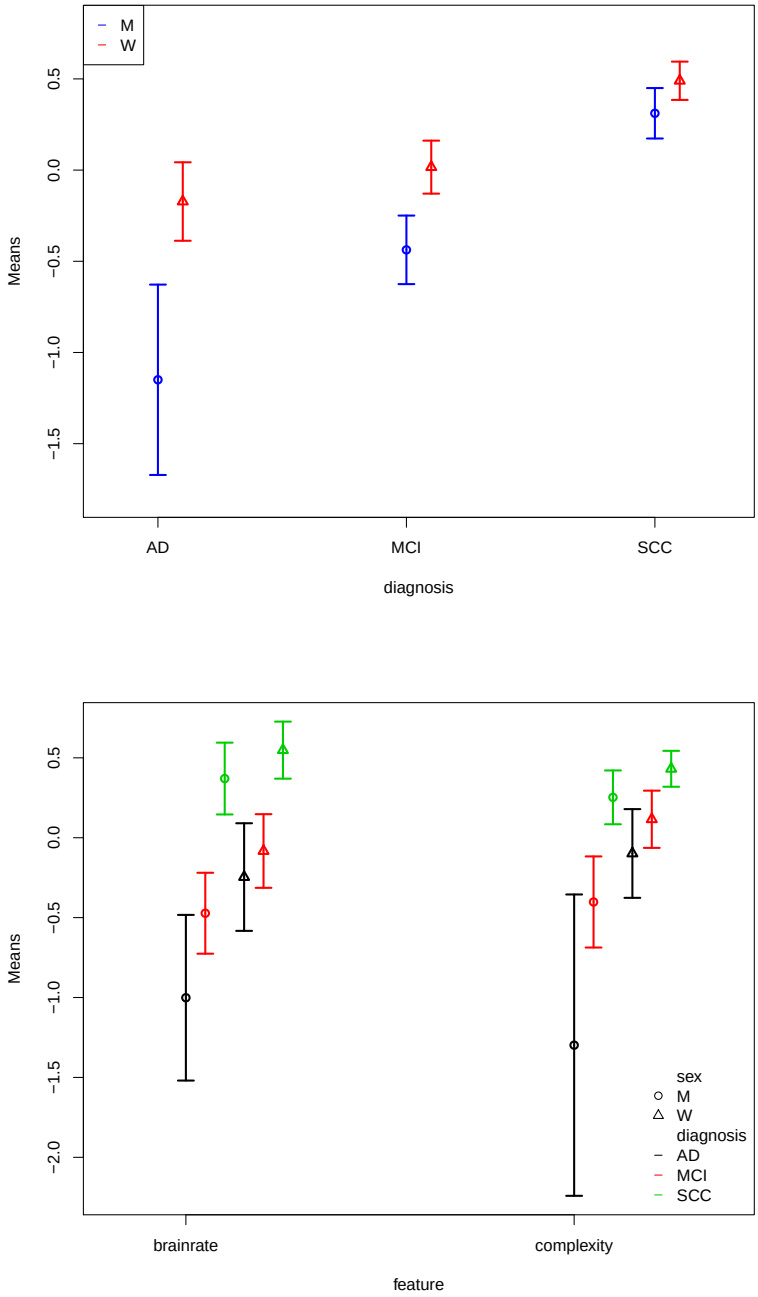


Figure 2: Plot for the interaction between "sex" and "diagnosis" (upper panel) as well as additionally taking "feature" into account (lower panel) in the RM model of the EEG data example.

```
4   W       MCI 30 -0.069 -0.074 -0.106  0.121  0.131  0.094
6   W       SCC 47  0.559  0.548  0.537  0.506  0.403  0.384

Wald-Type Statistic (WTS):
             Test statistic   df   p-value
sex           "12.604"       "6"  "0.05"
diagnosis     "55.158"       "12" "<0.001"
sex:diagnosis "9.79"         "12" "0.634"

modified ANOVA-Type Statistic (MATS):
               Test statistic
sex                  45.263
diagnosis           194.165
sex:diagnosis        18.401

p-values resampling:
        paramBS (WTS)  paramBS (MATS)
sex           "0.124"       "0.003"
diagnosis     "<0.001"      "<0.001"
sex:diagnosis "0.748"       "0.223"
```

The output consists of several parts: First, some descriptive statistics of the data set are displayed, namely the sample size and mean for each factor level combination and each dimension (dimensions occur in the same order as in the original data set). In this example, Mean 1 to Mean 3 correspond to the brainrate (temporal, frontal, central) while Mean 4–6 correspond to complexity. Second, the results based on the WTS are displayed. For each factor, the test statistic, degree of freedom and p value is given. For the MATS, only the value of the test statistic is given, since here inference is only based on resampling. The resampling-based p values are finally displayed for both test statistics.

To demonstrate the use of the `MANOVA.wide()` function, we consider the same data set in wide format, which is also included in the package. In the formula argument, the user now needs to specify the variables of interest bound together via `cbind`. A subject variable is no longer necessary, as every row of the data set belongs to one patient in wide format data. The output is almost identically to the one obtained from `MANOVA` with the difference that the mean values are now labeled according to the variable names supplied in the `formula` argument.

```
R> data("EEGwide")
R> EEG_wide <- MANOVA.wide(cbind(brainrate_temporal, brainrate_frontal,
+                brainrate_central, complexity_temporal,
+                complexity_frontal, complexity_central) ~ sex * diagnosis,
+                 data = EEGwide, resampling = "paramBS",  iter = 1000,
+                  alpha = 0.01, CPU = 1, seed = 987)
R> summary(EEG_wide)

Call:
cbind(brainrate_temporal, brainrate_frontal, brainrate_central,
complexity_temporal, complexity_frontal, complexity_central) ~
sex * diagnosis

Descriptive:
sex diagnosis  n brainrate_temporal  brainrate_frontal  brainrate_central
1   M        AD 12             -0.987             -1.007             -1.010
2   W        AD 27             -0.506             -0.464             -0.447
3   M       MCI 20              0.409              0.243              0.459
4   W       MCI 24             -0.285             -0.159             -0.294
5   M       SCC 30             -0.069             -0.074             -0.106
6   W       SCC 47              0.559              0.548              0.537
complexity_temporal  complexity_frontal  complexity_central
1      -1.320            -1.086              -1.488
2      -0.490            -0.459              -0.257
3       0.314             0.095               0.349
4      -0.194             0.026              -0.128
5       0.121             0.131               0.094
6       0.506             0.403               0.384

Wald-Type Statistic (WTS):
```

```
         Test statistic    df   p-value
sex           "12.604"       "6"  "0.05"
diagnosis     "55.158"       "12" "<0.001"
sex:diagnosis "9.79"         "12" "0.634"

modified ANOVA-Type Statistic (MATS):
              Test statistic
sex                   45.263
diagnosis            194.165
sex:diagnosis         18.401

p-values resampling:
        paramBS (WTS)  paramBS (MATS)
sex           "0.122"       "0.005"
diagnosis     "<0.001"      "<0.001"
sex:diagnosis "0.742"       "0.21"
```

In this example, MATS detects a significant effect of sex, a finding that is not shared by the p value based on the parametric bootstrap WTS.

Confidence Regions

The `MANOVA` functions are equipped with a function for calculating and plotting of confidence regions. Details on the methods can be found in Friedrich and Pauly (2018). We would like to point out, however, that the MATS-based confidence regions have to be interpreted differently from the more usual WTS-based ones. For the latter, the WTS is compared to a fixed critical value (in the asymptotic choice from a χ^2-distribution) and we thus obtain geometric ellipsoids as the WTS is more or less a Mahalanobis distance in the inverse covariance matrix. The MATS statistic only involves the variances of the covariance matrix and we thus obtain a different geometric shape of the corresponding confidence region. However, here the correlation is implicitly involved in the critical value which now (different to the WTS) depends on the covariance matrix. More precisely, a confidence region for the vector of contrasts $\boldsymbol{H\mu}$ based on the MATS is determined by the set of all $\boldsymbol{H\mu}$ such that

$$N(\boldsymbol{H}\overline{\boldsymbol{X}}_{\cdot} - \boldsymbol{H\mu})^{\top}(\boldsymbol{H}\widehat{\boldsymbol{D}}_N\boldsymbol{H}^{\top})^{+}(\boldsymbol{H}\overline{\boldsymbol{X}}_{\cdot} - \boldsymbol{H\mu}) \leq c^*_{1-\alpha},$$

where $c^*_{1-\alpha}$ denotes the quantile of the respective resampling distribution. A confidence ellipsoid is now obtained based on the eigenvalues $\widehat{\lambda}_s$ and eigenvectors $\widehat{\boldsymbol{e}}_s$ of $\boldsymbol{H}\widehat{\boldsymbol{D}}_N\boldsymbol{H}^{\top}$. The classical WTS-based confidence ellipsoids, in contrast, are based on eigenvectors and eigenvalues of $\boldsymbol{H}\widehat{\boldsymbol{\Sigma}}_N\boldsymbol{H}^{\top}$ instead. Confidence regions can be calculated using the `conf.reg` function. Note that confidence regions can only be plotted in designs with 2 dimensions.

```
R> conf.reg(object, nullhypo)
```

`Object` must be an object of class `"MANOVA"`, i.e., created using either `MANOVA` or `MANOVA.wide`, whereas `nullhypo` specifies the desired null hypothesis, i.e., the contrast of interest in designs involving more than one factor. As an example, we consider the data set water from the **HSAUR** package (Everitt and Hothorn, 2017). The data set contains measurements of mortality and drinking water hardness for 61 cities in England and Wales. Suppose we want to analyse whether these measurements differ between northern and southern towns. Since the data set is in wide format, we need to use the `MANOVA.wide` function.

```
R> library("HSAUR")
R> data("water")
R> test <- MANOVA.wide(cbind(mortality, hardness) ~ location, data = water,
+                iter = 1000, resampling = "paramBS", CPU = 1, seed = 123)
R> summary(test)
R> cr <- conf.reg(test)
R> cr
R> plot(cr, xlab = "Difference in mortality",
+       ylab ="Difference in water hardness")

Call:
cbind(mortality, hardness) ~ location

Descriptive:
```

```
  location  n mortality  hardness
1    North 35  1633.600    30.400
2    South 26  1376.808    69.769

Wald-Type Statistic (WTS):
     Test statistic    df    p-value
location "51.584"       "2" "<0.001"

modified ANOVA-Type Statistic (MATS):
             Test statistic
location         69.882

p-values resampling:
        paramBS (WTS) paramBS (MATS)
location "<0.001"      "<0.001"
```

We find significant differences in mortality and water hardness between northern and southern towns.

The confidence region is returned as an ellipsoid specified by its center as well as its axes, which extend `Scale` units into the direction of the respective eigenvector. For two-dimensional outcomes as in this example, the confidence ellipsoid can also be plotted using the **ellipse** package (Murdoch and Chow, 2018), see Figure 3.

```
Center:
[,1]
[1,] 256.792
[2,] -39.369

Scale:
[1] 10.852716  2.736354

Eigenvectors:
[,1] [,2]
[1,]   -1    0
[2,]    0   -1
```

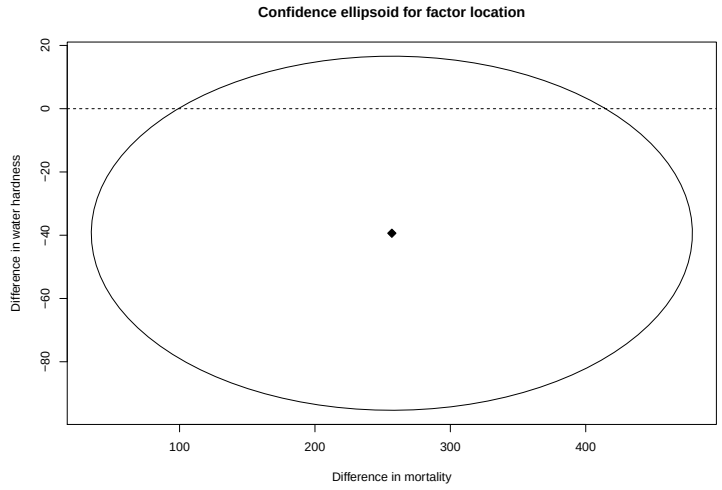


Figure 3: Plot of the confidence region for factor location in the water example from package **HSAUR**.

Nested design

To create a data example for a nested design, we use the `curdies` data set from the **GFD** package and extend it by introducing an artificial second outcome variable. In this data set, the levels of the nested factor (site) are named uniquely, i.e., levels 1-3 of factor site belong to "WINTER", whereas levels 4-6 belong to "SUMMER". Therefore, `nested.levels.unique` must be set to `TRUE`. The code for the analysis using both wide and long format is presented below.

```
R> library("GFD")
R> data("curdies")
R> set.seed(123)
R> curdies$dug2 <- curdies$dugesia + rnorm(36)

R> # first possibility: MANOVA.wide
R> fit1 <- MANOVA.wide(cbind(dugesia, dug2) ~ season + season:site,
+                data = curdies, iter = 1000,
+                nested.levels.unique = TRUE, seed = 123, CPU = 1)

R> # second possibility: MANOVA (long format)
R> dug <- c(curdies$dugesia, curdies$dug2)
R> season <- rep(curdies$season, 2)
R> site <- rep(curdies$site, 2)
R> curd <- data.frame(dug, season, site, subject = rep(1:36, 2))
R> fit2 <- MANOVA(dug ~ season + season:site, data = curd,
+                subject = "subject", nested.levels.unique = TRUE,
+                seed = 123, iter = 1000, CPU = 1)

R> # comparison of results
R> summary(fit1)
R> summary(fit2)

Call:
cbind(dugesia, dug2) ~ season + season:site



Descriptive:
season         site n  dugesia  dug2
1 SUMMER          4 6   0.419 -0.050
2 SUMMER          5 6   0.229  0.028
3 SUMMER          6 6   0.194  0.763
4 WINTER          1 6   2.049  2.497
5 WINTER          2 6   4.182  4.123
6 WINTER          3 6   0.678  0.724

Wald-Type Statistic (WTS):
            Test statistic df p-value
season               6.999  2   0.030
season:site         16.621  8   0.034

modified ANOVA-Type Statistic (MATS):
             Test statistic
season               12.296
season:site          15.064

p-values resampling:
              paramBS (WTS) paramBS (MATS)
season                0.064          0.032
season:site           0.275          0.216

Call:
dug ~ season + season:site

Descriptive:
season         site n  Mean 1 Mean 2
1 SUMMER          4 6  0.419 -0.050
2 SUMMER          5 6  0.229  0.028
3 SUMMER          6 6  0.194  0.763
4 WINTER          1 6  2.049  2.497
5 WINTER          2 6  4.182  4.123
6 WINTER          3 6  0.678  0.724
```

```
Wald-Type Statistic (WTS):
            Test statistic df p-value
season               6.999  2   0.030
season:site         16.621  8   0.034

modified ANOVA-Type Statistic (MATS):
             Test statistic
season              12.296
season:site         15.064

p-values resampling:
             paramBS (WTS) paramBS (MATS)
season               0.064          0.032
season:site          0.275          0.216
```

Post-hoc comparisons

In addition to global testing, the package **MANOVA.RM** allows for post-hoc comparisons. In particular, the following comparisons are implemented:

1. calculation of simultaneous multivariate p values for contrasts of the mean vector,
2. calculation of simultaneous confidence intervals based on summary effects (i. e. averaged across all dimensions) and
3. univariate comparisons for separate endpoints.

Calculation of simultaneous confidence intervals and p values for contrasts of the mean vector is based on the sum statistic, see Friedrich and Pauly (2018) for details. Confidence intervals are calculated based on summary effects, i.e., averaging over all dimensions, whereas the returned p-values are multivariate. Note that the confidence intervals and p values returned are simultaneous, i. e., they maintain the given alpha-level. Such contrasts include, e. g., Tukey's all-pairwise comparisons or Dunnett's many-to-one comparisons, see e. g. Hothorn et al. (2008a) for more examples. Confidence intervals for contrasts of the mean vector can be calculated using the function `simCI`, which is build on `contrMat` from the **multcomp** package (Hothorn et al., 2008b):

```
simCI(object, contrast, contmat, type, base)
```

Here, `object` is an object of class `"MANOVA"`. The user can choose between pairwise or user-defined contrasts. For user-defined constrast (`contrast = "user-defined"`), the contrast matrix of interest must be specified in `contmat`. Pairwise comparisons (`contrast = "pairwise"`) are calculated using the `contrMat` function of **multcomp** and accordingly, `type` and `base` specify the type of the pairwise comparison and the baseline group for Dunnett contrasts, see Hothorn et al. (2008b) for details on these parameters. To exemplify its application we reconsider the EEG example from above:

```
R> # pairwise comparison using Tukey contrasts
R> simCI(EEG_MANOVA, contrast = "pairwise", type = "Tukey")

#------ Call -----#

- Contrast:  Tukey
- Confidence level: 99 %

#------Multivariate post-hoc comparisons: p-values -----#

      contrast   p.value
1   M MCI - M AD   0.961
2   M SCC - M AD   0.548
3    W AD - M AD   0.899
4   W MCI - M AD   0.775
5   W SCC - M AD   0.417
6  M SCC - M MCI   0.368
7   W AD - M MCI   0.995
8  W MCI - M MCI   0.843
9  W SCC - M MCI   0.111
10  W AD - M SCC   0.845
11 W MCI - M SCC   0.938
```

```
12 W SCC - M SCC   0.989
13  W MCI - W AD   1.000
14  W SCC - W AD   0.526
15 W SCC - W MCI   0.563

#-----------Confidence intervals for summary effects-------------#

              Estimate    Lower     Upper
M MCI - M AD     4.275 -16.181707 24.731707
M SCC - M AD     8.767 -11.126510 28.660510
W AD - M AD      5.864 -14.947797 26.675797
W MCI - M AD     6.995 -12.978374 26.968374
W SCC - M AD     9.835  -9.799574 29.469574
M SCC - M MCI    4.492  -4.040216 13.024216
W AD - M MCI     1.589  -8.907565 12.085565
W MCI - M MCI    2.720  -5.996803 11.436803
W SCC - M MCI    5.560  -2.349709 13.469709
W AD - M SCC    -2.903 -12.254617  6.448617
W MCI - M SCC   -1.772  -9.069776  5.525776
W SCC - M SCC    1.068  -5.243764  7.379764
W MCI - W AD     1.131  -8.389330 10.651330
W SCC - W AD     3.971  -4.816350 12.758350
W SCC - W MCI    2.840  -3.719140  9.399140
```

The output is two-fold: First, the multivariate p values for the desired contrasts are returned. The second part of the output provides simultaneous confidence intervals for summary effects by averaging over all dimensions.

As another example using a user-defined contrast matrix, we consider the following one-way layout of the EEG data:

```
R> oneway <- MANOVA.wide(cbind(brainrate_temporal, brainrate_central)
+                         ~ diagnosis, data = EEGwide, iter = 1000,
+                        CPU = 1)
R> # a user-defined contrast matrix
R> H <- as.matrix(cbind(rep(1, 5), -1*Matrix::Diagonal(5)))
R> simCI(oneway, contrast = "user-defined", contmat = H)

#------ Call -----#

- Contrast:  user-defined
- Confidence level: 95 %

#------Multivariate post-hoc comparisons: p-values -----#

[1] 1.000 0.673 0.655 0.008 0.008

#-----------Confidence intervals for summary effects-------------#

  Estimate     Lower      Upper
1    0.013 -1.003489  1.0294895
2   -0.243 -1.050099  0.5640992
3   -0.251 -1.058376  0.5563761
4   -1.033 -1.812577 -0.2534227
5   -1.033 -1.791749 -0.2742508
```

Note that interpretation of the results depends on the user-defined contrast matrix.

If the global null hypothesis, e. g.

$$H_0(\boldsymbol{A}) : \{(\boldsymbol{P}_a \otimes \boldsymbol{I}_d)\boldsymbol{\mu} = \boldsymbol{0}\} \quad = \quad \{\boldsymbol{\mu}_1 = \cdots = \boldsymbol{\mu}_a\}$$

has been rejected, it is usually of interest to further investigate which univariate endpoints caused the rejection. A straight-forward way to do this is to calculate the univariate p values and adjust them for multiple testing, e. g., using Bonferroni correction. Consider the one-way layout above: Since the global null hypothesis can be rejected, we now wish to analyze which of the two univariate endpoints caused this rejection.

```
R> EEG1 <- MANOVA.wide(brainrate_temporal ~ diagnosis, data = EEGwide,
+                      iter = 1000, seed = 987, CPU = 1)
R> EEG2 <- MANOVA.wide(brainrate_central ~ diagnosis, data = EEGwide,
+                       iter = 1000, seed = 987, CPU = 1)
R> p.adjust(c(EEG1$resampling[, 2], EEG2$resampling[, 2]),
+           method = "bonferroni")
[1] 0 0
```

Thus, in this example both endpoints showed significant effects.
Note that it is often possible to conduct post-hoc comparisons according to the closure principle, thus avoiding the need to correct for multiple comparisons. Implementation of these methods for both MANOVA and RM designs is part of future research.

Graphical user interface

The GUI is started in R with the command `GUI.RM()`, `GUI.MANOVA()` and `GUI.MANOVAwide()` for repeated measures designs and multivariate data in long or wide format, respectively. Note that the GUI depends on **RGtk2** and will only work if **RGtk2** is installed. The user can specify the data location (either directly or via the "load data" button) and the formula as well as the number of iterations, the significance level α, the number of within subject factors (for repeated measures designs) and the name of the subject variable, see Figure 4. Furthermore, the user has the choice between the three resampling approaches "Perm" (only for RM designs), "paramBS" and "WildBS" denoting the permutation procedure, the parametric bootstrap and the wild bootstrap, respectively. Additionally, one can specify whether or not headers are included in the data file, and which separator and character symbols are used for decimals in the data file. The GUI for repeated measures also provides a plotting option, which generates a new window for specifying the factors to be plotted (in higher-way layouts) along with a few plotting parameters, see Figure 5.

```
R> library("MANOVA.RM")
R> GUI.RM()
```

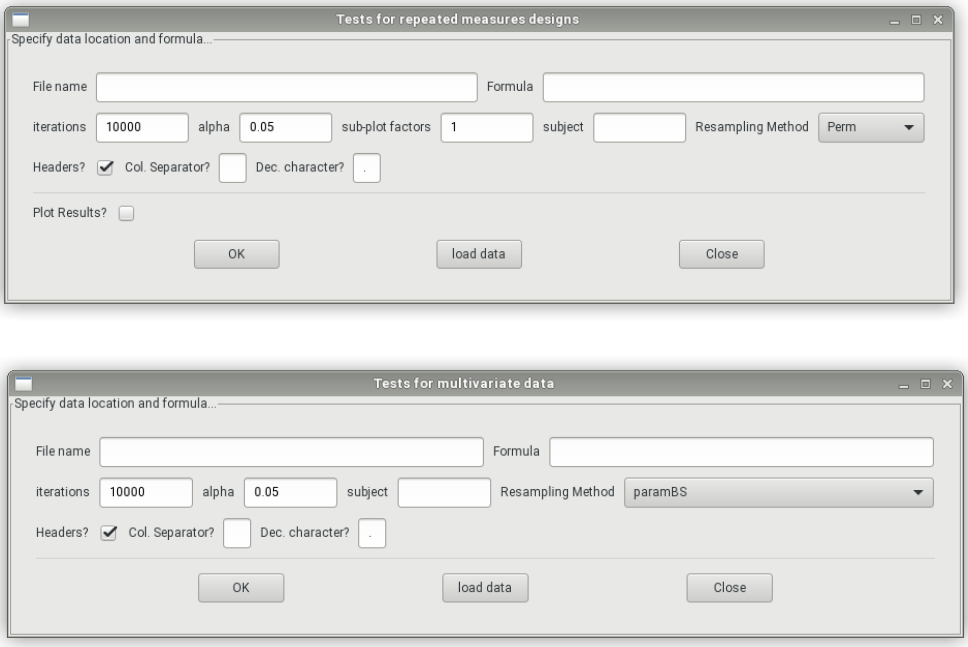


Figure 4: The GUI for tests in repeated measures designs (upper panel) and multivariate data (lower panel): The user can specify the data location and the formula as well as the resampling approach.

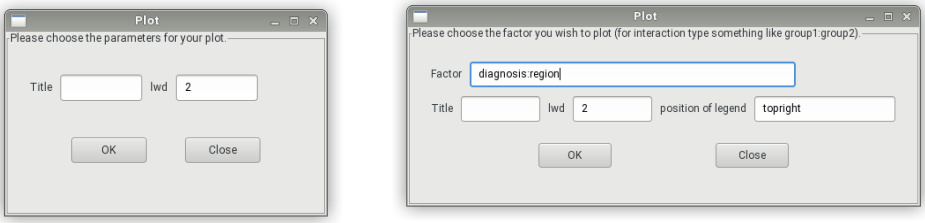


Figure 5: Graphical user interfaces for plotting: The left GUI is for the one-way layout (no choice of factors possible), the right one is for a two-way layout with an example for plotting interactions.

Discussion and Outlook

We have explicitly described the usage of the R package **MANOVA.RM** for analyzing various nonparametric multivariate MANOVA and RM designs making use of novel bootstrap- and permutation-approaches. Moreover, the corresponding models and inference procedures that have been derived and theoretically analyzed in previous papers are explained as well. In particular, three different test statistics of Wald-, ANOVA- and modified ANOVA-type are implemented together with appropriate critical values derived from asymptotic considerations, approximations or novel resampling approaches. Here, the latter is recommended in case of small to moderate sample sizes. All methods can be applied *without* assuming usual presumptions such as multivariate normality or specific covariance structures. Moreover, all procedures are particularly constructed to tackle covariance matrix heterogeneity across groups or even covariance singularity (in case of the MATS). In this way **MANOVA.RM** provides a flexible tool box for inferring hypotheses about (i) main and interaction effects in general factorial MANOVA and (ii) between and within subject effects in RM designs with possibly complex factorial structures on both, between and within subject factors.

In addition, we have placed a graphical user interface (GUI) at the users disposal to allow for a simple and intuitive use. It is planned to update the package on a regular basis; respecting the development of new procedures for general RM and MANOVA designs. For example, our working group is currently investigating the implementation of covariates in the above model in theoretical research and the resulting procedure may be incorporated in the future. Other topics include the possible implementation/improvement of subsequent multiple comparisons by the closure principle, as in Burchett et al. (2017), for both MANOVA and RM designs.

Acknowledgments

The work of Sarah Friedrich and Markus Pauly was supported by the German Research Foundation project DFG-PA 2409/3-1.

Bibliography

M. J. Anderson. A new method for non-parametric multivariate analysis of variance. *Austral Ecology*, 26(1):32–46, 2001. [p380]

A. C. Bathke, S. Friedrich, F. Konietschke, M. Pauly, W. Staffen, N. Strobl, and Y. Höller. Testing Mean Differences Among Groups: Multivariate and Repeated Measures Analysis with Minimal Assumptions. *Multivariate Behavioral Research*, 2018. URL https://doi.org/10.1080/00273171.2018.1446320. [p381, 382, 386]

E. Brunner. Asymptotic and approximate analysis of repeated measures designs under heteroscedasticity. *Mathematical Statistics with Applications in Biometry*, 2001. [p381, 382, 383, 386]

E. Brunner, H. Dette, and A. Munk. Box-type approximations in nonparametric factorial designs. *Journal of the American Statistical Association*, 92(440):1494–1502, 1997. [p383]

E. Brunner, S. Domhof, and F. Langer. *Nonparametric Analysis of Longitudinal Data in Factorial Experiments*. John Wiley & Sons, New York, USA, 2002. [p383]

W. W. Burchett, A. R. Ellis, S. W. Harrar, and A. C. Bathke. Nonparametric inference for multivariate data: The R package npmv. *Journal of Statistical Software*, 76(4):1–18, 2017. URL https://doi.org/10.18637/jss.v076.i04. [p380, 398]

W. F. Christensen and A. C. Rencher. A Comparison of Type I Error Rates and Power Levels for Seven Solutions to the Multivariate Behrens-Fisher Problem. *Communications in Statistics - Simulation and Computation*, 26(4):1251–1273, 1997. [p383]

E. Chung and J. P. Romano. Exact and asymptotically robust permutation tests. *The Annals of Statistics*, 41(2):484–507, 2013. [p381]

G. Cornu, F. Mortier, C. Trottier, and X. Bry. *SCGLR: Supervised Component Generalized Linear Regression*, 2018. URL https://CRAN.R-project.org/package=SCGLR. R package version 3.0. [p381]

C. S. Davis. *Statistical Methods for the Analysis of Repeated Measurements*. Springer-Verlag, 2002. [p380]

B. S. Everitt and T. Hothorn. *HSAUR: A Handbook of Statistical Analyses Using R (1st Edition)*, 2017. URL https://CRAN.R-project.org/package=HSAUR. R package version 1.3-9. [p392]

L. Finos, with contributions by Florian Klinglmueller, D. Basso, A. Solari, L. Benetazzo, J. Goeman, and M. Rinaldo. *Flip: Multivariate Permutation Tests*, 2018. URL https://CRAN.R-project.org/package=flip. R package version 2.5.0. [p381]

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, 2nd edition, 2011. URL http://socserv.socsci.mcmaster.ca/jfox/Books/Companion. [p381]

S. Friedrich and M. Pauly. MATS: Inference for potentially singular and heteroscedastic MANOVA. *Journal of Multivariate Analysis*, 165:166–179, 2018. [p381, 383, 388, 392, 395]

S. Friedrich, E. Brunner, and M. Pauly. Permuting longitudinal data in spite of the dependencies. *Journal of Multivariate Analysis*, 153:255–265, 2017a. [p381, 382, 385, 386]

S. Friedrich, F. Konietschke, and M. Pauly. Gfd: An r package for the analysis of general factorial designs. *Journal of Statistical Software, Code Snippets*, 79(1):1–18, 2017b. URL https://doi.org/10.18637/jss.v079.c01. [p381]

S. Friedrich, F. Konietschke, and M. Pauly. *MANOVA.RM: Resampling-Based Analysis of Multivariate Data and Repeated Measures Designs*, 2019. URL http://github.com/smn74/MANOVA.RM. R package version 0.4.1. [p381]

A. K. Gupta, S. W. Harrar, and Y. Fujikoshi. MANOVA for large hypothesis degrees of freedom under non-normality. *Test*, 17(1):120–137, 2008. [p381]

S. W. Harrar and A. C. Bathke. A modified two-factor multivariate analysis of variance: Asymptotics and small sample approximations. *Annals of the Institute of Statistical Mathematics*, 64(1):135–165, 2012. [p381]

T. Hothorn, F. Bretz, and P. Westfall. Simultaneous inference in general parametric models. *Biometrical Journal*, 50(3):346–363, 2008a. [p395]

T. Hothorn, F. Bretz, and P. Westfall. Simultaneous inference in general parametric models. *Biometrical Journal*, 50(3):346–363, 2008b. [p395]

Y. Huang, H. Xu, V. Calian, and J. C. Hsu. To permute or not to permute. *Bioinformatics*, 22(18):2244–2248, 2006. [p381]

R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 2007. [p380]

F. Konietschke, A. C. Bathke, S. W. Harrar, and M. Pauly. Parametric and nonparametric bootstrap methods for general MANOVA. *Journal of Multivariate Analysis*, 140:291–301, 2015. [p381, 382, 383, 386]

K. Krishnamoorthy and J. Yu. Modified Nel and Van Der Merwe Test for the Multivariate Behrens–Fisher Problem. *Statistics & Probability Letters*, 66(2):161–169, 2004. [p383]

M. Lawrence and D. Temple Lang. RGtk2: A graphical user interface toolkit for R. *Journal of Statistical Software*, 37(8):1–52, 2010. URL http://www.jstatsoft.org/v37/i08/. [p384]

J. Lemon. Plotrix: a package in the red light district of r. *R-News*, 6(4):8–12, 2006. [p385]

P. Livacic-Rojas, G. Vallejo, and P. Fernandez. Analysis of type i error rates of univariate and multivariate procedures in repeated measures designs. *Communications in Statistics - Simulation and Computation*, 39(3):624–640, 2010. [p381]

P. Livacic-Rojas, G. Vallejo, P. Fernández, and E. Tuero-Herrero. Power of modified brown-forsythe and mixed-model approaches in split-plot designs. *Methodology*, 13:9–22, 2017. [p381]

L. M. Lix and H. Keselman. Multivariate tests of means in independent groups designs: Effects of covariance heterogeneity and nonnormality. *Evaluation & the Health Professions*, 27(1):45–69, 2004. [p381]

L. M. Lix and A. M. Lloyd. A comparison of procedures for the analysis of multivariate repeated measurements. *Journal of Modern Applied Statistical Methods*, 6(2):380–398, 2007. [p381]

M. McFarquhar, S. McKie, R. Emsley, J. Suckling, R. Elliott, and S. Williams. Multivariate and repeated measures (mrm): A new toolbox for dependent and multimodal group-level neuroimaging data. *NeuroImage*, 132:373–389, 2016. [p381]

D. Murdoch and E. D. Chow. *Ellipse: Functions for Drawing Ellipses and Ellipse-Like Confidence Regions*, 2018. URL https://CRAN.R-project.org/package=ellipse. R package version 0.4.1. [p393]

D. Nel and C. Van der Merwe. A Solution to the Multivariate Behrens-Fisher Problem. *Communications in Statistics-Theory and Methods*, 15(12):3719–3735, 1986. [p383]

K. Noguchi, Y. R. Gel, E. Brunner, and F. Konietschke. nparld: An r software package for the nonparametric analysis of longitudinal data in factorial experiments. *Journal of Statistical Software*, 50(12):1–23, 2012. URL http://www.jstatsoft.org/v50/i12/. [p381]

G. Vallejo, A. Fidalgo, and P. Fernandez. Effects of covariance heterogeneity on three procedures for analyzing multivariate repeated measures designs. *Multivariate Behavioral Research*, 36(1):01–27, 2001. [p381]

G. Vallejo Seco, J. A. Gras, and M. Ato García. Comparative robustness of recent methods for analyzing multivariate repeated measures designs. *Educational and Psychological Measurement*, 67(3):410–432, 2007. [p381]

S. Xiao and J.-T. Zhang. Modified Tests for Heteroscedastic Two-Way MANOVA. *Journal of Advanced Statistics*, 1(1):1–16, 2016. [p381]

H. Yanagihara and K.-H. Yuan. Three Approximate Solutions to the Multivariate Behrens–Fisher Problem. *Communications in Statistics - Simulation and Computation*, 34(4):975–988, 2005. [p383]

Y. Yao. An Approximate Degrees of Freedom Solution to the Multivariate Behrens–Fisher Problem. *Biometrika*, 52(1/2):139–147, 1965. [p383]

J.-T. Zhang. Two-way MANOVA with unequal cell sizes and unequal cell covariance matrices. *Technometrics*, 53(4):426–439, 2011. [p381]

Øyvind Langsrud and B.-H. Mevik. *Ffmanova: Fifty-Fifty MANOVA*, 2019. URL https://CRAN.R-project.org/package=ffmanova. R package version 1.1.0. [p381]

Sarah Friedrich
Department of Medical Statistics, University Medical Center Göttingen
Humboldtallee 32, 37073 Göttingen
Germany
sarah.friedrich@med.uni-goettingen.de

Frank Konietschke
Charité — Universitätsmedizin Berlin
Corporate Member of Freie Universität Berlin
Humboldt-Universität zu Berlin, and
Berlin Institute of Health, Institute of Biometry and Clinical Epidemiology
Charitéplatz 1, 10117 Berlin
Germany
frank.konietschke@charite.de

Markus Pauly
Fakultät Statistik, Technische Universität Dortmund
44221 Dortmund
Germany
markus.pauly@tu-dortmund.de

spGARCH: An R-Package for Spatial and Spatiotemporal ARCH and GARCH models

by Philipp Otto

Abstract In this paper, a general overview on spatial and spatiotemporal ARCH models is provided. In particular, we distinguish between three different spatial ARCH-type models. In addition to the original definition of Otto et al. (2016), we introduce an logarithmic spatial ARCH model in this paper. For this new model, maximum-likelihood estimators for the parameters are proposed. In addition, we consider a new complex-valued definition of the spatial ARCH process. Moreover, spatial GARCH models are briefly discussed. From a practical point of view, the use of the R-package **spGARCH** is demonstrated. To be precise, we show how the proposed spatial ARCH models can be simulated and summarize the variety of spatial models, which can be estimated by the estimation functions provided in the package. Eventually, we apply all procedures to a real-data example.

Introduction

Whereas autoregressive conditional heteroscedasticity (ARCH) models are applied widely in time series analysis, especially in financial econometrics, spatial conditional heteroscedasticity has not been seen as critical issue in spatial econometrics up to now. Although it is well-known that classical least squares estimators are biased for spatially correlated data as well as for spatial data with an inhomogeneous variance across space, there are just a few papers proposing statistical models accounting for spatial conditional heteroscedasticity in terms of the ARCH and GARCH models of Engle (1982) and Bollerslev (1986). The first extensions to spatial models attempted were time series models incorporating spatial effects in temporal lags (see Borovkova and Lopuhaa 2012 and Caporin and Paruolo 2006, for instance). Instantaneous spatial autoregressive dependence in the conditional second moments, i.e., the conditional variance in each spatial location is influenced by the variance nearby, has been introduced by Otto et al. (2016). Further details and derivations can also be found in Otto et al. (2018, 2019). Their models allow for these instantaneous effects but require certain regularity conditions. In this paper, we propose an alternative specification of spatial autoregressive conditional heteroscedasticity based on an exponential definition of the conditional variance. This new model can be seen as the spatial equivalent of the log-GARCH model by Pantula (1986); Geweke (1986); Milhøj (1987). Other recent papers propose a mixture of these two approaches (see Sato and Matsuda 2017, 2018b). Moreover, all these models can be used in spatiotemporal settings (see Otto et al. 2018; Sato and Matsuda 2018a).

In addition to the novel spatial logarithmic ARCH model, this paper demonstrates the use of the R-package **spGARCH**. From this practical point of view, the simulation of several spatial ARCH-type models as well as the estimation of a variety of spatial models with conditional heteroscedasticity are shown. There are several packages implementing geostatistical models, kriging approaches, and other spatial models (cf. Cressie 1993; Cressie and Wikle 2011). One of the most powerful packages used to deal with models of spatial dependence is **spdep**, written by Bivand and Piras (2015). It implements most spatial models in a user-friendly way, such as spatial autoregressive models, spatial lag models, and so forth (see, also, Elhorst 2010 for an overview). These models are typically called spatial econometrics models, although they are not tied to applications in economics. In contrast, the package **gstat** provides functions for geostatistical models, variogram estimation, and various kriging approaches (see Pebesma 2004 for details). For dealing with big geospatial data, the **Stem** package uses an expectation-maximization (EM) algorithm for fitting hierarchical spatiotemporal models (see Cameletti 2015 for details). For a distributed computing environment, the MATLAB software D-STEM from Finazzi and Fasso (2014) also provides powerful tools for dealing with heterogeneous spatial supports, large multivariate data sets, and heterogeneous spatial sampling networks. Additionally, these fitted models are suitable for spatial imputation. Contrary to these EM approaches, Bayesian methods for modeling spatial data are implemented in the R-**INLA** package (see Rue et al. 2009 for technical details of the integrated nested Laplace approximations and Martins et al. 2013 for recently implemented features). Along with this package, the R-**INLA** project provides several functions for diverse spatial models incorporating integrated nested Laplace approximations.

In contrast to the above mentioned software for spatial models, the prevalent R-package for time series GARCH-type models is **rugarch** from Ghalanos (2018). Since **spGARCH** has been developed mainly to deal with spatial data, we aim to provide a package which is user-friendly for researchers

and data scientists working in applied spatial science. Thus, the package is coordinated with the objects and ideas of R packages for spatial data rather than packages for dealing with time series.

We structure the paper as follows. In the next Section Spatial ARCH-type models, we discuss all covered spatial and spatiotemporal ARCH-type models. In addition, we introduce a novel logarithmic spatial ARCH model, which has weaker regularity conditions than the other spatial ARCH models. In the subsequent section, parameter estimation based on the maximum-likelihood principle is discussed for both the previously proposed spatial ARCH models as well as the new logarithmic spatial ARCH model. Furthermore, spatial GARCH models are briefly discussed. However, the focus of this paper should be on ARCH-type models. After these theoretical sections, we demonstrate the use of the R-package **spGARCH** in Section Overview of the R-Package spGARCH. Further, we fit a spatial autoregressive model with exogenous regressors and spatial ARCH residuals for a real-world data set. In particular, we analyze prostate cancer incidence rates in southeastern U.S. states. Section Summary and discussion concludes the paper.

Spatial ARCH-type models

Let $\{Y(\boldsymbol{s}) \in \mathbb{R} : \boldsymbol{s} \in D\}$ be a univariate stochastic process having a spatial autoregressive structure in the conditional variance. The process is defined in a multidimensional space D, which is typically a subset of the q-dimensional real numbers $\mathbb{R}^q$, as space is usually finite. For dealing with spatial lattice data, D is subset of the q-dimensional integers $\mathbb{Z}^q$. For both cases, it is important that the subset contains a q-dimensional rectangle of positive volume (cf. Cressie and Wikle 2011). Moreover, this definition is suitable for modeling spatiotemporal data, as one might assume that D is the product set $\mathbb{R}^k \times \mathbb{Z}^l$ with $k + l = d$.

To define spatial models, in particular areal spatial models such as the simultaneous autoregressive (SAR) models, it is convenient to consider a vector of observations $\boldsymbol{Y} = (Y(\boldsymbol{s}_1), \ldots, Y(\boldsymbol{s}_n))'$ at all locations $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$. For spatial ARCH models, we specify this vector as

$$\boldsymbol{Y} = \operatorname{diag}(\boldsymbol{h})^{1/2}\boldsymbol{\varepsilon}\,, \tag{1}$$

an analogue to the well-known time series ARCH models (cf. Engle 1982; Bollerslev 1986). However, note that the vector $\boldsymbol{h}$ does not necessarily coincide with the conditional variance

$$Var(Y(\boldsymbol{s}_i)|Y(\boldsymbol{s}_1), \ldots, Y(\boldsymbol{s}_{i-1}))\,,$$

as the variance in any location $\boldsymbol{s}_j$ also depends on $Y(\boldsymbol{s}_i)$ for $j \neq i$ (see Otto et al. 2018 for details). We now distinguish between several spatial ARCH-type models via the definition of $\boldsymbol{h}$.

Spatial ARCH model

First, we define this vector $\boldsymbol{h}$ in such a way as to be analogous to the definition in Otto et al. (2018). For this model, the vector $\boldsymbol{h}_O$ is given by

$$\boldsymbol{h}_O = \alpha\mathbf{1} + \rho\mathbf{W}\operatorname{diag}(\boldsymbol{Y})\boldsymbol{Y}\,, \tag{2}$$

where $\operatorname{diag}(\boldsymbol{a})$ is a diagonal matrix with the entries of $\boldsymbol{a}$ on the diagonal. In order to be consistent with the implementation in the R-package **spGARCH**, we focus on the special case with two parameters α and ρ, whereas Otto et al. (2018) proposed a more general model with a vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)'$ and the first-order spatial lag $\mathbf{W}\operatorname{diag}(\boldsymbol{Y})\boldsymbol{Y}$.

For this definition, there is a one-to-one relation between $\boldsymbol{Y}$ and $\boldsymbol{\varepsilon}$ via the squared observations $\boldsymbol{Y}^{(2)} = (Y(\boldsymbol{s}_1)^2, \ldots, Y(\boldsymbol{s}_n)^2)'$ and squared errors $\boldsymbol{\varepsilon}^{(2)} = (\varepsilon(\boldsymbol{s}_1)^2, \ldots, \varepsilon(\boldsymbol{s}_n)^2)'$ with

$$\boldsymbol{Y}^{(2)} = \alpha\ (\mathbf{I} - \mathbf{A})^{-1}\,\boldsymbol{\varepsilon}^{(2)}\,, \tag{3}$$

where $\mathbf{W}$ is a predefined spatial weighting matrix and

$$\mathbf{A} = \rho\,\operatorname{diag}\left(\varepsilon(\boldsymbol{s}_1)^2, \ldots, \varepsilon(\boldsymbol{s}_n)^2\right)\mathbf{W}\,.$$

Thus,

$$\boldsymbol{h}_O = \alpha\mathbf{1} + \rho\alpha\mathbf{W}\,(\mathbf{I} - \mathbf{A})^{-1}\,\boldsymbol{\varepsilon}^{(2)}\,.$$

It is important to assume that the spatial weighting matrix is a non-stochastic, positive matrix with zeros on the main diagonal to ensure that a location is not influenced by itself (cf. Elhorst 2010; Cressie

and Wikle 2011). The vector of random errors is denoted by $\boldsymbol{\varepsilon}$. Due to the complex dependence implied by the weighting matrix $\mathbf{W}$, $\boldsymbol{h}_O$ is not necessarily positive; thus, $\text{diag}(\boldsymbol{h})^{1/2}$ does not necessarily have a solution in the real numbers such that the process in (1) is well-defined. This is only the case if the condition of the following lemma is fulfilled.

Lemma 1 (Otto et al. 2018)**.** *Suppose that $\alpha \geq 0$, $\rho \geq 0$ and that $\det(\mathbf{I} - \mathbf{A}^2) \neq 0$. If all elements of the matrix*

$$(\mathbf{I} - \mathbf{A}^2)^{-1} \tag{4}$$

are nonnegative, then all components of $\boldsymbol{Y}^{(2)}$ are nonnegative, i.e., $Y(\boldsymbol{s}_i)^2 \geq 0$ for $i = 1, \ldots, n$. Moreover, $h_O(\boldsymbol{s}_i) \geq 0$ for $i = 1, \ldots, n$.

It is important to note that $\mathbf{A}$ depends on both the weighting matrix and the realizations of the errors. In order to ensure that this condition is fulfilled, Otto et al. (2018) propose to truncate the support of the error distribution on the interval $(-a, a)$ with

$$a = \begin{cases} \infty & \exists k > 0 : \rho \mathbf{W}^k = \mathbf{0} \\ 1/\sqrt[4]{\rho^2 ||\mathbf{W}^2||_1} & \rho^2 ||\mathbf{W}^2||_1 > 0 \end{cases},$$

where $||\cdot||_1$ denotes the matrix norm based on the Manhattan norm.

There are two cases in which the support of the errors does not need to be constrained. If $\rho = 0$, the process coincides with a spatial white noise process such that a equals ∞. Moreover, all entries of $\boldsymbol{h}$ are non-negative if $\mathbf{W}$ is similar to a strictly triangular matrix. Then, $\mathbf{W}$ is nilpotent. This case covers the classical time-series ARCH(p) models introduced by Engle (1982) as well as the so-called oriented spARCH processes. For these processes, the spatial dependence has a certain direction, e.g., observations are only influenced by observations in a southward direction or by observations which are closer to an arbitrarily chosen center. The setting also covers recent time-series GARCH models incorporating spatial information (e.g., Borovkova and Lopuhaa 2012; Caporin and Paruolo 2006).

Of course, the truncated support of the errors has an impact on the extent of the spatial dependence on the conditional variances. Obviously, the support need not be constrained regarding $\rho = 0$. However, this support decreases with increasing values of ρ. For instance, if $\rho = 1$, then the parameter a is equal to 0.968 for Rook's contiguity matrices on a two-dimensional lattice. As a measure of the spatial dependence of the variance, one might consider Moran's I for the squared observations (see Moran 1950). Moreover, we observe that the growth rate of I decreases with increasing spatial weights. This trend can be explained by the compact support of the errors. Since there cannot be large variations $\varepsilon(\boldsymbol{s}_i)$ in absolute terms, there also cannot be large spatial clusters of high or low variance. To illustrate this behavior, Figure 1 depicts Moran's I for simulated observations $\boldsymbol{Y}$ and their squares for $\rho \in \{0, 0.05, \ldots, 2\}$. For the Monte Carlo simulation study, we simulate $n = 400$ observation on a two-dimensional lattice $D = \{\boldsymbol{s} = (s_1, s_2)' \in \mathbb{Z}^2 : 0 \leq s_1, s_2 \leq 20\}$. The weighting matrix is a common Rook's contiguity matrix, and the simulation is done for 10^5 replications. Although the exact distribution of Moran's statistic is bounded, the standardized statistic is asymptotically normally distributed for the "majority of spatial structures" (Tiefelsdorf and Boots 1995, see also Cliff and Ord 1981). Thus, the asymptotic 95% confidence intervals are plotted in Figure 1, as well.

Spatial Log-ARCH model

Next, we consider an logarithmic spatial ARCH process (log-spARCH). In this setting, we define the natural logarithm of $\boldsymbol{h}_E = (h_E(\boldsymbol{s}_1), \ldots, h_E(\boldsymbol{s}_n))'$ as

$$\ln \boldsymbol{h}_E = \alpha \mathbf{1} + \rho \mathbf{W} g_b(\boldsymbol{\varepsilon}), \tag{5}$$

with a function $g_b : \mathbb{R}^n \to \mathbb{R}^n$. Like Nelson (1991), we assume that

$$g_b(\boldsymbol{\varepsilon}) = (\ln |\varepsilon(\boldsymbol{s}_1)|^b, \ldots, \ln |\varepsilon(\boldsymbol{s}_n)|^b)'$$

for positive values of b. For this definition, there is a one-to-one relation between $\boldsymbol{Y}$ and $\boldsymbol{\varepsilon}$, as we show in the following theorem.

Theorem 2. *Suppose that $\alpha > 0$, $\rho \geq 0$, and $w_{ij} \geq 0$ for all $i, j = 1, \ldots, n$ and $g_b(\boldsymbol{\varepsilon}) = (\ln |\varepsilon(\boldsymbol{s}_1)|^b, \ldots, \ln |\varepsilon(\boldsymbol{s}_n)|^b)'$. Then there exists one and only one $Y(\boldsymbol{s}_1), \ldots, Y(\boldsymbol{s}_n)$ that corresponds to each $\varepsilon(\boldsymbol{s}_1), \ldots, \varepsilon(\boldsymbol{s}_n)$ for $b > 0$.*

At location $\boldsymbol{s}_i$, the value of $h_E(\boldsymbol{s}_i)$ is then given by

$$\ln h_E(\boldsymbol{s}_i) = \alpha + \sum_{v=1}^{n} \rho b w_{iv} \ln |\varepsilon(\boldsymbol{s}_v)| \text{ for } i = 1, \ldots, n\,.$$

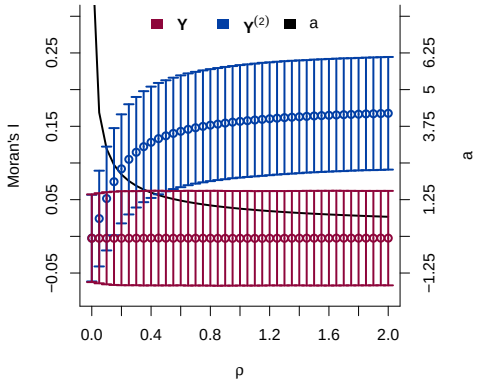


Figure 1: Moran's I of the observations $\boldsymbol{Y}$ and the squared observations $\boldsymbol{Y}^{(2)}$, including the asymptotic 95% confidence intervals of I for $\rho \in \{0, 0.05, \ldots, 2\}$. The resulting bound a is plotted as a bold, black line.

For this definition of g_b, one could rewrite $\ln \boldsymbol{h}$ as

$$\ln \boldsymbol{h}_E = \mathbf{S}\left(\alpha \mathbf{1} + \rho b \mathbf{W} \ln |\boldsymbol{Y}|\right) \tag{6}$$

with

$$\mathbf{S} = (s_{ij})_{i,j=1,\ldots,n} = \left(\mathbf{I} + \frac{1}{2}\rho b \mathbf{W}\right)^{-1}.$$

In contrast to the spARCH process described in Section Spatial ARCH model, Corollary 1 shows that the entries of $\boldsymbol{h}_E$ are positive for all $\rho \geq 0$ and $\alpha > 0$. Hence, the process is well-defined and there are no further restrictions needed, as in the case for the spARCH model.

Corollary 1. *Assume that the assumptions of Theorem 2 are fulfilled, then $\boldsymbol{h}_E(\boldsymbol{s}_i) \geq 0$ for all $i = 1, \ldots, n$.*

For all proofs, we refer to the Appendix.

Complex Spatial ARCH model

Now, we propose a complex-valued spARCH process. In order to obtain a solution of $\operatorname{diag}(\boldsymbol{h})^{1/2}$ in the n-dimensional space of real numbers for the model defined in (2), all elements of the matrix $(\mathbf{I} - \mathbf{A}^2)^{-1}$ must be nonnegative (see Otto et al. 2018). For the complex spARCH process, we relax the assumption that there should be a solution to $\operatorname{diag}(\boldsymbol{h})^{1/2}$ in the real numbers and also consider complex solutions. Thus, the definition of $\boldsymbol{h}$ coincides with $\boldsymbol{h}_O$ of the original model, i.e.,

$$h_C(\boldsymbol{s}_i) = \alpha + \sum_{v=1}^{n} \rho w_{iv} Y(\boldsymbol{s}_v)^2. \tag{7}$$

Spatiotemporal ARCH model

Finally, we show that spatiotemporal processes are covered directly by these approaches. For spatiotemporal data, the vector $\boldsymbol{s}$ simply includes both the spatial location $\boldsymbol{s}_s$ and the point in time t, i.e., $\boldsymbol{s} = (\boldsymbol{s}_s, t)'$. In addition, it is important to assume that future observations do not influence past observations, i.e., the weights w_{ij} must be zero if $t_j \geq t_i$. However, the dimension of the weighting matrix $\mathbf{W}$ might become very large for this representation. More precisely, the matrix has dimension $NT \times NT$, where N is the total number of spatial locations and T stands for the total number of time points. From a computational perspective, this is not necessarily a drawback since $\mathbf{W}$ is usually sparse and could also have a block diagonal structure. Moreover, it is often reasonable to assume that $h(\boldsymbol{s}_i)$ is only influenced by the neighbors of $\boldsymbol{s}_{s,i}$ at the same point of time and by past observations at the same

Process type	Definition of $\boldsymbol{h}$	Comments
spARCH	$\boldsymbol{h}_O = \alpha\mathbf{1} + \rho\mathbf{W}\left(\mathbf{I} - \mathbf{A}\right)^{-1}\left(\alpha\boldsymbol{\varepsilon}^{(2)}\right)$	$\boldsymbol{\varepsilon}$ is simulated from multivariate normal distribution (MN) truncated on the interval $\left[-1/\sqrt[4]{\lvert\lvert\rho^2\mathbf{W}^2\rvert\rvert_1}, 1/\sqrt[4]{\lvert\lvert\rho^2\mathbf{W}^2\rvert\rvert_1}\right]$
spARCH (oriented)	$\boldsymbol{h}_O = \alpha\mathbf{1} + \rho\mathbf{W}\left(\mathbf{I} - \mathbf{A}\right)^{-1}\left(\alpha\boldsymbol{\varepsilon}^{(2)}\right)$	$\boldsymbol{\varepsilon} \sim \mathrm{MN}(\mathbf{0}, \mathbf{I})$, $\mathbf{W}$ must be a strictly triangular weighting matrix
spatial log-ARCH	$\ln \boldsymbol{h}_E = \mathbf{S}\left(\alpha\mathbf{1} + \rho b\mathbf{W}\ln\lvert\boldsymbol{Y}\rvert\right)$	$\boldsymbol{\varepsilon} \sim \mathrm{MN}(\mathbf{0}, \mathbf{I})$, but moments of $\boldsymbol{Y}$ differ from the moments of classical spARCH process (cf. Otto et al. 2018)
spARCH (complex)	$\boldsymbol{h}_C = \alpha\mathbf{1} + \rho\mathbf{W}\left(\mathbf{I} - \mathbf{A}\right)^{-1}\left(\alpha\boldsymbol{\varepsilon}^{(2)}\right)$	$\boldsymbol{\varepsilon} \sim \mathrm{MN}(\mathbf{0}, \mathbf{I})$, but complex-valued $\boldsymbol{Y}$

Table 1: Overview of all types of spARCH models implemented in the **spGARCH** package.

location. Then the weighting matrix would have the following structure

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{I} & \mathbf{W}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{W}_T \end{pmatrix}.$$

Indeed, it is plausible to weight the spatial and temporal lags differently by replacing $\rho\mathbf{W}$ by a sum

$$\rho \begin{pmatrix} \mathbf{W}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{W}_T \end{pmatrix} + \phi_1 \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} + \cdots$$

with positive weights ϕ_k for all temporal lags $1 \leq k \leq p$.

Spatial ARCH Disturbances

Since all conditional and unconditional odd moments of spatial ARCH processes are equal to zero, these ARCH-type models can easily be added to any kind of (spatial) regression model without influencing the mean equation as well as the spatial dependence in the first conditional and unconditional moments. This makes the spatial ARCH models flexible tools for dealing with conditional spatial heteroscedasticity in the residuals of spatial models. For instance, one can consider spatial autoregressive models for $\boldsymbol{Y}$, i.e.,

$$\boldsymbol{Y} = \lambda\mathbf{B}\boldsymbol{Y} + \mathbf{X}\boldsymbol{\beta} + \boldsymbol{u} \tag{8}$$

with $\boldsymbol{u}$ following either a spatial ARCH model with the original definition $\boldsymbol{h}_O$ or the logarithmic model with $\boldsymbol{h}_E$. Thus,

$$\boldsymbol{u} = \mathrm{diag}(\boldsymbol{h})^{1/2}\boldsymbol{\varepsilon}. \tag{9}$$

Further, we call this model the SARspARCH model. For $\lambda = 0$, the model collapses to a simple linear regression model; if, additionally, $\boldsymbol{\beta} = \mathbf{0}$, the model coincides with the previously discussed ARCH models. Thus, these coefficients can be used for testing against nested models.

In contrast to other models for heteroscedastic errors, such as the SARAR or SARMA models, which assume spatial autoregressive or spatial moving average error terms (cf. Kelejian and Prucha 2010; Fingleton 2008; Haining 1978), the SARspARCH model does not affect the spatial autocorrelation of the process, just the spatial heteroscedasticity, because all conditional and unconditional odd moments are equal to zero. Thus, $\lambda\mathbf{B}$ can be interpreted directly as the spatial dependence of the process, while $\rho\mathbf{W}$ describes the spatial dependence in the second conditional moments. Moreover, these two parts can be interpreted separately, as we will demonstrate in the last section via an empirical example.

Generalized Spatial ARCH Models

Additionally, one may include spatially lagged observations of $\boldsymbol{h}$ to construct spatial GARCH-type models. For instance, a spatial GARCH model is given by

$$\begin{aligned} \boldsymbol{h}_G &= \alpha\mathbf{1} + \rho\mathbf{W}\text{diag}(\boldsymbol{Y})\boldsymbol{Y} + \lambda\breve{\mathbf{W}}\boldsymbol{h}_G \quad (10)\\ &= (\mathbf{I} - \lambda\breve{\mathbf{W}})^{-1}(\alpha\mathbf{1} + \rho\mathbf{W}\text{diag}(\boldsymbol{Y})\boldsymbol{Y}) \,, \quad (11)\end{aligned}$$

where $\breve{\mathbf{W}}$ is a second spatial weighting matrix and λ is the corresponding spatial GARCH parameter. Obviously, the spatial GARCH-type models requires that $(\mathbf{I} - \lambda\breve{\mathbf{W}})$ is non-singular. In a similar manner, $\boldsymbol{h}_{LG}$ can be specified as

$$\ln \boldsymbol{h}_{LG} = \alpha\mathbf{1} + \rho\mathbf{W}g_b(\boldsymbol{\varepsilon}) + \lambda\breve{\mathbf{W}}\boldsymbol{h}_{LG} \,, \quad (12)$$

to define a spatial log-GARCH model. For theoretical details of spatial GARCH-type models, we refer to Otto and Schmid (2019) introducing a unified spatial GARCH model covering various spatial ARCH and GARCH models. Moreover, Otto and Schmid (2019) introduce an exponential spatial GARCH model allowing for asymmetry via an alternative definition of g in (5). To be precise, g is given by

$$g(\boldsymbol{\varepsilon}) = (\Theta\varepsilon_1 + \zeta(|\varepsilon_1| - E(|\varepsilon_1|)), \ldots, \Theta\varepsilon_n + \zeta(|\varepsilon_n| - E(|\varepsilon_n|))'$$

for the exponential spatial GARCH model.

Parameter Estimation

The parameters of a spatial ARCH process can be estimated by the maximum-likelihood approach. To obtain the joint density for $\boldsymbol{Y} = k(\boldsymbol{\varepsilon})$, the Jacobian matrix of k^{-1} at the observed values $\boldsymbol{y}$ must be computed (e.g., Bickel and Doksum 2015). If $f_{\boldsymbol{\varepsilon}}$ is the distribution of the error process, then the joint density $f_{\boldsymbol{Y}}$ of $\boldsymbol{Y}$ is given by

$$\begin{aligned} f_{\boldsymbol{Y}}(\boldsymbol{y}) &= f_{(Y(\boldsymbol{s}_1),\ldots,Y(\boldsymbol{s}_n))}(y_1,\ldots,y_n)\\ &= f_{\boldsymbol{\varepsilon}}\left(\frac{y_1}{\sqrt{h_1}},\ldots,\frac{y_n}{\sqrt{h_n}}\right) |\det\left(\left(\frac{\partial y_j/\sqrt{h_j}}{\partial y_i}\right)_{i,j=1,\ldots,n}\right)| \,. \quad (13)\end{aligned}$$

If the residuals are additionally independent and identically distributed, the parameter estimates can be obtained from the maximization of the log-likelihood as follows

$$(\hat{\alpha},\hat{\rho}) = \underset{\alpha>0,\rho\geq 0}{\arg\max}\ \ln|\det\left(\left(\frac{\partial y_j/\sqrt{h_j}}{\partial y_i}\right)_{i,j=1,\ldots,n}\right)| + \sum_{i=1}^{n}\ln f_\varepsilon(y_i) \,.$$

The Jacobian matrix, of course, depends on the definition of $\boldsymbol{h}$. For the spARCH process, this Jacobian matrix can be specified as

$$\frac{\partial y_j/\sqrt{h_j}}{\partial y_i} = \begin{cases} 1/\sqrt{h_j} & \text{for} \quad i = j \\ -\frac{y_i y_j}{h_j^{3/2}}\rho w_{ji} & \text{for} \quad i \neq j \end{cases} .$$

In contrast, the Jacobian matrix for the log-spARCH process is slightly different, namely

$$\frac{\partial y_j/\sqrt{h_j}}{\partial y_i} = \begin{cases} 1/\sqrt{h_j} & \text{for} \quad i = j \\ -\frac{by_j}{2y_i h_j^{3/2}}\rho s_{ji} w_{ji} & \text{for} \quad i \neq j \end{cases}$$

with

$$h_j = \exp\left(\sum_{v=1}^{n} s_{jv}\left(\alpha + \rho w_{jv}\ln|y_v|\right)\right) \,.$$

From a computational perspective, the computation of the log determinant of this matrix is feasible for large data sets. To be precise, the log-determinant is equal to

$$\ln|\det\left(\text{diag}\left(\frac{h_1}{y_1^2},\ldots,\frac{h_n}{y_n^2}\right) - \rho\mathbf{W}'\right)| + \sum_{i=1}^{n}\ln\frac{y_i^2}{h_i^{3/2}}$$

for the spARCH process. Similarly, it is given by

$$\ln|\det\left(\operatorname{diag}\left(\frac{2h_1}{b},\ldots,\frac{2h_n}{b}\right)-\rho\mathbf{S}'\circ\mathbf{W}'\right)|+\sum_{i=1}^{n}\ln\frac{b}{2h_i^{3/2}}\,.$$

for the log-spARCH process, where $\circ$ stands for the Hadamard product.

In the **spGARCH** package, we implemented the iterative maximization algorithm with inequality constraints proposed by Ye (1988), which is implemented in the R-package **Rsolnp** (see Ghalanos and Theussl 2012). It is important to note that the log determinant of the Jacobian also depends on the parameters in such a way that it needs to be computed in each iteration (see, also, Theorem 13.7.3 of Harville (2008) for the computation of a determinant for the sum of a diagonal matrix and an arbitrary matrix), but $\mathbf{W}$, and therefore $\mathbf{S}\circ\mathbf{W}$, are usually sparse. Thus, the required time for the estimation of the parameters depends mainly on the dimension and sparsity of $\mathbf{W}$.

Certainly, the choice of the weighting matrices are an important design choice of the models, which has to be prespecified. However, the true structure of $\mathbf{W}$ is rarely known in practice. Moreover, all estimated parameters depend on the selection of this matrix. Hence, inference on these parameters and the coefficients itself must be interpreted based on the choice of $\mathbf{W}$. For empirical applications, one might gain insights on the structure of $\mathbf{W}$ by looking at spatial autocorrelation functions or variograms. It is worth noting that the observations are uncorrelated for spatial ARCH models, so one should also look at squared observations. Then, the weighting scheme is typically chosen from a set of candidate schemes by maximizing certain goodness-of-fit criteria, like information criteria or out-of-sample prediction errors. For instance, $\mathbf{W}$ could be chosen as contiguity matrix, i.e., two locations are connected having positive weights, if they share a common border or if their distance is less than a certain threshold. For instance, in studies in spatial econometrics or epidemiology, the spatial domain is often a set of municipalities or counties (e.g., Amin et al. 2014; Buettner 2003). In this case, contiguity matrices are straightforward and if these binary matrices are additionally row-standardized, $\mathbf{W}\operatorname{diag}(\boldsymbol{Y})\boldsymbol{Y}$ can be interpreted as average of the squared neighboring observations. Alternatively, $\mathbf{W}$ can be specified as k-nearest-neighbor matrix, i.e., only the k nearest locations get positive weights, or as inverse-distance matrix, i.e., the weight between two locations is based on the distance between these locations. Further choices of $\mathbf{W}$ are discussed by Otto et al. (2018). Finally, it is worthy to mention that the weights could also depend on exogenous variables or other factors. For instance, they could incorporate economic disparities, e.g., differences in the gross domestic products, poverty rates, household incomes etc., or other covariates, like the wind direction and speed when modeling spatial dependence of air pollutants (cf. Merk and Otto 2019). For spatiotemporal autoregressive processes, there are also some approaches to estimate the entire spatial dependence structure using machine learning methods (e.g., Lam and Souza 2016; Otto and Steinert 2018).

Overview of the R-Package spGARCH

The R-package **spGARCH** provides several basic functions for the analysis of spatial data showing spatial conditional heteroscedasticity. In particular, the process can be simulated for arbitrarily chosen weighting matrices according to the definitions in Section Spatial ARCH-type models. Moreover, we implement a function for the computation of the maximum-likelihood estimators. To generate a user-friendly output, the object generated by the estimation function can easily be summarized by the generic `summary()` function. We also provide all common generic methods, such as `plot()`, `print()`, `logLik()`, and so forth. To maximize the computational efficiency, the actual version of the package contains compiled C++ code (using the packages **Rcpp** and **RcppEigen**, cf. Eddelbuettel and François 2011; Bates and Eddelbuettel 2013). A brief overview of the package and its main functions is given in Table 2. Further, we focus on the two main aspects of the package, i.e., the simulation (described in detail in Section Simulation of ARCH-type stochastic processes) and estimation (Section Maximum-likelihood estimation) aspects of the spARCH, log-spARCH, and SARspARCH processes.

Simulation of ARCH-type stochastic processes

The simulations of all spatial ARCH-type models are implemented in one function, namely, the `sim.spARCH()` function. The different definitions of the model are specified via the argument `type`. The use of `sim.spARCH()` is very similar to how a basic random number generator is used, meaning that the first argument `n` is the number of generated values and all further arguments specify the parameters of the spARCH process. For instance, one might simulate an oriented spARCH process (meaning $\mathbf{W}$ is triangular) on a $d\times d$ spatial lattice with $\rho=0.7$ and $\alpha=1$ using the following lines.

```
R> require("spdep")
R> rho             <- 0.7
```

Function	Description
Main functions	
`sim.spARCH()`	Simulation of spARCH and log-spARCH processes
`sim.spGARCH()`	Simulation of spGARCH, E-spGARCH, and log-spGARCH processes
`qml.spARCH()`	Quasi-maximum-likelihood estimation for spARCH models
`qml.SARspARCH()`	Quasi-maximum-likelihood estimation for SAR models with spARCH residuals
Generic methods	
`summary()`	Summary of an object of 'spARCH' class generated by `qml.spARCH()` or `qml.SARspARCH()`
`print()`	Printing method for 'spARCH' class or `summary.spARCH` class
`fitted()`	Extracts the fitted values of an object of 'spARCH' class
`residuals()`	Extracts the residuals of an object of 'spARCH' class
`logLik()`	Extracts the log-likelihood of an object of 'spARCH' class
`extractAIC()`	Extracts the AIC of an object of 'spARCH' class
`plot()`	Provides several descriptive plots of the residuals of an object of 'spARCH' class

Table 2: Summary of the main functions of the **spGARCH** package.

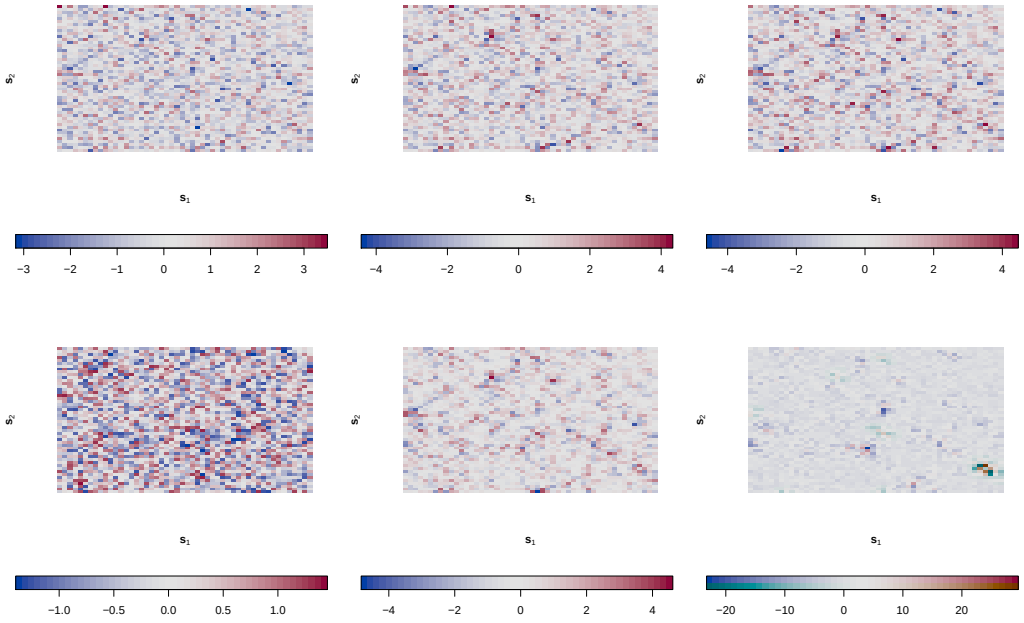
```
R> alpha            <- 1
R> d                <- 50
R> n                <- d^2
R> nblist           <- cell2nb(d, d, type = "queen")
R> W                <- nb2mat(nblist)
R> W[upper.tri(W)]  <- 0
R> Y                <- sim.spARCH(n = n, rho = rho, alpha = alpha, W = W,
+                                 type = "spARCH", control = list(seed = 5515))
```

To build the spatial weighting matrix, we used `cell2nb()` from the **spdep** package, returning an `nb` object of a $d \times d$ lattice (see Cressie 1993; Bivand and Piras 2015). Further, we converted the `nb` object into a contiguity matrix, as `sim.spARCH()` requires either a matrix (class `matrix`) or a sparse matrix (class `dgCMatrix`) as an argument. Usually, spatial weighting matrices are sparse by construction. Thus, $\mathbf{W}$ is always converted internally to a `dgCMatrix` matrix or rather to a `SparseMatrix` object defined in the eigen library in C++. Via the `control` parameter, a random seed might be passed to the simulation function. If not, a random seed is assigned randomly from a uniform distribution and printed in console in order that one might reproduce the result even without having a random seed specified in advance. We prefer to print a single number in the console rather than returning to the random number generator (RNG) state as an attribute of the returned vector. Thus, a random seed might either be passed as an optional argument to `sim.spARCH()` or set before calling `sim.spARCH()` by `set.seed()`.

There are several types of spatial ARCH processes which can be simulated by `sim.spARCH()`. They are all specified by the argument `type`. If

- `type = "spARCH"`, then the original spARCH process according to the definition in Otto et al. (2018) is simulated.
  - If there exists a permutation such that $\mathbf{W}$ is a strictly triangular matrix, then the function simulates automatically an oriented spARCH process with independent and identically gaussian distributed errors.
  - If there is no such permutation, then the errors are simulated from a truncated normal distribution with $a = 1/\sqrt[4]{\rho^2||\mathbf{W}||_1}$.
- `type = "log-spARCH"`, an log-spARCH process is simulated with an user-specified value of b (default 2) and standard normal random errors.
- `type = "complex-spARCH"`, complex solutions of $\text{diag}(\boldsymbol{h})^{1/2}$ are considered in order to simulate the spARCH process.

Figure 2 illustrates the behavior of different types of spatial ARCH processes. All of them are simulated with the same parameters and random seeds in such a manner that the vector $\boldsymbol{\varepsilon}$ is identical for all types of processes, except for the spARCH process with the truncated normal errors. In the first row, the spatial weighting is achieved via a strictly triangular Queen's contiguity matrix, which



Above left: spatial white noise for comparison; center: oriented spARCH (`type = ``gaussian''`); right: spatial E-ARCH (`type = ``exp''`).

Below left: spARCH with truncated normal errors (`type = ``gaussian''`); center: spatial E-ARCH (`type = ``exp''`), right: complex spARCH (`type = ``complex''`).

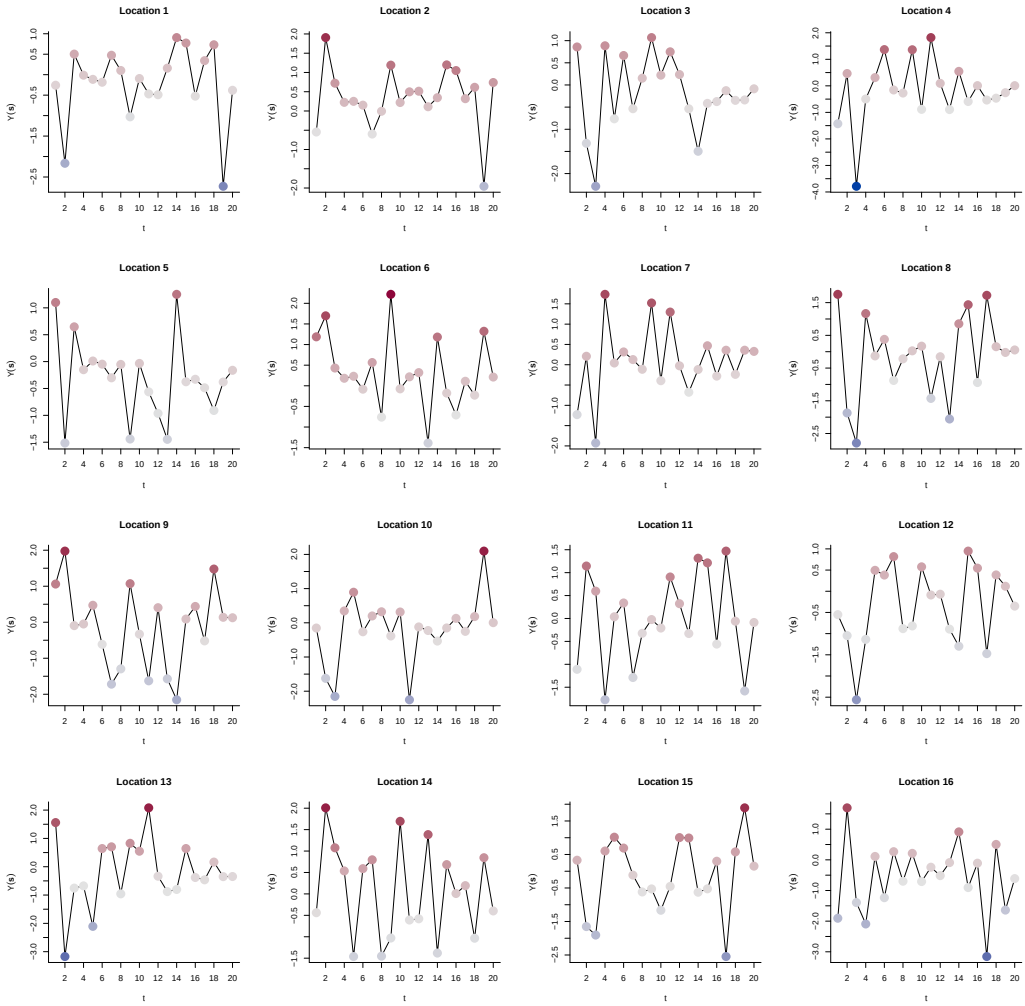
Figure 2: Simulations on a two-dimensional lattice for triangular matrices (above) and non-triangular matrices (below). For all simulations, we set $\rho = 0.7$ and $\alpha = 1$, and $\mathbf{W}$ is chosen to be the Queen's contiguity matrix.

means that the spatial dependence has its origin in the upper left corner. To the contrary, $\mathbf{W}$ presents a classical Queen's contiguity matrix in the second row. We additionally plot a spatial white noise process for comparison, as we used a rather unconventional two-color scheme. Using this kind of color scheme, one might distinguish between positive and negative observations, such that it is easier to see the spatial volatility clusters. Areas of smaller volatility are characterized by rather evenly gray pixels, whereas clusters of high volatility have rather intense colors. Moreover, the colors fluctuate irregularly between blue and red.

As pointed out in Section Spatial ARCH-type models, spatiotemporal ARCH models are directly covered if time is considered as one dimension of the q-dimensional space D. Thus, a two-dimensional spatiotemporal process $\{Y_t(\boldsymbol{s}) : t = 1, \ldots, T; \boldsymbol{s} \in D_{\boldsymbol{s}}\}$ with $D_{\boldsymbol{s}}$ being a $d \times d$ unit grid and $T = 20$ points of time could be simulated as follows.

```
R> d      <- 4
R> T      <- 20
R> D_s    <- 1:(d^2)
R> D_t    <- 1:T
R> n      <- length(D_s) * length(D_t)
R> nblist <- cell2nb(d, d, type = "queen")
R> W_list <- nb2listw(nblist)
R> W_s    <- Matrix(listw2mat(W_list))
R> W      <- W_s
R> for(t in D_t[-1]){
R>   W    <- bdiag(W, W_s)
R> }
R> diag(W[-c(1:length(D_s)), -c((n - length(D_s)+1):n)]) <- 0.2
R> set.seed(1)
R> Y      <- sim.spARCH(n = n, rho = 0.8, alpha = 1, W = W, type = "log-spARCH")
```

The spatial weighting scheme has been chosen as block diagonal matrix like proposed above, i.e., constant matrices $\mathbf{W}_1 = \mathbf{W}_2 = \ldots = \mathbf{W}_T = \mathbf{W}_s$ along the diagonal define the instantaneous



Note that the true spatial orientation is preserved in this representation (4×4 grid), i.e., plots appearing close to each other are also located close to each other in space and they are, therefore, more related than more distant locations. An alternative representation as consecutive spatial random fields is shown in Figure 4.

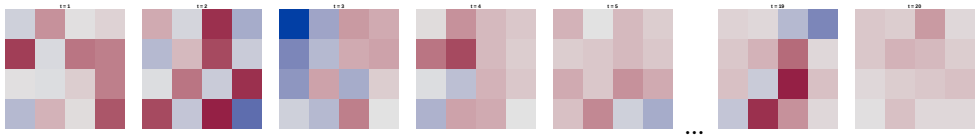
Figure 3: Simulated spatiotemporal log-ARCH process depicted as individual time series. The process has been simulated on a 4×4 spatial unit grid and for 20 points in time. The spatial ARCH parameter ρ equals 0.9 and $\alpha = 1$.

spatial interactions, while constant weights of 0.2 below the diagonal define the extend of the temporal dependence. For instance, a central location would be weighted equally by all eight spatial neighbors with a weight of 0.125 and by the observation of the same location at the previous time point by 0.2. Thus, $\mathbf{W}$ describes the structure and the extend of the dependence in space and time. Eventually, ρ has been chosen as 0.8 and we simulated the process as logarithmic spatial ARCH process.

The resulting simulation is depicted in Figures 3 and 4. Whereas the simulated values are shown as time series plots placed at their correct spatial locations in Figure 3, 4 depicts the observations as consecutive spatial random fields. Note that the same color coding has been chosen for both representations. On the one hand side, one can observe spatial volatility clusters (e.g. in the pre-last plot in Figure 4, $t = 19$, in which the conditional variance is low in the upper left corner, whereas the conditional variance of the remaining locations is high). On the other hand, temporal volatility cluster can be observed as well. For example, at location 16, the variance is high at the first and last five time points, while it is lower between $t = 6$ and $t = 14$.

For sake of completeness, we briefly demonstrate the simulation of spatial GARCH-type models using the `sim.spGARCH()` function. Like for `sim.spARCH()`, the type of the spatial GARCH model can be chosen by the argument `type`. More precisely, there are the following options

- `type = "spGARCH"` for simulation of spatial GARCH models according to the definition in (10),
- `type = "e-spGARCH"` for simulation of exponential spatial GARCH models according to the



The first five and the last two time points are plotted as spatial random fields, i.e., the simulations are shown in their natural temporal ordering. An alternative representation as time series in their true spatial ordering is shown in Figure 3.

Figure 4: Simulated spatiotemporal log-ARCH process depicted as consecutive spatial random fields. The process has been simulated on a 4×4 spatial unit grid and for 20 points in time. The spatial ARCH parameter ρ equals 0.9 and $\alpha = 1$.

definition in (12) with g,

- `type = "log-spGARCH"` for simulation of logarithmic spatial GARCH models according to the definition in (12) with g_b, and
- `type = "complex-spGARCH"` for simulation of a complex-valued spatial GARCH model.

To simulate a spatial GARCH process, two spatial weights matrices need to be specified via the arguments `W1` and `W2`. Moreover, two parameters ρ and λ are passed to the simulation function by the arguments `rho` and `lambda`. For instance, a spatial GARCH model can be simulated on a $d \times d$ spatial unit grid as follows

```
R> require("spdep")
R> rho    <- 0.5
R> lambda <- 0.3
R> alpha  <- 1
R> d      <- 20
R> nblist <- cell2nb(d, d, type = "rook") # Rook's contiguity matrix
R> W_1    <- nb2mat(nblist)
R> W_2    <- W_1
R> Y      <- sim.spGARCH(rho = rho, lambda = lambda, alpha = alpha,
+                        W1 = W_1, W2 = W_2, type = "spGARCH")
```

Similarly, spatial log-GARCH processes and exponential spatial GARCH processes can be simulated by changing the argument `type`. In this case, the parameters b must be provided for the log-GARCH or Θ and ζ for the e-spGARCH, respectively. These parameters can easily passed to `sim.spGARCH()` by the arguments `b`, `theta`, and `zeta`.

Maximum-likelihood estimation

Other important functions of the package are the `qml.spARCH()` and `qml.SARspARCH()` functions, which implement a quasi-maximum-likelihood estimation algorithm (QML). As for the `sim.spARCH()` function, many spARCH models are covered in the `qml.spARCH()` and `qml.SARspARCH()` function. Thus, the user needs to specify which particular spARCH model is to be fitted via the argument `type`. Moreover, the model for the mean equation is a user-specified `formula`, making the use of the estimation functions similar to the use of the common `lm()` or `glm()` functions.

In general, the estimators exhibited good performances for a variety of error distributions in simulation studies, although the likelihood function was derived under the normality assumption. This is not surprising, as the maximum-likelihood estimators have good properties under mild assumptions for the error processes of a variety of similar spatial econometrics models (cf. Lee 2004; Lee and Yu 2012, 2010b,a). Thus, we refer to the approach as the QML approach, and the name of the estimation functions start with `qml` instead of `ml`. In the following paragraphs, we start the simulation of one specific sample, which is then used further to illustrate the log-likelihood functions as well as to demonstrate parameter estimation.

Compared to the log-spARCH processes, the likelihood functions of spARCH models are rather flat around the global maximum. This behavior is illustrated for simulated processes in Figure 5. The observations for the log-spARCH process have been simulated as follows.

```
R> nblist          <- cell2nb(20, 20, type = "queen")
R> W               <- nb2mat(nblist)
```

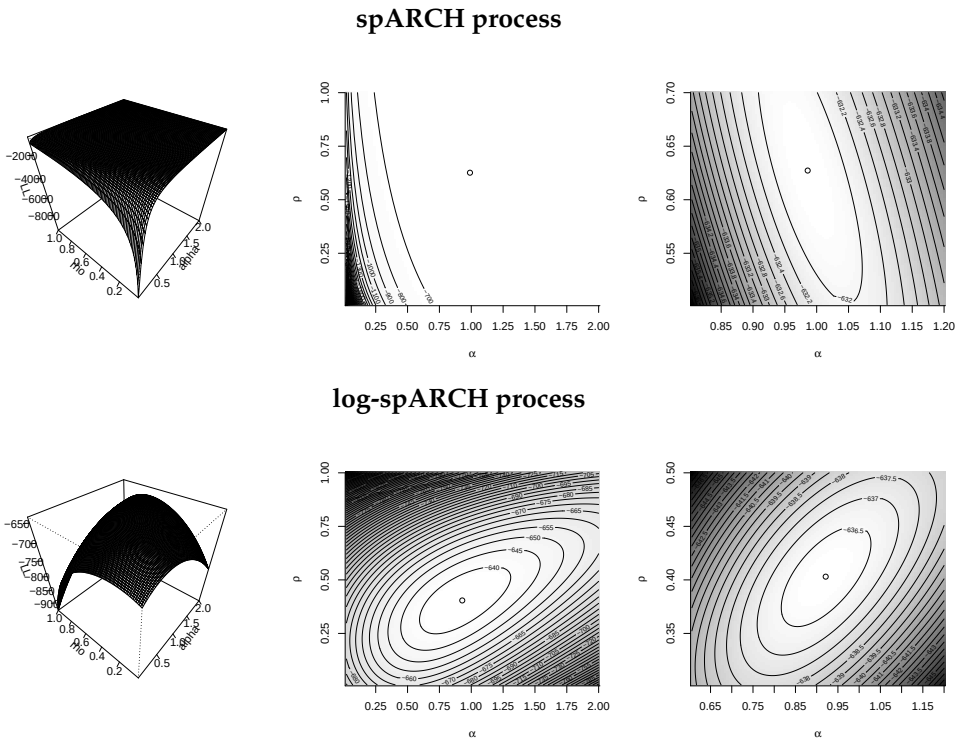


Figure 5: Logarithmic likelihood function.

```
R> y               <- sim.spARCH(n = 20^2, rho = 0.5, alpha = 1, W = W,
+                                type = "log-spARCH", control = list(seed = 5515))
```

To simulate an oriented process, the entries of **W** above the diagonal must be set to zero and the argument `type` must be changed to `"spARCH"`, i.e.,

```
R> W[upper.tri(W)] <- 0
R> y2              <- sim.spARCH(n = 20^2, rho = 0.5, alpha = 1, W = W,
+                                type = "spARCH",
+                                control = list(seed = 5515))
```

To estimate the parameters of an intercept-free log-spARCH model without any regressors, the formula passed to the function `qml.spARCH()` should be specified as `y ~ 0`. In addition, a `data.frame` can be passed via the `data` argument to the `qml` functions. Although the likelihood function of a spARCH process is flat, good estimates can be obtained through iterative maximization. Otto et al. (2018) analyze the performance of the estimators in detail. The algorithm implemented in the packages is based on the **Rsolnp** package, allowing for both equality and inequality parameter constraints (cf. Ghalanos and Theussl 2012).

The results of the estimation procedure are returned via an object of the class 'spARCH', for which we provide additionally several generic functions. First, there is a `summary()` function for the 'spARCH' object. The summary shows all important estimation results, i.e., the parameter estimates, standard errors, test statistics, and asymptotic p-values, including significance stars. The estimation of the above simulated log-spARCH process would return the following results.

```
R> spARCH_object   <- qml.spARCH(y ~ 0, W = W, type = "log-spARCH")
R> summary(spARCH_object)
 Call:
qml.spARCH(formula = y ~ 0, W = W, type = "log-spARCH")

 Residuals:
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-2.6867629 -0.6197315 -0.0053580 -0.0002615  0.5708346  2.8576621

 Coefficients:
      Estimate Std. Error t value  Pr(>|t|)
alpha 0.919324   0.128544  7.1518 8.564e-13 ***
```

```
rho   0.402998   0.056519  7.1304 1.001e-12 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

 AIC: 543.01, BIC: 539.01 (Log-Likelihood: -269.51)

 Moran's I (residuals): -0.028568, p-value: 0.31795

 Moran's I (squared residuals): 0.035239, p-value: 0.14479
```

The standard errors are estimated as Cramer-Rao bounds from the Hessian matrix of the log-likelihood function. For triangular weighting matrices, the estimators are asymptotically normally distributed (Otto et al. 2018). In addition to the Akaike and Bayesian Schwarz information criteria, the results of Moran's test on the residuals and squared residuals are reported for the spatial autocorrelation of the residuals. However, it is possible to use functions like `AIC()` or `BIC()`, since there is a `logLik()` method for the objects from class 'spARCH'. Additionally, the fitted values and residuals can be extracted by `fitted()` and `residuals()`, respectively.

To analyze the residuals, we provide additionally several descriptive plots via the generic `plot()` function. The first two plots are produced by `moran.plot()` imported from the package **spdep**. They inspect the spatial autocorrelation of the residuals and the squared residuals. In addition, the error distribution is depicted in the third graphic by a normal Q-Q-plot. The output obtained for the above numerical example is given below and in Figure 6.

```
%\begin{CodeInput}
%R> AIC(spARCH_object)
%R> BIC(spARCH_object)
%R> par(mfcol = c(1,3))
%R> plot(spARCH_object)
%\end{CodeInput}
R> AIC(spARCH_object)
[1] 543.0126
R> BIC(spARCH_object)
[1] 550.9956
R> par(mfcol = c(1,3))
R> plot(spARCH_object)
Reproduce the results as follows:
         eps <- residuals(x)
         W <- as.matrix(x$W)
         moran.plot(eps, mat2listw(W), zero.policy = TRUE,
     xlab = "Residuals", ylab = "Spatially Lagged Residuals")
Reproduce the results as follows:
         eps <- residuals(x)
         W <- as.matrix(x$W)
         moran.plot(eps, mat2listw(W), zero.policy = TRUE,
     xlab = "Residuals", ylab = "Spatially Lagged Residuals")
Reproduce the results as follows:
         eps <- residuals(x)
         std_eps <- (eps - mean(eps))/sd(eps)
         qqnorm(eps, ylab = "Standardized Residuals")
         qqline(eps)
```

The mean equation can be specified as `formula` for all models, i.e., the spARCH, log-spARCH, and SARspARCH models. Thus, there is a huge variety of possible spatial ARCH models as well as regression models with spARCH residuals which can be fitted by the estimation functions. In addition to linear models of the form `y ~ a + b`, more sophisticated models can also be fitted, e.g., models with interactions `y ~ a + b:c`, factor models `y ~ factor`, polynomial models `y ~ poly(a,3)`, seasonally or regularly varying models of the form `y ~ sin(t) + cos(t)` or `y ~ sin(long) + cos(long) + sin(lat) + cos(lat)`, and so forth. We also implement an `extractAIC()` method for 'spARCH' objects, such that one might also use `step()` for stepwise model selection. Table 3 provides an overview of possible combinations of the arguments `formula` and `type` and shows the resulting models, which can be fitted by the functions `qml.spARCH()` and `qml.SARspARCH()`, respectively.

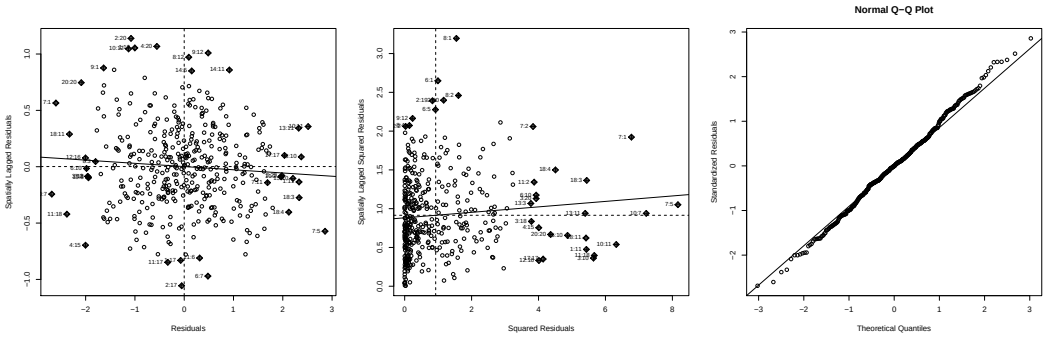


Figure 6: Resulting graphical output of `plot()`.

Function	formula	type	Resulting model
`qml.spARCH()`	`y 0`	`"spARCH"`	spARCH model (see (1) and (2))
`qml.spARCH()`	`y 1`	`"spARCH"`	spARCH model with an additional intercept for the mean equation
`qml.spARCH()`	`y a + b`	`"spARCH"`	Linear Regression with regressors a and b and spARCH residuals
`qml.spARCH()`	`y a + b:c`	`"spARCH"`	Linear Regression with more complex expressions and spARCH residuals
`qml.spARCH()`	`y 0`	`"log-spARCH"`	log-spARCH model (see (1) and (5))
`qml.spARCH()`	`y 1`	`"log-spARCH"`	log-spARCH model with an additional intercept for the mean equation
`qml.spARCH()`	`y a + b`	`"log-spARCH"`	Linear Regression with regressors a and b and log-spARCH residuals
`qml.spARCH()`	`y a + b:c`	`"log-spARCH"`	Linear Regression with more complex expressions and log-spARCH residuals
`qml.SARspARCH()`	`y 0`	`"spARCH"`	SAR model without an intercept, but with spARCH residuals (see (8) and (9))
`qml.SARspARCH()`	`y 1`	`"spARCH"`	SAR model with an intercept and spARCH residuals
`qml.SARspARCH()`	`y a + b`	`"spARCH"`	SAR model with an intercept and the regressors a and b and spARCH residuals
`qml.SARspARCH()`	`y a + b:c`	`"spARCH"`	SAR model with more complex expressions and spARCH residuals
`qml.SARspARCH()`	`y 0`	`"log-spARCH"`	SAR model without an intercept, but with log-spARCH residuals (see (8) and (9))
`qml.SARspARCH()`	`y 1`	`"log-spARCH"`	SAR model with an intercept and log-spARCH residuals
`qml.SARspARCH()`	`y a + b`	`"log-spARCH"`	SAR model with an intercept and the regressors a and b plus log-spARCH residuals
`qml.SARspARCH()`	`y a + b:c`	`"log-spARCH"`	SAR model with more complex expressions and log-spARCH residuals

Table 3: Overview of spatial models, which can be fitted by `qml.spARCH()` and `qml.SARspARCH()`.

Real-data example: prostate cancer incidence rates

Below, the focus is on the incidence rates (2008–2012) for prostate cancer provided by the Centers for Disease Control and Prevention (of Health et al. 2015). In particular, we analyze the incidence rates in all counties of several southeastern U.S. states, namely Arkansas, Louisiana, Mississippi, Tennessee, North and South Carolina, Georgia, Alabama, and Florida. This area also covers the counties along the Mississippi River collectively known as "cancer alley" (see Nitzkin 1992; Brent 2010; Berry 2003). All rates are age-adjusted to the 2000 U.S. standard population (cf. of Health et al. 2015). To reproduce the example, the logarithmic incidence rates as well as several covariates are included in the package.

As explanatory variables, we included a large set of environmental, climate, behavioral, and health covariates, which might have an influence on incidence rates for prostate cancer. For instance, we consider air pollution, such as $PM_{2.5}$, PM_{10}, SO_2, NO_2, CO, O_3, and $CH2O$, as potential environmental hazard factors. Moreover, we account for smoking, drinking, sport activities, and further healthcare-related variables as potential influences on the cancer incidence rates. In total, we account for 34 explanatory variables, which were obtained by inverse-distance-kriging from spatial points processes. Most of the variables are correlated, so we performed a factor analysis on 5 subgroups to identify 10 common factors. The factor loadings are summarized in Table 4. Note that the factor scores are directly included in the dataset `prostate_cancer`. Eventually, the final explanatory factors were chosen by minimizing the Bayesian information criterion using the generic function `step()` as follows.

```
R> data(prostate_cancer)
R> out <- step(qml.SARspARCH(formula, B = B, W = W, type = "spARCH",
+                            data = prostate_cancer), k = log(length(Y)))
```

The `formula` object simply defines a linear model between the logarithmic incidence rates and all factors. Further, matrix **B** describes the predefined spatial dependence structure in the mean equation. For this analysis, **B** has been chosen as a row-standardized contiguity matrix of the direct neighbors. For the spatial dependence in the spatial ARCH term of the residuals, we also included all neighbors up to order 4. Hence, **W** is the row-standardized matrix of the sum of the first-, second-, third-, and fourth-lag neighbors.

By minimizing the BIC criterion, the 2nd and 10th factor has been selected. Whereas the 2nd factor has positive loadings mainly for fine particulate matters, $PM_{2.5}$ and PM_{10}, the 10th describes the tendency for high blood pressure and cholesterol in the county's population. However, note that this analysis is based on aggregated data rather than individual patients; hence, the selected factors cannot be interpreted as carcinogenic factors.

Using the generic `summary()` for the 'spARCH' class, the estimated model can be summarized as follows.

```
 Call:
qml.SARspARCH(formula = formula, B = B, W = W, type = "spARCH",
    data = prostate_cancer)

 Residuals:
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-0.7492270 -0.1079639 -0.0001509 -0.0005261  0.1121190  0.6404564

 Coefficients:
                 Estimate Std. Error t value  Pr(>|t|)
alpha (spARCH)  0.0203839  0.0042674  4.7766 1.783e-06 ***
rho (spARCH)    0.3782104  0.1309656  2.8879  0.003879 **
lambda (SAR)    0.6768133  0.0356765 18.9708 < 2.2e-16 ***
(Intercept)     1.5388985  0.1702222  9.0405 < 2.2e-16 ***
F_2             0.0192857  0.0069917  2.7584  0.005809 **
F_10           -0.0205693  0.0064450 -3.1915  0.001415 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

 AIC: -1734.2, BIC: -1746.2 (Log-Likelihood: 873.11)

 Moran's I (residuals): -0.022899, p-value: 0.32023

 Moran's I (squared residuals): 0.021409, p-value: 0.00050052
```

First, we see that the model has a significant spatial autocorrelation in the mean equation since $\hat{\lambda}$

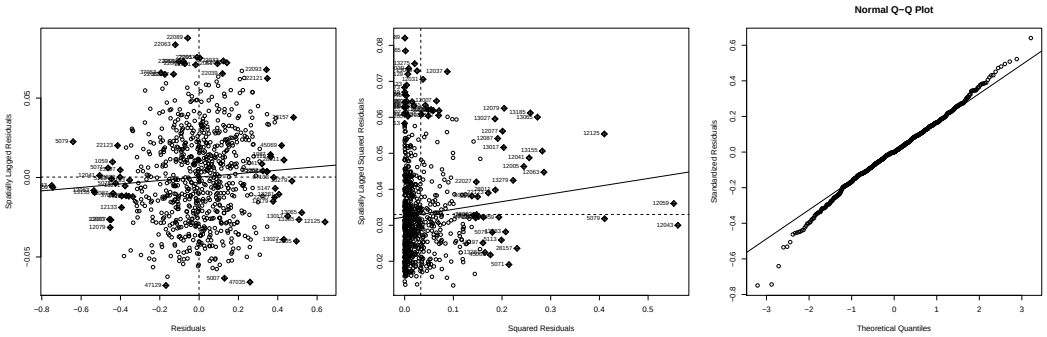


Figure 7: Resulting graphical output of `plot()` for the real-data example.

(`lambda (SAR)`) differs significantly from zero. This implies that there are clusters of higher prostate cancer incidence rates and, vice versa, lower incidence rates. Second, the error process shows conditional, autoregressive heteroscedasticity in space, which is captured by the spARCH component of the model, i.e., $\hat{\rho} = 0.378$ and $\hat{\alpha} = 0.020$. This can be interpreted as differences in the local uncertainty of the model. Hence, there are regions where the model predicts the true incidence rates more accurately, and there are regions with a worse fit. This can also be interpreted as local risks coming from unobserved, hidden factors. Note additionally that it is important to account for spatial conditional heteroscedasticity, as the estimates of spatial autoregressive models are biased if the error variance is not homogeneous across space. Inspecting the residuals, one can see that the spatial autocorrelation has been fully captured by the model, as Moran's I of the residuals is close to zero. In contrast, there is a weak spatial dependence in the squared residuals. To inspect the reason for this dependence graphically, the function `plot()` can be used to produce the plots shown in Figure 7.

After fitting the model, one also may include further regressors or estimate an intercept-only model via `update()`. For illustration, we added the percentage of positive results for a prostate-specific antigen (PSA) test in each county as an additional explanatory variable by

```
R> out2 <- update(out, . ~ . + PSA_test)
```

The PSA test is used for prostate cancer screening, meaning that there should definitely be a positive dependence between the PSA test and the incidence rates. In fact, the estimated parameter is positive, and the AIC is lower compared to the previous model. To be precise, the updated parameters are

```
                   Estimate Std. Error t value  Pr(>|t|)
alpha (spARCH)    0.0199281  0.0043105  4.6231  3.78e-06 ***
rho (spARCH)      0.3902185  0.1280266  3.0479 0.0023041 **
lambda (SAR)      0.6643605  0.0366748 18.1149 < 2.2e-16 ***
(Intercept)       1.1349551  0.2301554  4.9313  8.17e-07 ***
F_2               0.0198504  0.0069903  2.8397 0.0045159 **
F_10             -0.0224035  0.0065828 -3.4034 0.0006656 ***
PSA_test          0.0095962  0.0042728  2.2459 0.0247125 *
```

Summary and discussion

This paper examines spatial models for autoregressive conditional heteroscedasticity. In contrast to previously proposed spatial GARCH models, these models allow for instantaneous autoregressive dependence in the second conditional moments. Previous approaches only allowed for spatial dependence in the first temporal lag. However, these models are also captured by the spatial ARCH approach, since temporal dependence can be included by appropriate choices of the weighting matrix. In addition to discussing previously proposed models, we introduced a novel spatial logarithmic ARCH model, for which the probability density has been derived and maximum-likelihood estimators discussed.

In addition to this theoretical model, we focus on the computational implementation of all considered spatial ARCH models in the R-package **spGARCH**. In particular, the simulation and estimation has been demonstrated. Regarding maximum-likelihood estimation, a broad range of spatial models are implemented in the package. Furthermore, the spatial weights matrices, as well as the mean model, can easily be specified by the user, providing a flexible and easy-to-use tool for spatial ARCH models. All estimation functions return an object for class 'spARCH', for which several generic functions are provided, such as `summary()`, `plot()`, and `AIC()`. This setup also allows the use of the R-base functions,

	F. 1	F. 2	F. 3	F. 4	F. 5	F. 6	F. 7	F. 8	F. 9	F. 10
$PM_{2.5}$ concentration	0.69	0.72								
SO_2 concentration	0.33	-0.03								
NO_2 concentration	0.13	-0.12								
CO concentration	0.31	0.05								
PM_{10} concentration	0.07	0.44								
O_3 concentration	1.00	-0.02								
Solar radiation			0.60	0.44						
Precipitation			-0.08	-0.26						
Outdoor temperature			1.00	-0.05						
Temperature differences			0.32	0.94						
Ambient maximal temperature			0.08	-0.39						
CH_2O	-0.23	0.32								
Percentage of current smokers					0.47	-0.85				
Percentage of former smokers					0.92	0.37				
Smoke some days					-0.07	-0.62				
Never smoked					-0.96	0.25				
Aerobic activity							-0.05	0.58		
Exercises							0.41	0.33		
Physical activity index							-0.09	0.99		
Alcohol consumption					0.04	0.62				
Binge drinking					0.07	0.44				
Heavy drinking					0.43	0.02				
High cholesterol									0.00	1.00
Cholesterol checked									0.55	0.00
Overweight (BMI 25.0-29.9)									0.99	0.09
Obese (BMI 30.0 - 99.8)									-0.75	0.01
Blood stool test									0.56	-0.23
Sigmoidoscopy									0.14	-0.16
High blood pressure									0.03	0.79
Flu shot							0.81	-0.13		
Pneumonia vaccination							0.51	-0.26		
Health care coverage							0.58	0.18		
Seatbelt use							-0.58	0.10		

Table 4: Overview of all included regressors and factor loading for the 10 common factors. The regressors were divided into 5 subgroups to allow for distinctions between the factors.

such as `step()` for stepwise model selection or `update()` for updating the results of different mean models. Eventually, the use of these functions are demonstrated by an empirical example, namely county-level incidence rates of prostate cancer.

In the future, the package should be extended for further spatial ARCH-type models. Along this vein, a class for model specifications should be added alongside the actual implementations via arguments for the fitting functions. In that way, the package can be aligned to common time series ARCH packages, such as the **rugarch** package. Furthermore, the package could benefit from robust estimation methods, another focus for future research.

Bibliography

R. Amin, M. Hendryx, M. Shull, and A. Bohnert. A Cluster Analysis of Pediatric Cancer Incidence Rates in Florida: 2000–2010. *Statistics and Public Policy*, 1(1):69–77, 2014. [p407]

D. Bates and D. Eddelbuettel. Fast and elegant numerical linear algebra using the RcppEigen package. *Journal of Statistical Software*, 52(5):1–24, 2013. URL http://www.jstatsoft.org/v52/i05/. [p407]

G. R. Berry. Organizing against multinational corporate power in cancer alley the activist community as primary stakeholder. *Organization & Environment*, 16(1):3–33, 2003. [p415]

P. J. Bickel and K. A. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*, volume 117. CRC Press, 2015. [p406]

R. Bivand and G. Piras. Comparing implementations of estimation methods for spatial econometrics. *Journal of Statistical Software, Articles*, 63(18):1–36, 2015. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v063.i18. [p401, 408]

T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3): 307–327, 1986. [p401, 402]

S. Borovkova and R. Lopuhaa. Spatial GARCH: A spatial approach to multivariate volatility modeling. *Available at SSRN 2176781*, 2012. [p401, 403]

K. Brent. Gender, race, and perceived environmental risk: The "white male" effect in cancer alley, la. *Sociological Spectrum*, 2010. [p415]

T. Buettner. Tax base effects and fiscal externalities of local capital taxation: Evidence from a panel of german jurisdictions. *Journal of Urban Economics*, 54(1):110–128, 2003. [p407]

M. Cameletti. *Stem: Spatio-Temporal EM*, 2015. R package version 1.0. [p401]

M. Caporin and P. Paruolo. GARCH models with spatial structure. *SIS Statistica*, pages 447–450, 2006. [p401, 403]

A. Cliff and K. Ord. *Spatial Processes: Models & Applications*, volume 44. Pion London, 1981. [p403]

N. Cressie. *Statistics for Spatial Data*. John Wiley & Sons, 1993. URL https://books.google.de/books?id=4L_dCgAAQBAJ. [p401, 408]

N. Cressie and C. K. Wikle. *Statistics for Spatio-Temporal Data*. John Wiley & Sons, 2011. [p401, 402]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL https://doi.org/10.18637/jss.v040.i08. [p407]

J. P. Elhorst. Applied spatial econometrics: Raising the bar. *Spatial Economic Analysis*, 5(1):9–28, 2010. URL https://doi.org/10.1080/17421770903541772. [p401, 402]

R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982. [p401, 402, 403]

F. Finazzi and A. Fasso. D-STEM: a Software for the Analysis and Mapping of Environmental Space-Time Variables. *Journal of Statistical Software*, 62(6):1–29, 2014. [p401]

B. Fingleton. A generalized method of moments estimator for a spatial panel model with an endogenous spatial lag and spatial moving average errors. *Spatial Economic Analysis*, 3(1):27–44, 2008. URL https://doi.org/10.1080/17421770701774922. [p405]

J. Geweke. Comment on modelling the persistence of conditional variances. *Econometric Reviews*, (1): 57–61, 1986. [p401]

A. Ghalanos. *Rugarch: Univariate GARCH Models.*, 2018. R package version 1.4-0. [p401]

A. Ghalanos and S. Theussl. *Rsolnp: General Non-Linear Optimization Using Augmented Lagrange Multiplier Method*, 2012. R package version 1.14. [p407, 412]

R. P. Haining. The moving average model for spatial interaction. *Transactions of the Institute of British Geographers*, 3(2):202–225, 1978. [p405]

D. A. Harville. *Matrix Algebra from a Statistician's Perspective*, volume 1. Springer-Verlag, 2008. [p407]

H. H. Kelejian and I. R. Prucha. Specification and estimation of spatial autoregressive models with autoregressive and heteroskedastic disturbances. *Journal of Econometrics*, 157(1):53–67, 2010. [p405]

C. Lam and P. C. Souza. Detection and estimation of block structure in spatial weight matrix. *Econometric Reviews*, 35(8-10):1347–1376, 2016. [p407]

L.-F. Lee. Asymptotic distributions of quasi-maximum likelihood estimators for spatial autoregressive models. *Econometrica*, 72(6):1899–1925, 2004. URL https://doi.org/10.1111/j.1468-0262.2004.00558.x. [p411]

L.-F. Lee and J. Yu. Some recent developments in spatial panel data models. *Regional Science and Urban Economics*, 40(5):255–271, 2010a. [p411]

L.-F. Lee and J. Yu. A spatial dynamic panel data model with both time and individual fixed effects. *Econometric Theory*, 26(2):564–597, 2010b. URL https://doi.org/10.1017/s0266466609100099. [p411]

L.-F. Lee and J. Yu. QML Estimation of Spatial Dynamic Panel Data Models with Time Varying Spatial Weights Matrices. *Spatial Economic Analysis*, 7(1):31–74, 2012. URL https://doi.org/10.1080/17421772.2011.647057. [p411]

T. G. Martins, D. Simpson, F. Lindgren, and H. Rue. Bayesian computing with INLA: New features. *Computational Statistics & Data Analysis*, 67:68–83, 2013. [p401]

M. S. Merk and P. Otto. Estimation of anisotropic, time-varying spatial spillovers of fine particulate matter due to wind direction. *Geographical Analysis*, 2019. URL https://doi.org/10.1111/gean.12205. [p407]

A. Milhøj. *A multiplicative parameterization of ARCH models*. Unpublished manuscript, 1987. [p401]

P. A. P. Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37:17–23, 1950. [p403]

D. B. Nelson. Conditional heteroskedasticity in asset returns: A new approach. *Econometrica: Journal of the Econometric Society*, pages 347–370, 1991. [p403]

J. L. Nitzkin. Cancer in louisiana: A public health perspective. *The Journal of the Louisiana State Medical Society: official organ of the Louisiana State Medical Society*, 144(4):162–162, 1992. [p415]

U. S. D. of Health, C. for Disease Control Human Services, Prevention, and ational Cancer Institute. United States Cancer Statistics 1999-2012 Incidence and Mortality Web-Based Report, 2015. [p415]

P. Otto and W. Schmid. Spatial and spatiotemporal GARCH models - A unified approach. *arXiv preprint arXiv:1908.08320*, 2019. [p406]

P. Otto and R. Steinert. Estimation of the spatial weighting matrix for spatiotemporal data under the presence of structural breaks. *arXiv preprint arXiv:1810.06940*, 2018. [p407]

P. Otto, W. Schmid, and R. Garthoff. Generalized spatial and spatiotemporal autoregressive conditional heteroscedasticity. *arXiv preprint arXiv:1609.00711*, 2016. [p401]

P. Otto, W. Schmid, and R. Garthoff. Generalised spatial and spatiotemporal autoregressive conditional heteroscedasticity. *Spatial Statistics*, 26:125–145, 2018. [p401, 402, 403, 404, 405, 407, 408, 412, 413]

P. Otto, W. Schmid, and R. Garthoff. Stochastic properties of spatial and spatiotemporal arch models. *Statistical Papers*, 2019. ISSN 1613-9798. URL https://doi.org/10.1007/s00362-019-01106-x. [p401]

S. G. Pantula. Comment on modelling the persistence of conditional variances. *Econometric Reviews*, 5 (1):71–74, 1986. [p401]

E. J. Pebesma. Multivariable geostatistics in S: The gstat package. *Computers & Geosciences*, 30:683–691, 2004. [p401]

H. Rue, S. Martino, and N. Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *J. of the Royal Statistical Society-B*, 71(2):319–392, 2009. [p401]

T. Sato and Y. Matsuda. Spatial autoregressive conditional heteroskedasticity models. *Journal of the Japan Statistical Society*, 47(2):221–236, 2017. [p401]

T. Sato and Y. Matsuda. Spatiotemporal ARCH models. Technical report, Graduate School of Economics and Management, Tohoku University, 2018a. [p401]

T. Sato and Y. Matsuda. Spatial GARCH models. Technical report, Graduate School of Economics and Management, Tohoku University, 2018b. [p401]

M. Tiefelsdorf and B. Boots. The exact distribution of Moran's I. *Environment and Planning A*, 27(6): 985–999, 1995. [p403]

Y. Ye. *Interior Algorithms for Linear, Quadratic, and Linearly Constrained Non-Linear Programming*. PhD thesis, Department of ESS, Stanford University, 1988. [p407]

Appendix

Proof of Theorem 2. For this definition of g_b, one could rewrite $\ln \boldsymbol{h}$ as

$$\ln \boldsymbol{h}_E = \mathbf{S}\left(\alpha \mathbf{1} + \rho b \mathbf{W} \ln |\boldsymbol{Y}|\right) \tag{14}$$

with

$$\mathbf{S} = (s_{ij})_{i,j=1,\ldots,n} = \left(\mathbf{I} + \frac{1}{2}\rho b \mathbf{W}\right)^{-1}.$$

Since $w_{ij} \geq 0$ for all $i, j = 1, \ldots, n$, $\mathbf{W}$ is positive definite and it holds that

$$\det\left(\mathbf{I} + \frac{1}{2}\rho b \mathbf{W}\right) \geq 1 + \frac{1}{2}\rho b \det(\mathbf{W}) > 0\,.$$

Thus, the relation between $Y(\boldsymbol{s}_1), \ldots, Y(\boldsymbol{s}_n)$ and $\varepsilon(\boldsymbol{s}_1), \ldots, \varepsilon(\boldsymbol{s}_n)$ is given by (1) and (14). □

Proof of Corollary 1. For $\rho \geq 0$, $b \geq 0$, and $w_{ij} \geq 0$ for all i, j, the inverse

$$\mathbf{S} = (s_{ij})_{i,j=1,\dots,n} = \left(\mathbf{I} + \frac{1}{2}\rho b \mathbf{W}\right)^{-1}.$$

is a non-negative matrix. Thus,

$$\ln \boldsymbol{h}_E = \mathbf{S}\left(\alpha \mathbf{1} + \rho b \mathbf{W} \ln |\boldsymbol{Y}|\right)$$

is positive for $\alpha > 0$. □

Philipp Otto
Leibniz University Hannover
Appelstraße 9a
30167 Hannover
Germany
otto@ikg.uni-hannover.de

lpirfs: An R Package to Estimate Impulse Response Functions by Local Projections

by Philipp Adämmer

Abstract Impulse response analysis is a cornerstone in applied (macro-)econometrics. Estimating impulse response functions using local projections (LPs) has become an appealing alternative to the traditional structural vector autoregressive (SVAR) approach. Despite its growing popularity and applications, however, no R package yet exists that makes this method available. In this paper, I introduce **lpirfs**, a fast and flexible R package that provides a broad framework to compute and visualize impulse response functions using LPs for a variety of data sets.

Introduction

Since the seminal paper of Sims (1980), analysing economic time series by Vector Auto Regressive (VAR) models has become a main pillar in empirical macroeconomic analysis. VARs have been traditionally used to recover structural shocks in order to estimate their propagating effects on economic variables. This approach, however, has been criticized for several drawbacks such as the imposed dynamics on the (economic) system, the curse of dimensionality and the more difficult application to nonlinearities (Auerbach and Gorodnichenko, 2013).

Estimating impulse response functions using local projections (LPs) has become an appealing alternative, which is reflected in the over 1,000 citations of the pioneering paper by Jordà (2005). Instead of extrapolating the parameters into increasingly distant horizons, LPs estimate the parameters sequentially at each point of interest. It is argued that LPs offer three advantages over the traditional structural vector autoregressive (SVAR) approach: first, LPs are easier to estimate since they rely solely on simple linear regressions; second, the point or joint-wise inference is easily conducted; and third, impulse responses that are estimated using LPs are more robust when a (linear) VAR is misspecified (Jordà, 2005). Although the latter argument has been questioned by Kilian and Kim (2011), the recent study by Brugnolini (2018) shows equal and even better performance of LPs when the lag lengths for each forecast horizon are adequately fixed. Yet, Plagborg-Møller and Wolf (2019) proved that LPs and VAR models estimate the same impulse responses when the lag structures are unrestricted. This finding implies that empirical impulse responses that are estimated using LPs and SVARs are likely similar at short horizons but differ at longer ones.[1]

Since their introduction in 2005, LPs have been broadly applied to investigate, among others, the macroeconomic effects of oil price shocks (Hamilton, 2011); state-dependent government spending multipliers (Owyang et al., 2013; Auerbach and Gorodnichenko, 2012, 2013); the effects of monetary policy on financial markets and economic aggregates (Tenreyro and Thwaites, 2016; Swanson, 2017; Jordà et al., 2019); and the link between credit growth, monetary policy, house prices, and financial stability (Jordà et al., 2015; Favara and Imbs, 2015; Jordà and Taylor, 2016). Apart from the different research questions, these studies further differ regarding the data structures because they use panel and nonpanel data.

Despite the rising popularity and applications, no R package yet exists that can estimate impulse responses using LPs. The only exception is the code for the smooth LP approach by Barnichon and Brownlees (2019). The approach reduces the variance of the LP parameters with a linear B-spline basis function because LP coefficients can suffer from high variance, sometimes making the interpretation more difficult. The code is partly available on GitHub and has been applied by Garín et al. (2019). The **vars** package by Pfaff (2008) only allows estimating impulse response functions that are based on the traditional SVAR approach.

As a remedy, this paper introduces **lpirfs** (Adämmer, 2019), a fast and flexible R package that enables estimating and visualizing impulse responses using LPs for a variety of data sets. The first part of this paper outlines the theory of LPs and the differences from the traditional SVAR approach. The second part outlines the main functions and options of the package, and the last section applies **lpirfs** by replicating the empirical results from the economic literature.

[1] The appendix contains a comparison of impulse responses that are estimated using LPs and the traditional SVAR approach.

Estimating impulse response functions using local projections

An SVAR with n variables can be written as follows:

$$\begin{pmatrix} \beta_{11}^0 & \cdots & \beta_{1n}^0 \\ \vdots & \ddots & \vdots \\ \beta_{n1}^0 & \cdots & \beta_{nn}^0 \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}_t = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} + \begin{pmatrix} \beta_{11}^1 & \cdots & \beta_{1n}^1 \\ \vdots & \ddots & \vdots \\ \beta_{n1}^1 & \cdots & \beta_{nn}^1 \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}_{t-1} + \cdots + \begin{pmatrix} \beta_{11}^p & \cdots & \beta_{1n}^p \\ \vdots & \ddots & \vdots \\ \beta_{n1}^p & \cdots & \beta_{nn}^p \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}_{t-p} + \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}_t,$$

which more concisely becomes the following:

$$\mathbf{B_0 Y}_t = \boldsymbol{\alpha}_t + \mathbf{B}(\mathbf{L})\mathbf{Y}_t + \boldsymbol{\varepsilon}_t.$$

The residuals $\boldsymbol{\varepsilon}_t$ are assumed to be white noise with zero mean.[2] This representation is appealing from an economic perspective because the structural shocks are contemporaneously uncorrelated, and the variables have a contemporaneous effect on each other. The contemporaneous effect is measured by the square matrix $\mathbf{B_0}$. However, estimating this SVAR without further assumptions is not possible because of the simultaneous identification problem. Merely assuming that the structural shocks are orthogonal does not fully identify the system.

The SVAR in reduced form (henceforth VAR) equals:

$$\mathbf{Y}_t = \tilde{\boldsymbol{\alpha}} + \tilde{\boldsymbol{B}}(\boldsymbol{L})\mathbf{Y}_t + \boldsymbol{u}_t,$$

where

$$\tilde{\boldsymbol{\alpha}} = \mathbf{B_0}^{-1}\boldsymbol{\alpha},\ \tilde{\boldsymbol{B}}(\boldsymbol{L}) = \mathbf{B_0}^{-1}\mathbf{B}(\mathbf{L})$$

and

$$E[\boldsymbol{u}_t, \boldsymbol{u}_\tau^{'}] = \begin{cases} \begin{pmatrix} \sigma_1^2 & \cdots & \sigma_{1,n}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{n,1} & \cdots & \sigma_n^2 \end{pmatrix}, & \text{for } t = \tau \\ 0, & else. \end{cases}$$

The coefficient matrix $\tilde{\boldsymbol{B}}(\boldsymbol{L})$ is a nonlinear function of the contemporaneous parameter matrix $\mathbf{B_0}$ and the structural parameter matrix $\boldsymbol{B}(\boldsymbol{L})$. In contrast to the SVAR, the VAR residuals $\boldsymbol{u}_t$ are contemporaneously correlated, which impedes an unbiased economic interpretation. The VAR residuals are assumed to be linked to the SVAR shocks by the following:

$$\mathbf{u_t} = \mathbf{B_0}^{-1}\boldsymbol{\varepsilon_t}, \qquad E[\mathbf{u_t}\mathbf{u_t}^{'}] = \boldsymbol{\Sigma}_\mathbf{u} = \mathbf{B_0}^{-1}\mathbf{B_0}^{-1'}.$$

Given that the covariance matrix of $\boldsymbol{\varepsilon}_t$ equals the identity matrix, one must still impose $n(n-1)/2$ restrictions to estimate the structural form. The most general approach is to separate the residuals into orthogonal shocks by calculating a Cholesky decomposition of the covariance matrix Σ_u. The first variable in such a system responds to its own exogenous shock, the second variable to the first variable plus an exogenous shock to the second variable, and so on. The results thus depend on ordering (Keating, 1992). The Wold representation states that any covariance-stationary time series can be rewritten as a sum of present and past innovations. This theorem enables mapping the estimated VAR(p) coefficients recursively to the infinite-order vector moving-average coefficients (Brugnolini, 2018). Impulse response functions are estimated iteratively by rewriting VAR(p) into its companion form (i.e., a VAR(1)):

[2] The assumption of independent and identically distributed innovations is common in applied work but can be relaxed (Kilian and Kim, 2011).

$$
\begin{aligned}
\hat{IR}(0) &= \mathbf{B_0}^{-1} \\
\hat{IR}(1) &= \mathbf{\Phi^1 B_0}^{-1} \\
\hat{IR}(2) &= \mathbf{\Phi^2 B_0}^{-1} \\
&\vdots ,
\end{aligned}
$$

where the matrix $\mathbf{\Phi}$ contains the coefficients of the VAR(1).

In his pioneering paper, Jordà (2005) proposed an alternative approach to estimate impulse responses. His first step consists of ordinary least squares (OLS) regressions for each forecast horizon:

$$\mathbf{y}_{t+h} = \boldsymbol{\alpha}^h + \mathbf{B}_1^h \mathbf{y}_{t-1} + \cdots + \mathbf{B}_p^h \mathbf{y}_{t-p} + \mathbf{u}_{t+h}^h, \quad h = 0, 1, \ldots, H-1, \tag{1}$$

where $\boldsymbol{\alpha}^h$ is a vector of constants, and $\mathbf{B}_i^h$ are parameter matrices for lag p and forecast horizon h. The vector elements $\mathbf{u}_{t+h}^h$ are autocorrelated and/or heteroscedastic disturbances. The collection of all regressions of Eq. (1) are called LPs. The slope matrix $\mathbf{B}_1^h$ can be interpreted as the response of $\mathbf{y}_{t+h}$ to a reduced form shock in t (Kilian and Kim, 2011). Structural impulse responses are then estimated by the following:

$$\hat{IR}(t, h, \mathbf{d}_i) = \hat{\mathbf{B}}_1^h \mathbf{d}_i,$$

where $\mathbf{d}_i = \mathbf{B}_0^{-1}$. As in the SVAR approach, the shock matrix $\mathbf{d}_i$ must be identified from a linear VAR. The LP approach thus does not overcome the problem of identification. Given the serial correlation of $\mathbf{u}_{t+h}^h$, Jordà (2005) proposed to estimate robust standard errors using the approach by Newey and West (1987).

A great advantage of LPs is their easy extension to nonlinear frameworks. The simplest approach to separate data into two regimes is using a binary (dummy) variable. The drawback, however, is that it lowers the degrees of freedom. As a remedy, Auerbach and Gorodnichenko (2012) proposed computing state probabilities with a logistic function that allows using all observations for the estimations. The logistic function equals the following:

$$F(z_t) = \frac{e^{(-\gamma z_t)}}{\left(1 + e^{(-\gamma z_t)}\right)}, \tag{2a}$$

$$var(z_t) = 1, \; E(z_t) = 0, \tag{2b}$$

where z_t is standardized so that γ (> 0) is scale-invariant. The value of γ must be provided by the user. For example, if z_t corresponds to changes in the gross domestic product (GDP) at time t, an increase in z_t would lead to a decrease in $F(z_t)$. Values close to zero of $F(z_t)$ would thus indicate periods of economic expansion. Auerbach and Gorodnichenko (2013) proposed standardizing the cyclical components of the filter according to the method by Hodrick and Prescott (1997) to obtain the variable z_t. The observations for the two regimes are the product of the transition function and the endogenous variables:

$$
\begin{aligned}
&\text{Regime 1 } (R_1) : \mathbf{y}_{t-l} \cdot (1 - F(z_{t-1})), \qquad l = 1, \ldots, p, \\
&\text{Regime 2 } (R_2) : \mathbf{y}_{t-l} \cdot F(z_{t-1}), \qquad l = 1, \ldots, p.
\end{aligned} \tag{3}
$$

Auerbach and Gorodnichenko (2012) used the values of the transition function at $t-1$ to avoid contemporaneous feedback from policy actions regarding whether the economy is in a recession or an expansion. Structural nonlinear impulse responses are estimated using the following:

$$
\begin{aligned}
\hat{IR}^{R_1}(t, h, \mathbf{d}_i) &= \hat{\mathbf{B}}_{1,R_1}^h \mathbf{d}_i, \qquad h = 0, \ldots, H-1, \\
\hat{IR}^{R_2}(t, h, \mathbf{d}_i) &= \hat{\mathbf{B}}_{1,R_2}^h \mathbf{d}_i, \qquad h = 0, \ldots, H-1,
\end{aligned}
$$

where $\hat{\mathbf{B}}_{1,R1}^0 = I$ and $\hat{\mathbf{B}}_{1,R2}^0 = I$. The coefficient matrices $\hat{\mathbf{B}}_{1,R_1}^h$ and $\hat{\mathbf{B}}_{1,R_2}^h$ are obtained from the

following LPs:

$$\begin{aligned}\mathbf{y}_{t+h} = \boldsymbol{\alpha}^h + \mathbf{B}^h_{1,R_1}\left(\mathbf{y}_{t-1}\cdot(1-F(z_{t-1}))\right) + \ldots + \mathbf{B}^h_{p,R_1}\left(\mathbf{y}_{t-p}\cdot(1-F(z_{t-1}))\right) + \\ \mathbf{B}^h_{1,R_2}\left(\mathbf{y}_{t-1}\cdot F(z_{t-1})\right) + \ldots + \mathbf{B}^h_{p,R_2}\left(\boldsymbol{y}_{t-p}\cdot F(z_{t-1})\right) + \mathbf{u}^h_{t+h},\end{aligned} \tag{4}$$

with $h = 0, \ldots, H-1$. This nonlinear approach has been used by Ahmed and Cassou (2016) to investigate the effect of consumer confidence on durable goods during periods of economic expansion and recession.

Estimating impulse responses with an identified shock

Besides the easy extension to nonlinear frameworks, another advantage of LPs is their flexible application to situations in which an exogenous shock can be identified outside of an SVAR. For example, Ramey and Zubairy (2018) constructed a military news shock to investigate whether US government spending multipliers are higher during periods of economic slack or when interest rates are near the zero lower bound. Once an exogenous shock has been identified, impulse responses can be directly estimated using OLS regressions:

$$y_{t+h} = \alpha^h + \beta_h shock_t + \phi x_t + u^h_{t+h}, \quad h = 0, 1, \ldots, H-1,$$

where α^h denotes the regression constant, x_t is a vector of control variables, and $shock_t$ is the identified shock variable. The coefficient β_h corresponds to the response of y at time $t+h$ to the shock variable (*shock*) at time t. The impulse responses are the sequence of all estimated β_h. As above, robust standard errors can be estimated using the approach by Newey and West (1987). If the shock variable is endogenous, $shock_t$ can be estimated using the two-stage least squares (2SLS) regression. In the case of nonlinearities, the variables can either be multiplied with a dummy variable or with the values of the transition function in Eq. (3).

Estimating impulse responses for panel data

Another advantage of LPs is that they can be applied to panel data as well. Estimating impulse responses based on panel data have been put forward by Auerbach and Gorodnichenko (2013), Owyang et al. (2013), and Jordà et al. (2015), among others. The general equation for panel data is the following:

$$y_{i,t+h} = \alpha_{i,h} + shock_{i,t}\beta_h + x_{i,t}\gamma_h + \varepsilon_{i,t+h}, \quad h = 0, 1, \ldots, H-1,$$

where $\alpha_{i,h}$ denotes (cross-section) fixed effect, $x_{i,t}$ is a vector of control variables, and $shock_{i,t}$ denotes the identified shock variable. Besides using the absolute values of y_t, **lpirfs** also allows estimating cumulative impulse responses using $(y_{i,t+h} - y_{i,t-1})$ as the endogenous variable, which is often done for panel data (see, e.g., Jordà et al., 2015). Similar to the univariate approach, $shock_t$ can also be first estimated by an instrument variable approach (see, e.g., Jordà et al., 2019). It is further crucial to account for heteroskedasticity and autocorrelation in panel models. The importance of robust standard errors in the context of corporate finance and asset pricing has been shown by Petersen (2009).

The lpirfs package

lpirfs enables estimating all of the above models and specifications. The main functions of the package are the following:

i.) `lp_lin()` and `lp_nl()`, which estimate linear and nonlinear impulse responses based on structural VARs,

ii.) `lp_lin_iv()` and `lp_nl_iv()`, which estimate linear and nonlinear impulse responses for a shock that has been identified outside of the VAR, and

iii.) `lp_lin_panel()` and `lp_nl_panel()`, which estimate linear and nonlinear impulse responses for panel data.

The functions `lp_lin()` and `lp_nl()` estimate linear and nonlinear impulse responses based on SVARs whose shocks are identified by the Cholesky decomposition. The functions `lp_lin_iv()` and `lp_nl_iv()` allow estimating impulse responses when a shock has been identified outside of the VAR. The functions `lp_lin_panel()` and `lp_nl_panel()` can be used for panel data.

Nonpanel functions rely on several routines written with **Rcpp** by Eddelbuettel et al. (2011), making the computations very fast. The functions for panel data are based on the well-established **plm** package by Croissant and Millo (2008). Parallel computation, which is optional and available for all functions, can further reduce the computation time.

Nonpanel functions allow computing ordinary and heteroskedasticity and autocorrelation consistent (HAC) estimators for the impulse responses based on the approach by Newey and West (1987). By default, **lpirfs** increases the truncation parameter with the number of horizons h, but a fixed value can also be manually provided. In addition, pre-whitening is another option that might improve the confidence interval coverage (Andrews and Monahan, 1992). The user can also apply an information criterion for each forecast horizon to find the optimal number of lags. The included criteria are those developed by Akaike (1974), Schwarz (1978), and Hurvich and Tsai (1989). The endogenous, exogenous, and switching variables must be given separately as a `data.frame`. Table 1 summarizes the input options of the nonpanel functions.

Applying **lpirfs** to panel data works slightly differently than using nonpanel functions due to the dependency on the **plm** package. Instead of providing the endogenous and exogenous variables separately, the user must provide the entire panel data set first, and then give the column names for the endogenous, exogenous, and other variables. The default is to estimate a fixed-effects model, but all options available for the **plm** package are also available within **lpirfs**. Table 2 summarizes the input options for the linear and nonlinear panel functions.

Each function output becomes an S3 object, which enables using the generic R functions `plot()` and `summary()`. In addition, the package also contains the functions `plot_lin()` and `plot_nl()`, which enable creating individual graphs of impulse responses.

Table 1: Comparison of linear and nonlinear LP functions.

	Function names				
Input name	`lp_lin()`	`lp_nl()`	`lp_lin_iv()`	`lp_nl_iv()`	Input description
endog_data	✓	✓	✓	✓	Data.frame with endogenous variables for VAR model.
shock	-	-	✓	✓	One column data.frame with the identified shock.
use_twosls	-	-	✓	-	Option to estimate shock with 2SLS approach.
instrum	-	-	✓	-	Data.frame with the instrument(s) for the 2SLS approach.
lags_endog_lin	✓	✓	✓	-	Number of lags for linear model.
lags_endog_nl	-	✓	-	✓	Number of lags for nonlinear model in Eq. (4).
lags_criterion	✓	✓	✓	✓	Choose lags based on information criterion (*AICc, AIC* or *BIC*).
max_lags	✓	✓	✓	✓	Maximum number of lags for information criterion.
trend	✓	✓	✓	✓	Options to include constant, trend and quadratic trend.
shock_type	✓	✓	-	-	Two types of shock: standard deviation or unit shock.
use_nw	✓	✓	✓	✓	Option to estimate standard errors by Newey and West (1987).
nw_lag	✓	✓	✓	✓	Option to manually fix the truncation parameter.
nw_prewhite	✓	✓	✓	✓	Option for pre-whitening (Andrews and Monahan, 1992).
adjust_se	✓	✓	✓	✓	Option to adjust standard errors for small samples.
confint	✓	✓	✓	✓	Value of width for confidence bands.
hor	✓	✓	✓	✓	Number of horizons for impulse responses.
switching	-	✓	-	✓	Switching variable z_t. See Eq. (2).
lag_switching	-	✓	-	✓	Option to lag the values of the logistic function $F(z_t)$.
use_logistic	-	✓	-	✓	Option to use the logistic function. See Eq. (2).
use_hp	-	✓	-	✓	Option to use the filter by Hodrick and Prescott (1997).
lambda	-	✓	-	✓	Value of λ for the HP-filter. See Ravn and Uhlig (2002).
gamma	-	✓	-	✓	Value of γ. See Eq. (2a).
exog_data	✓	✓	✓	✓	Optional data for exogenous variables.
lags_exog	✓	✓	✓	✓	Number of lags for exogenous variables.
contemp_data	✓	✓	✓	✓	Variables with contemporaneous impact.
num_cores	✓	✓	✓	✓	Option to choose number of cores.

The table compares the options for `lp_lin()`, `lp_nl()`, `lp_lin_iv()`, and `lp_nl_iv()`. The symbol ✓ indicates whether the option, denoted by the row, is available for the function, denoted by the column. The optional lag length criteria are those by Hurvich and Tsai (1989), Akaike (1974) and Schwarz (1978).

Table 2: Comparison of linear and nonlinear LP functions for panel data.

	Function names		
Input name	`lp_lin_panel()`	`lp_nl_panel()`	Input description
data_set	✓	✓	A data.frame, containing the panel data set.
data_sample	✓	✓	Option to estimate a subset of the data.
endog_data	✓	✓	Character name of the endogenous variable.
cumul_mult	✓	✓	Option to estimate cumulative multipliers.
shock	✓	✓	Character name of the variable to shock with.
diff_shock	✓	✓	Option to use first differences of the shock variable.
iv_reg	✓	-	Option to use instrument variable approach.
instrum	✓	-	The name(s) of the instrument variable(s).
panel_model	✓	✓	Option to choose type of panel model. See **plm** package.
panel_effect	✓	✓	The effects introduced in the panel-model. See **plm** package.
robust_cov	✓	✓	Options for robust covariance matrix estimator. See **plm** package.
robust_method	✓	✓	Option for robust_cov. See **plm** package.
robust_type	✓	✓	Option for robust_cov. See **plm** package.
robust_cluster	✓	✓	Option for robust_cov. See **plm** package.
robust_maxlag	✓	✓	Option for robust_cov. See **plm** package.
use_gmm	✓	✓	Option to use GMM for estimation.
gmm_model	✓	✓	Option to use "onestep" or "twosteps" approach. See **plm** package.
gmm_effect	✓	✓	The effects introduced in the panel-model. See **plm** package.
gmm_transformation	✓	✓	Additional option for GMM model. See **plm** package.
c_exog_data	✓	✓	Name(s) of the exogenous variable(s) with contemporaneous impact.
l_exog_data	✓	✓	Name(s) of the exogenous variable(s) with lagged impact.
lags_exog_data	✓	✓	Lag length for the exogenous variable(s) with lagged impact.
c_fd_exog_data	✓	✓	Exogenous variable(s) with contemporaneous impact of first differences.
l_fd_exog_data	✓	✓	Exogenous variable(s) with lagged impact of first differences.
lags_fd_exog_data	✓	✓	Number of lags for variable(s) with impact of first differences.
confint	✓	✓	Value of width for confidence bands.
switching	-	✓	Column name of the switching variable.
use_logistic	-	✓	Option to use the logistic function. See Eq. (2).
use_hp	-	✓	Option to use the filter by Hodrick and Prescott (1997) to obtain z_t.
lag_switching	-	✓	Option to lag the values of the logistic function $F(z_t)$.
lambda	-	-	Value of λ for the HP-filter. See Ravn and Uhlig (2002).
gamma	-	-	Value of γ. See Eq. (2a).
hor	✓	✓	Number of horizons for impulse responses.

The table compares the options for `lp_lin_panel()` and `lp_nl_panel()`. The symbol ✓ indicates whether the option, denoted by the row, is available for the function, denoted by the column. The functions estimate linear and nonlinear impulse responses for models with panel data.

Examples and replications

In this section, I apply all the main functions of **lpirfs** to three different settings. The impulse responses are visualized by the generic `plot()` function, which serves as a wrapper for the built-in functions `plot_lin()` and `plot_nl()`.

Two exercises replicate empirical results by Jordà (2005) and Ramey and Zubairy (2018). The data sets are included in **lpirfs**. The third example uses the external *Jordà-Schularick-Taylor Macrohistory Database*, which covers 17 advanced economies since 1870 on an annual basis and comprises 25 real and nominal variables. I estimate how an increase in the interest rate affects mortgage lending. This example is based on a STATA code provided on Oscar Jordá's website. Due to copyright issues, the database could not be included in the package, but I show how it can be easily downloaded with R.

Traditional approach: Replicating results by Jordà (2005)

The following code replicates parts of Figure 5 in Jordà (2005, p. 176). It shows how the output gap, inflation rate, and federal funds rate react to the corresponding structural shocks. The results are shown in Figure 1.[3] **lpirfs** follows the convention by Jordà (2005), namely that the first horizon, denoted on the x-axis, equals $h = 0$. Applying the generic function `summary()` to the output returns a list of several matrices with OLS diagnostics. The first level of the list corresponds to the shock variable and the second level to the horizon. Table 3 shows OLS diagnostics for the first horizon of the first shock (output gap).

[3] In the appendix, I compare these results with those estimated by a general SVAR.

```
# --- Code to replicate Figure 5 in Jordá (2005, p. 176)

# Load packages
  library(lpirfs)

# Load data set
  endog_data  <- interest_rules_var_data

# Estimate linear model
  results_lin <- lp_lin(endog_data     = endog_data,
                        lags_endog_lin = 4,
                        trend          = 0,
                        shock_type     = 1,
                        confint        = 1.96,
                        hor            = 12)

# Show impulse responses
  plot(results_lin)

# Show OLS diagnostics for the first shock of the first horizon
  summary(results_lin)[[1]][1]


# --- End example code
```

Table 3: OLS diagnostics shown by using `summary()`.

	R.sqrd.	Adj. R.sqrd.	F.stat	p.value
h 1: GDP_gap	0.91	0.90	146.88	0.00
h 1: Infl	0.84	0.83	77.17	0.00
h 1: FF	0.94	0.93	218.74	0.00

The table shows OLS diagnostics for the Jordà (2005) example of the first horizon for the first identified shock (output gap).

The example above only estimates impulse responses for the linear case, but Jordà (2005) also tested for nonlinearities. Although he found no "business-cycle" asymmetries, he identified significant asymmetries for several lags of both inflation and the federal funds rate. The following code uses a dummy approach to estimate the nonlinear impulse responses of the variables to a shock in the federal funds rate. Jordà (2005) used a threshold of 4.75% for the inflation rate, applied to its third lag. Figure 2 shows the empirical results for the nonlinear example. The results are comparable to the findings by Jordà (2005), namely that the magnitudes of responses of inflation and output to interest rates are more responsive in the low-inflation regime (left panel) than in the high-inflation regime (right panel).

```
# --- Code for nonlinear effects of the federal funds rate.

# Load packages for creating plots
  library(dplyr)
  library(gridExtra)
  library(ggpubr)

# Create dummy: apply threshold of 4.75 percent to the third lag of the inflation rate
  switching_data <-  if_else(dplyr::lag(endog_data$Infl, 3) > 4.75, 1, 0)

# Estimate nonlinear model
  results_nl <- lp_nl(endog_data,
                      lags_endog_lin = 4,              lags_endog_nl = 4,
                      trend          = 1,              shock_type    = 0,
                      confint        = 1.67,           hor           = 12,
                      switching      = switching_data, lag_switching = FALSE,
```

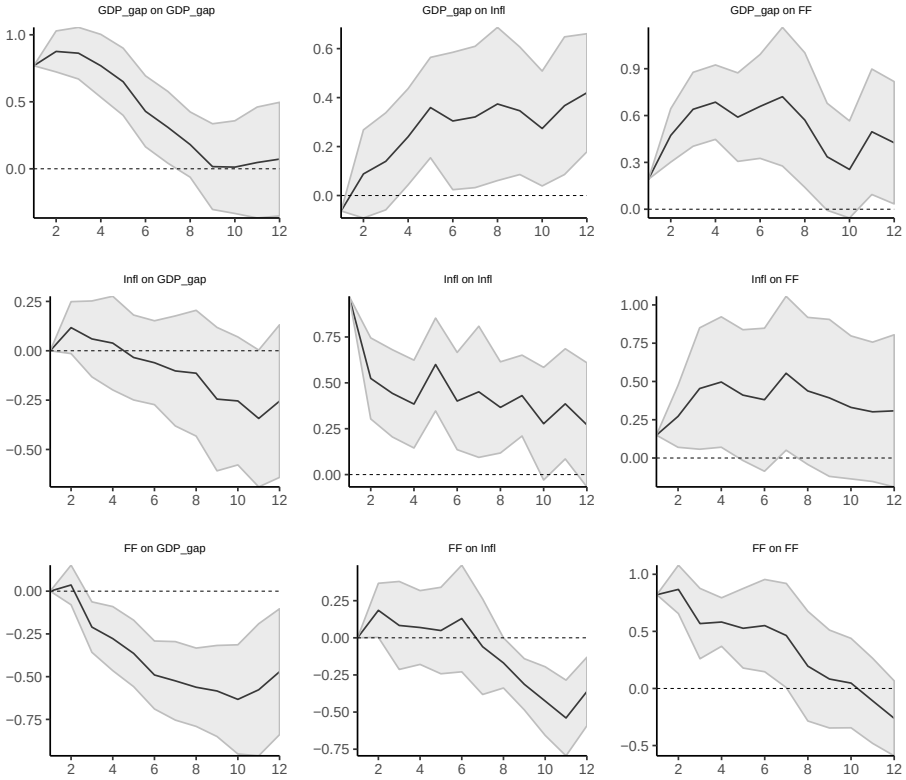


Figure 1: Replication of Figure 5 in Jordà (2005, p. 176).

```
                    use_logistic   = FALSE)

# Create nonlinear impulse responses
  nl_plots <- plot_nl(results_nl)

# Combine and show plots using 'ggpubr' and 'gridExtra'
  single_plots      <- nl_plots$gg_s1[c(3, 6, 9)]
  single_plots[4:6] <- nl_plots$gg_s2[c(3, 6, 9)]
  all_plots         <- sapply(single_plots, ggplotGrob)
  marrangeGrob(all_plots, nrow = 3, ncol = 2, top = NULL)

# --- End example code
```

Using an external shock: Replicating results by Ramey and Zubairy (2018)

In this section, I replicate the empirical results by Ramey and Zubairy (2018). The authors, among others, re-evaluate the findings by Auerbach and Gorodnichenko (2012), who argued that government spending multipliers are more pronounced during economic recession than during economic expansion. Auerbach and Gorodnichenko (2012) applied a smooth transition VAR (STVAR) to estimate state-dependent fiscal multipliers. Ramey and Zubairy (2018), however, showed that the estimated fiscal multipliers are much smaller when the impulse responses are estimated using LPs. The reason is that the LP approach does not assume that the system remains in a fixed regime once it has entered it.

The following code replicates parts of Figure 12 in the supplementary appendix by Ramey and Zubairy (2018, p. 35). The results are depicted in Figure 3. It shows how government spending and the GDP react to a government spending shock in the linear case as well as during periods of economic expansion and recession. The linear shock is identified according to Blanchard and Perotti (2002). The absolute values of the figures differ because Ramey and Zubairy (2018) multiplied the log output response by a conversion factor of 5.6.

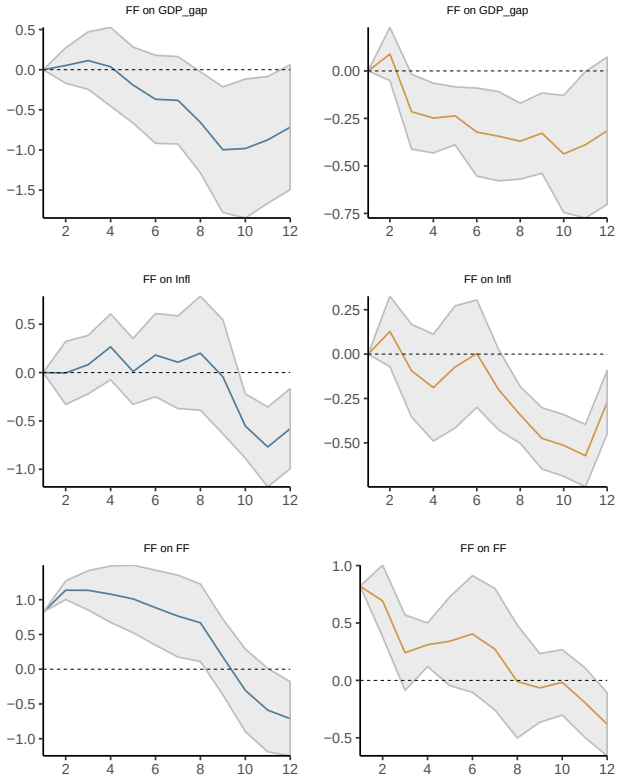


Figure 2: Nonlinear impulse responses based on Jordà (2005).

The figure depicts nonlinear impulse responses of the output gap, inflation rate, and federal funds rate to a shock in the federal funds rate during periods of low (left panel) and high (right panel) inflation rates. The threshold of 4.75 is applied to the third lag of the inflation rate.

```
# --- Code to replicate parts of Figure 12 in the supplementary appendix by
# --- Ramey and Zubairy (2018, p.35)

# Load packages for creating plots
  library(gridExtra)
  library(ggpubr)

# Load data from package
  ag_data      <- ag_data
  sample_start <- 7
  sample_end   <- dim(ag_data)[1]

# Endogenous data
  endog_data <- ag_data[sample_start:sample_end,3:5]

# Shock variable
  shock      <- ag_data[sample_start:sample_end, 3]

# Estimate linear model
  results_lin_iv <- lp_lin_iv(endog_data = endog_data, lags_endog_lin = 4,
                              shock      = shock,      trend          = 0,
                              confint    = 1.96,       hor            = 20)

# Make and save linear plots
  iv_lin_plots <- plot_lin(results_lin_iv)

# Nonlinear shock (estimated by Ramey and Zubairy (2018))
  shock <- ag_data[sample_start:sample_end, 7]
```

```
# Use moving average growth rate of GDP as exogenous variable
  exog_data <- ag_data[sample_start:sample_end, 6]

# Use moving average growth rate of GDP as switching variable
  switching_variable <- ag_data$GDP_MA[sample_start:sample_end] - 0.8

# Estimate nonlinear model
  results_nl_iv <- lp_nl_iv(endog_data = endog_data, lags_endog_nl  = 3,
                            shock      = shock,      exog_data      = exog_data,
                            lags_exog  = 4,          trend          = 0,
                            confint    = 1.96,       hor            = 20,
                            switching  = switching_variable, use_hp = FALSE,
                            gamma      = 3)

# Make and save nonlinear plots
  plots_nl_iv <- plot_nl(results_nl_iv)

# Make list to save all plots
  combine_plots <- list()

# Save linear plots in list
  combine_plots[[1]] <- iv_lin_plots[[1]]
  combine_plots[[2]] <- iv_lin_plots[[3]]

# Save nonlinear plots for expansion period
  combine_plots[[3]] <- plots_nl_iv$gg_s1[[1]]
  combine_plots[[4]] <- plots_nl_iv$gg_s1[[3]]

# Save nonlinear plots for recession period
  combine_plots[[5]] <- plots_nl_iv$gg_s2[[1]]
  combine_plots[[6]] <- plots_nl_iv$gg_s2[[3]]

# Show all plots
  lin_plots_all      <- sapply(combine_plots, ggplotGrob)
  marrangeGrob(lin_plots_all, nrow = 2, ncol = 3, top = NULL)

# --- End example code
```

Estimating impulse responses for panel data

Using the *Jordà-Schularick-Taylor Macrohistory Database*, the following example estimates the impulse responses of the ratio of mortgage lending divided by the GDP to a 1% increase in the short-term interest rate. Observations during World Wars I and II and observations after 2013 are excluded.[4] The empirical results are shown in Figure 4. An increase in the short-term interest rate leads to a decrease in the mortgage lending rate, whose effect attenuates after approximately 8 years.

```
#--- Begin code for panel data

# Load libraries to download and read excel file from the website
  library(httr)
  library(readxl)
  library(dplyr)

# Retrieve the external JST Macrohistory Database
  url_jst <-"http://www.macrohistory.net/JST/JSTdatasetR3.xlsx"
  GET(url_jst, write_disk(jst_link <- tempfile(fileext = ".xlsx")))
  jst_data <- read_excel(jst_link, 2L)
```

[4] **lpirfs** first computes all lags and lags of the first differences of the exogenous data. If the user wants to use a sub-sample (see example), the observations will be dropped after the lags have been constructed.

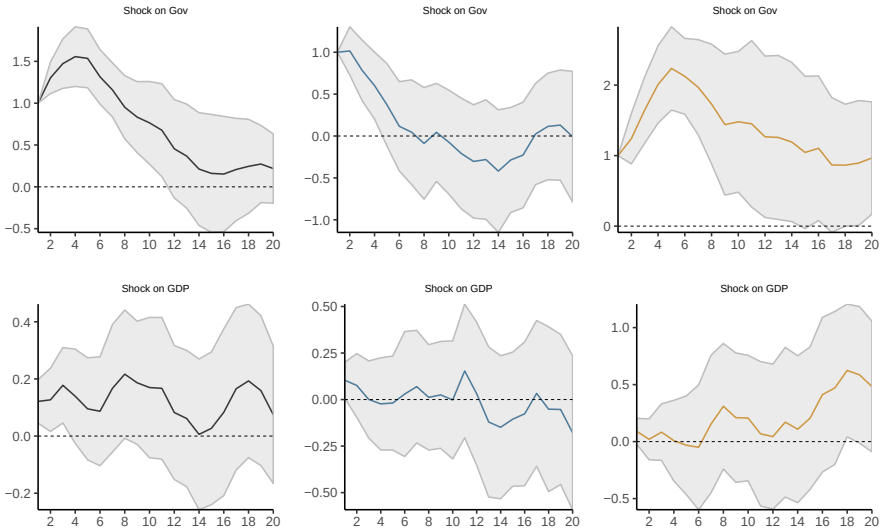


Figure 3: The figure replicates empirical results of Figure 12 from the supplementary appendix by Ramey and Zubairy (2018, p. 35). The left column shows the (linear) reaction of government spending (first row) and GDP (second row) to a government spending shock. The middle column shows the reactions during periods of economic expansion and the right column during periods of economic slack.

```
# Remove observations after 2013 and swap the first two columns
  jst_data <- jst_data                          %>%
              dplyr::filter(year <= 2013)   %>%
              dplyr::select(country, year, everything())

# Prepare variables
  data_set <- jst_data                                        %>%
              mutate(stir    = stir)                          %>%
              mutate(mortgdp = 100*(tmort/gdp))               %>%
              mutate(hpreal  = hpnom/cpi)                     %>%
              group_by(country)                               %>%
              mutate(hpreal  = hpreal/hpreal[year==1990][1]) %>%
              mutate(lhpreal = log(hpreal))                   %>%

              mutate(lhpy    = lhpreal - log(rgdppc))         %>%
              mutate(lhpy    = lhpy - lhpy[year == 1990][1]) %>%
              mutate(lhpreal = 100*lhpreal)                   %>%
              mutate(lhpy    = 100*lhpy)                      %>%
              ungroup()                                       %>%

              mutate(lrgdp   = 100*log(rgdppc))               %>%
              mutate(lcpi    = 100*log(cpi))                  %>%
              mutate(lriy    = 100*log(iy*rgdppc))            %>%
              mutate(cay     = 100*(ca/gdp))                  %>%
              mutate(tnmort  = tloans - tmort)                %>%
              mutate(nmortgdp = 100*(tnmort/gdp))             %>%
              dplyr::select(country, year, mortgdp, stir, ltrate,
              lhpy, lrgdp, lcpi, lriy, cay, nmortgdp)


# Exclude observations from WWI and WWII
  data_sample <- seq(1870, 2016)[which(!(seq(1870, 2016) %in%
                                     c(seq(1914, 1918),
                                       seq(1939, 1947))))]

# Estimate linear panel model with robust covariance matrix
  results_panel <- lp_lin_panel(data_set  = data_set,  data_sample  = data_sample,
                        endog_data        = "mortgdp", cumul_mult   = TRUE,
```

```
                        shock             = "stir",    diff_shock   = TRUE,
                        panel_model       = "within",  panel_effect = "individual",
                        robust_cov        = "vcovSCC", c_exog_data  = "cay",
                        c_fd_exog_data    = colnames(data_set)[c(seq(4,9),11)],
                        l_fd_exog_data    = colnames(data_set)[c(seq(3,9),11)],
                        lags_fd_exog_data = 2,      confint      = 1.67,
                        hor               = 10)

# Show irfs
  plot(results_panel)

#--- End example
```

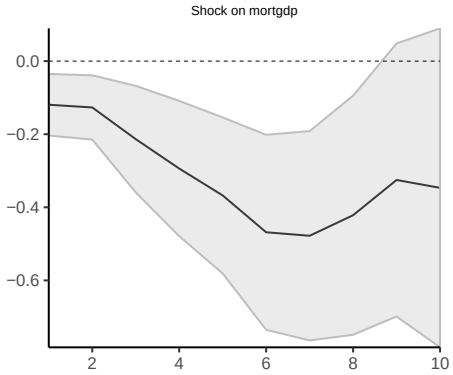


Figure 4: The figure shows the reaction of the ratio of mortgage lending divided by the GDP to a +1% change in the short-term interest rate.

The following example uses the Hodrick–Prescott filter to decompose the log-GDP time series for each country to obtain the standardized variable z_t for the logistic function in Eq. (2). Figure 5 shows the impulse responses for both regimes. The mortgage lending ratio declines in both regimes, although it is more pronounced during periods of economic expansion (left panel) than periods of economic slack (right panel).

```
# --- Begin example

# Estimate panel model
  results_panel <- lp_nl_panel(data_set   = data_set,  data_sample    = data_sample,
                        endog_data        = "mortgdp", cumul_mult     = TRUE,
                        shock             = "stir",    diff_shock     = TRUE,
                        panel_model       = "within",  panel_effect   = "individual",
                        robust_cov        = "vcovSCC", switching      = "lrgdp",
                        lag_switching     = TRUE,      use_hp         = TRUE,
                        lambda            = 6.25,      gamma          = 10,
                        c_exog_data       = "cay",
                        c_fd_exog_data    = colnames(data_set)[c(seq(4,9),11)],
                        l_fd_exog_data    = colnames(data_set)[c(seq(3,9),11)],
                        lags_fd_exog_data = 2,
                        confint           = 1.67,
                        hor               = 10)

# Show irfs
  plot(results_panel)


# --- End example
```

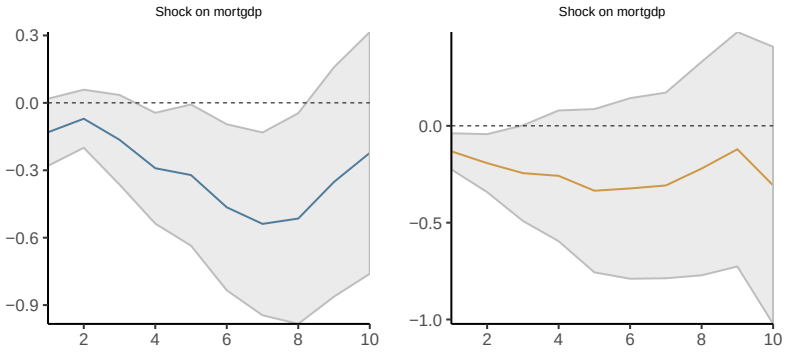


Figure 5: The figure shows the reaction of the ratio of mortgage lending divided by the GDP to a +1% change in the short-term interest rate during periods of economic expansions (left panel) and economic slack (right panel).

Summary

Since the 1980s, impulse response analysis has become a cornerstone in (macro-)econometrics. The traditional approach of recovering the impulse responses recursively from a (linear) VAR has been criticized due to some drawbacks, such as the imposed dynamics on the (economic) system, the curse of dimensionality, and the more difficult application to nonlinear frameworks.

The LPs by Jordà (2005) have become a widely applied alternative to estimate impulse response functions. This paper introduced **lpirfs**, an R package that provides a broad framework for estimating and visualizing impulse response functions using LPs for a variety of data sets. I replicated the empirical results from the economic literature to prove the validity of the package and to show its usefulness for future research.

Appendix

This appendix contains a comparison between impulse responses estimated using the **vars** package and **lpirfs**. In addition, I conduct sensitivity analyses regarding the choices of γ in Eq. (2) for the switching function and different values of λ for the filter by Hodrick and Prescott (1997). The code for all examples can be found in the `vignette` of **lpirfs**.

Comparison of impulse responses between lpirfs and vars

Plagborg-Møller and Wolf (2019) showed that LPs and VARs compute the same impulse responses when the lag structure is unrestricted. For empirical studies, this implies that impulse responses that are estimated by LPs and SVARs are likely to agree at short horizons but differ at longer ones. To verify this implication, I compare the impulse responses estimated using the **lpirfs** and **vars** packages. The latter relies on the common SVAR approach.

Figure 6 shows the empirical results. The black lines and gray-shaded areas are the same as in Figure 1, which replicates Figure 5 in Jordà (2005, p. 176). The orange lines and shaded areas correspond to impulse responses estimated by the **vars** package (i.e., a standard SVAR). The results show that impulse responses are similar up to that horizon, which equals the lag length p. For example, when the lag length p equals 2, the impulse responses and confidence intervals diverge very quickly (first column). When the lag length p equals 6, however, the impulse responses are much more similar. This finding coincides with the empirical results by Plagborg-Møller and Wolf (2019), who compared the dynamic response of corporate bond spreads to a monetary policy shock.

Sensitivity analyses for γ

The nonlinear functions in **lpirfs** allow separating the data into two regimes by either using a dummy approach or computing state probabilities with the logistic function given in Eq. (2). The logistic function depends on the parameter γ, which defines how sharply the two regimes are separated. To investigate how different choices of γ might affect the results, I compare the nonlinear impulse responses for a shock of the federal funds rate on the output gap.

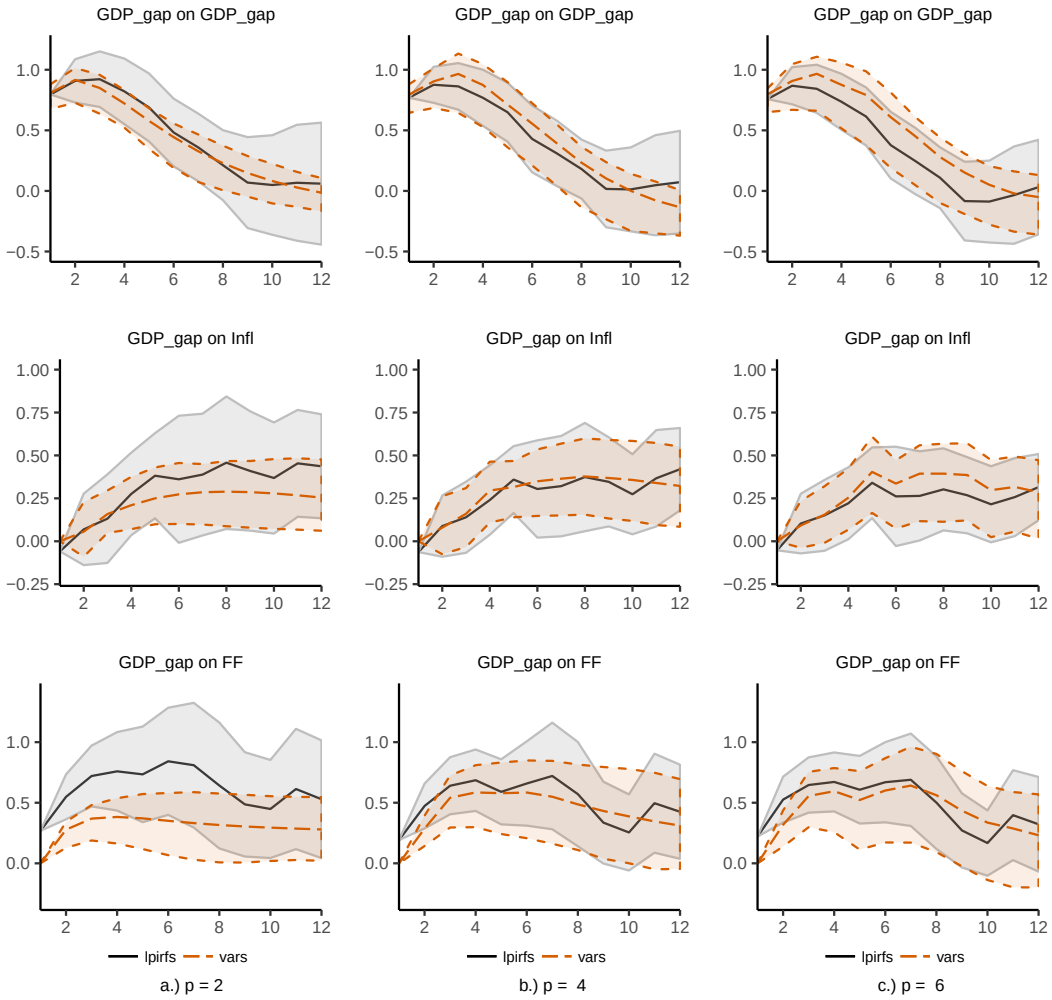


Figure 6: Comparison of impulse responses estimated by the packages **lpirfs** and **vars**. Each column shows the results for a fixed number of lags (i.e., $p = 2$, 4, and 6). The shaded areas correspond to the 95% confidence intervals.

Figure 7 shows the empirical results. Each column corresponds to one choice of γ, namely $\gamma = 1$, 5, and 10. I use the output gap as a switching variable and decompose it using the filter by Hodrick and Prescott (1997). The penalty term λ is set to 1 600 as suggested by Ravn and Uhlig (2002). The first row of Figure 7 shows the evolution of the transition variable $F(z_t)$, along with NBER-dated recessions. By construction, a high value of the transition variable corresponds to a low output gap (i.e., periods of economic slack). Choosing a low value of γ makes the regime-switching smooth, whereas higher values of γ cause the switching to be quick. The second and third rows show nonlinear impulse responses for Regimes 1 (economic expansion) and 2 (economic recession). Although the choice of γ has an effect on the results, it does not change the overall conclusion, namely that no "business-cycle" effects exist regarding the changes in the federal funds rate, which is in accordance with the findings by Jordà (2005). At most, the effect would be very shortly negative during periods of economic downturn.

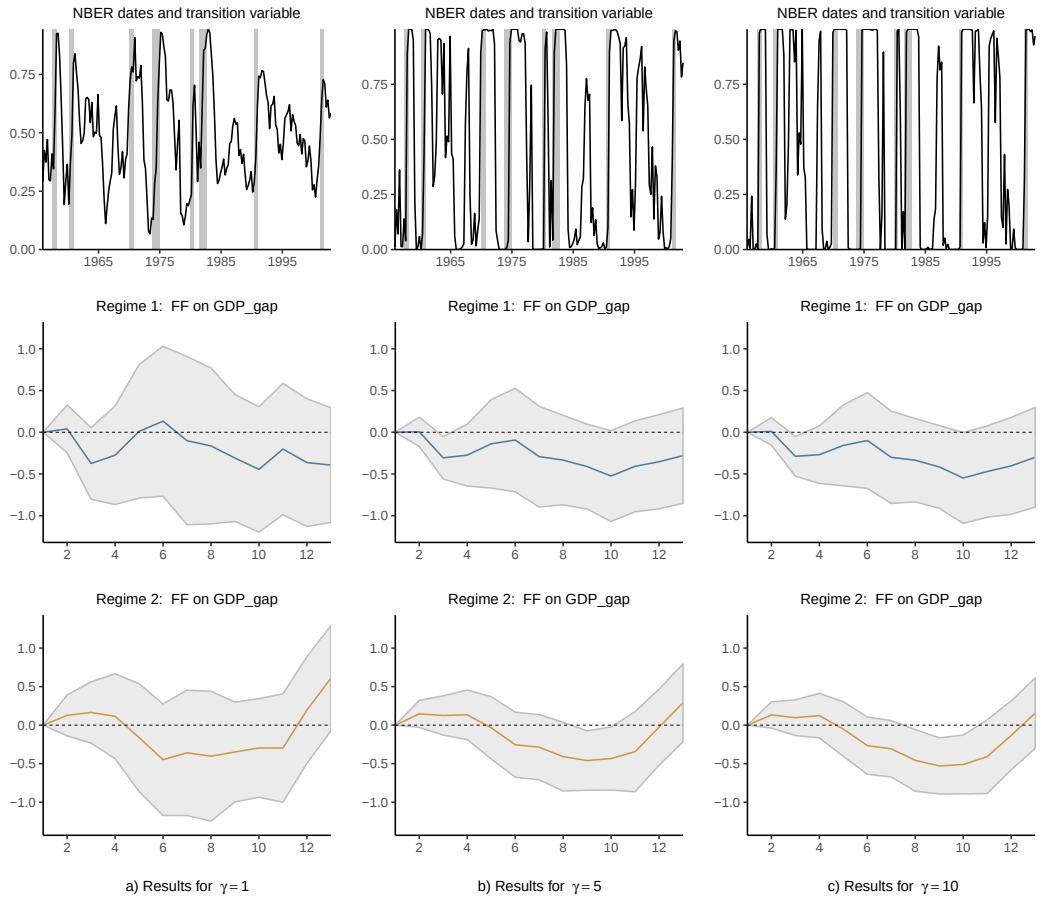


Figure 7: Comparison of nonlinear impulse responses for different values of γ. Each column shows the results for one parameter value (i.e., $\gamma = 1, 5$, and 10). The gray shaded areas in the first row correspond to NBER-dated recessions. The gray shaded areas in the second and third rows correspond to the 95% confidence intervals of the impulse responses.

Sensitivity analyses for λ

To use the switching function in Eq. (2), one must provide a standardized variable z_t. One option is to decompose a time series into a trend and a cyclical component using the filter by Hodrick and Prescott (1997) (hereafter the HP-filter). **lpirfs** includes this option whose routine is written in **Rcpp**, making the computation very fast. If applied, the cyclical component of the HP-filter will be standardized and used for z_t. The filter depends on a penalty term λ, which must be given by the user. Ravn and Uhlig (2002) argued that the parameter should be 6.25 for annual data, 1 600 for quarterly data, and 129 600 for monthly data. To see how different choices of λ influence the nonlinear impulse responses, I decompose the output gap for three different values of λ and compare the results, which are shown in Figure 8. The value of γ is fixed to 5. The first row shows the cyclical component of the HP-filter along with the NBER-dated recessions. A low value in the cyclical component denotes periods of economic downturn. Note that the results of the second column in Figure 8 are identical to those

in the second column of Figure 7. In contrast to the previous analysis, empirical results do change significantly, depending on the choice of λ. Thus, it is important to set the penalty term adequately.

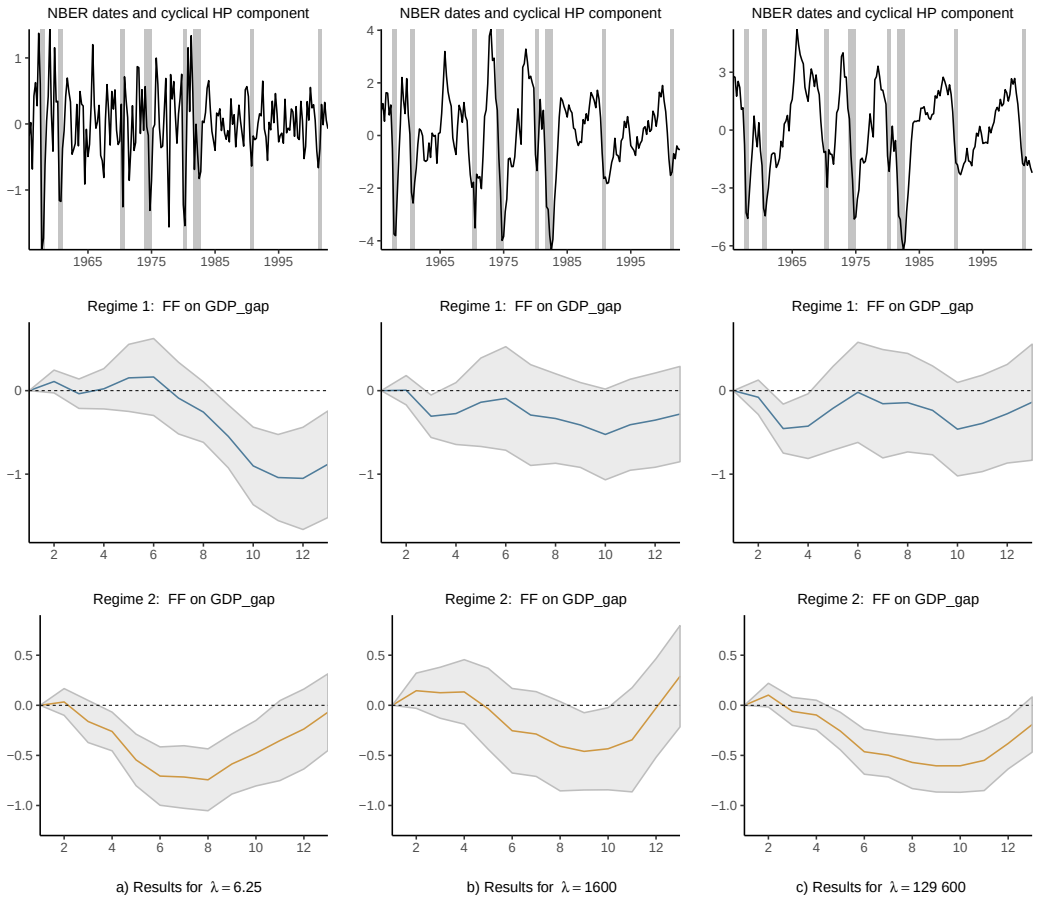


Figure 8: Comparison of nonlinear impulse responses with different values of λ for the filter by Hodrick and Prescott (1997). Each column shows the results for one parameter value (i.e., $\lambda = 6.25, 1\,600$, and $129\,600$). The gray shaded areas in the first row correspond to NBER-dated recessions. The gray shaded areas in the second and third rows correspond to the 95% confidence intervals of the impulse responses.

Acknowledgement

I am grateful to the two anonymous reviewers, Philipp Wittenberg, Jon Danielsson, Rainer Schüssler, Tom Philipp Dybowski, and Detlef Steuer for their helpful comments and suggestions on the paper and package.

Bibliography

P. Adämmer. *lpirfs: Local Projections Impulse Response Functions*, 2019. URL https://CRAN.R-project.org/package=lpirfs. R package version: 0.1.6. [p421]

M. I. Ahmed and S. P. Cassou. Does consumer confidence affect durable goods spending during bad and good economic times equally? *Journal of Macroeconomics*, 50:86–97, 2016. doi: https://doi.org/10.1016/j.jmacro.2016.08.008. [p424]

H. Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19 (6):716–723, 1974. doi: https://doi.org/10.1109/TAC.1974.1100705. [p425]

D. W. K. Andrews and J. C. Monahan. An improved heteroskedasticity and autocorrelation consistent covariance matrix estimator. *Econometrica*, 60(4):953–966, 1992. doi: https://doi.org/10.2307/2951574. [p425]

A. J. Auerbach and Y. Gorodnichenko. Measuring the output responses to fiscal policy. *American Economic Journal: Economic Policy*, 4(2):1–27, 2012. doi: https://doi.org/10.1257/pol.4.2.1. [p421, 423, 428]

A. J. Auerbach and Y. Gorodnichenko. Output spillovers from fiscal policy. *American Economic Review*, 103(3):141–46, 2013. doi: https://doi.org/10.1257/aer.103.3.141. [p421, 423, 424]

R. Barnichon and C. Brownlees. Impulse response estimation by smooth local projections. *Review of Economics and Statistics*, 101(3):522–530, 2019. doi: https://doi.org/10.1162/rest_a_00778. [p421]

O. Blanchard and R. Perotti. An Empirical Characterization of the Dynamic Effects of Changes in Government Spending and Taxes on Output. *The Quarterly Journal of Economics*, 117(4):1329–1368, 2002. doi: https://doi.org/10.1162/003355302320935043. [p428]

L. Brugnolini. About local projection impulse response function reliability. *CEIS Working Paper*, 2018. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3229218. [p421, 422]

Y. Croissant and G. Millo. Panel data econometrics in R: The plm package. *Journal of Statistical Software*, 27(2):1–43, 2008. doi: https://doi.org/10.18637/jss.v027.i02. [p425]

D. Eddelbuettel, R. François, J. Allaire, K. Ushey, Q. Kou, N. Russel, J. Chambers, and D. Bates. Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: https://doi.org/10.18637/jss.v040.i08. [p425]

G. Favara and J. Imbs. Credit supply and the price of housing. *American Economic Review*, 105(3): 958–92, 2015. doi: https://doi.org/10.1257/aer.20121416. [p421]

J. Garín, R. Lester, and E. Sims. Are supply shocks contractionary at the zlb? evidence from utilization-adjusted tfp data. *Review of Economics and Statistics*, 101(1):160–175, 2019. doi: https://doi.org/10.1162/rest_a_00723. [p421]

J. D. Hamilton. Nonlinearities and the macroeconomic effects of oil prices. *Macroeconomic Dynamics*, 15(S3):364–378, 2011. doi: https://doi.org/10.1017/S1365100511000307. [p421]

R. J. Hodrick and E. C. Prescott. Postwar us business cycles: an empirical investigation. *Journal of Money, Credit, and Banking*, pages 1–16, 1997. doi: https://doi.org/10.2307/2953682. [p423, 425, 426, 433, 435, 436]

C. M. Hurvich and C.-L. Tsai. Regression and time series model selection in small samples. *Biometrika*, 76(2):297–307, 1989. doi: https://doi.org/10.1093/biomet/76.2.297. [p425]

Ò. Jordà. Estimation and inference of impulse responses by local projections. *American Economic Review*, 95(1):161–182, 2005. doi: https://doi.org/10.1257/0002828053828518. [p421, 423, 426, 427, 428, 429, 433, 435]

Ò. Jordà and A. M. Taylor. The time for austerity: estimating the average treatment effect of fiscal policy. *The Economic Journal*, 126(590):219–255, 2016. doi: https://doi.org/10.1111/ecoj.12332. [p421]

Ò. Jordà, M. Schularick, and A. M. Taylor. Betting the house. *Journal of International Economics*, 96: S2–S18, 2015. doi: https://doi.org/10.1016/j.jinteco.2014.12.011. [p421, 424]

Ò. Jordà, M. Schularick, and A. M. Taylor. The effects of quasi-random monetary experiments. *Journal of Monetary Economics*, In press, 2019. doi: https://doi.org/10.1016/j.jmoneco.2019.01.021. [p421, 424]

J. Keating. Structural approaches to vector autoregressions. *Federal Reserve Bank of St. Louis Review*, 74 (September/October), 1992. URL https://files.stlouisfed.org/files/htdocs/publications/review/92/09/Vector_Sep_Oct1992.pdf. [p422]

L. Kilian and Y. J. Kim. How reliable are local projection estimators of impulse responses? *Review of Economics and Statistics*, 93(4):1460–1466, 2011. doi: https://doi.org/10.1162/REST_a_00143. [p421, 422, 423]

W. K. Newey and K. D. West. Hypothesis testing with efficient method of moments estimation. *International Economic Review*, pages 777–787, 1987. doi: https://doi.org/10.2307/2526578. [p423, 424, 425]

M. T. Owyang, V. A. Ramey, and S. Zubairy. Are government spending multipliers greater during periods of slack? evidence from twentieth-century historical data. *American Economic Review*, 103(3): 129–134, 2013. doi: https://doi.org/10.1257/aer.103.3.129. [p421, 424]

M. A. Petersen. Estimating standard errors in finance panel data sets: Comparing approaches. *The Review of Financial Studies*, 22(1):435–480, 2009. doi: https://doi.org/10.1093/rfs/hhn053. [p424]

B. Pfaff. VAR, SVAR and SVEC models: Implementation Within R Package vars. *Journal of Statistical Software*, 27(4), 2008. doi: https://doi.org/10.18637/jss.v027.i04. [p421]

M. Plagborg-Møller and C. K. Wolf. Local projections and vars estimate the same impulse responses. *Unpublished paper: Department of Economics, Princeton University*, 2019. URL https://scholar.princeton.edu/sites/default/files/mikkelpm/files/lp_var.pdf. [p421, 433]

V. A. Ramey and S. Zubairy. Government spending multipliers in good times and in bad: evidence from us historical data. *Journal of Political Economy*, 126(2):850–901, 2018. doi: https://doi.org/10.1086/696277. [p424, 426, 428, 431]

M. O. Ravn and H. Uhlig. On adjusting the hodrick-prescott filter for the frequency of observations. *Review of Economics and Statistics*, 84(2):371–376, 2002. doi: https://doi.org/10.1162/003465302317411604. [p425, 426, 435]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. doi: https://doi.org/10.1214/aos/1176344136. [p425]

C. A. Sims. Macroeconomics and reality. *Econometrica: Journal of the Econometric Society*, pages 1–48, 1980. doi: https://doi.org/10.2307/1912017. [p421]

E. T. Swanson. Measuring the effects of federal reserve forward guidance and asset purchases on financial markets. Technical report, National Bureau of Economic Research, 2017. URL https://www.nber.org/papers/w23311. [p421]

S. Tenreyro and G. Thwaites. Pushing on a string: Us monetary policy is less powerful in recessions. *American Economic Journal: Macroeconomics*, 8(4):43–74, 2016. doi: https://doi.org/10.1257/mac.20150016. [p421]

Philipp Adämmer
Department of Mathematics and Statistics
Helmut Schmidt University
Hamburg, Germany
(ORCiD: 0000-0003-3770-0097)
adaemmer@hsu-hh.de

Conference Report: ConectaR 2019

by Marcela Alfaro Córdoba, Frans van Dunné, Agustín Gómez Meléndez, and Jacob van Etten

About the event

ConectaR 2019: Encuentro de Usuarios R en Latinoamérica, took place during January 24-26, 2019 at the University of Costa Rica, in San José, Costa Rica. It was the first event in Central America endorsed by The R Foundation, and it was held completely in Spanish. The majority of the attendants were from Costa Rica (85%), but we had participants from 12 countries: Costa Rica, Guatemala, Peru, Colombia, Mexico, Argentina, Uruguay, Chile, Spain, the Netherlands, France and the USA. The three-day event consisted of talks, workshops, and poster sessions.

The primary purpose of ConectaR conference was to provide a space to create a community among R users in industry, academia, citizen science and teaching. In this way, we aim to encourage the use of R, promote learning and advance the development of R packages adapted to our regional needs.

ConectaR 2019 was organized by the University of Costa Rica -through the School of Statistics, the Development Observatory, and the Research Center for Pure and Applied Mathematics- the company ixpantia, and the research institution Bioversity International. The initiative originated thanks to the encouragement of Heather Turner, who contacted several networks in the region, through the R Users Groups, R-ladies groups and other connections.

From the 150 registered participants, 33% were female and 23% were full time students. Professionals from finance, government and data companies were present, as well as faculty members from all four major universities in the country. The event was possible thanks to the effort of a team of about 50 people including 4 chairs, a 23-member scientific committee and a motivated group of 23 volunteers.

Figure 1: The logo of ConectaR.

Conference program

The first two days of the conference were dedicated to talks (invited and contributed) and poster presentations. On the third day of the event (a Saturday) four workshops ran in parallel: two during the morning and two during the afternoon.

The event had four invited talks: two that were in person and two via video conference. Edgar Ruiz from RStudio, was the first keynote. He gave a remarkably clear explanation about how to use R and Spark for Data Science. During the afternoon, Maëlle Salmon from rOpenSci and Locke Data, presented the second keynote (remote), where she talked about the ROpenSci initiative (https://ropensci.org/), and about her experience curating R packages. She gave the audience tips on how to write R packages, a clear explanation on the importance of citing, curating and recognizing R packages as part of the scientific process.

During the second day of the event, Robert Hijmans from UC Davis, explained the use of

Figure 2: Picture from the event.

R for spatial data science, and talked about his experience using R for scientific production and teaching, including the creation of new packages. His talk ended with an invitation to translate the material from his web page into Spanish: https://rspatial.org/. To close the last day of talks, Antonio Vasquez Brust from Buenos Aires University (UBA), Argentina gave a detailed description on how to use R and Open Data to understand our cities. His discussion encouraged good practices in visualization as well as a conversation about city planning using R when Open Data is available.

A panel named "Connecting data innovation initiatives in Latin America with R" was facilitated by Diego May (ixpantia) during the second day of the event. The intention of the panel was to have professionals talk about the opportunities and challenges around the use of R in their different work contexts. Alexia Pacheco of ICE, the largest utility company in Costa Rica, explained how data science and R has pervaded their work since its origin. Jacob van Etten (Bioversity International) explained how R is used in a multi-country team in an international agricultural research institute. It has provided important opportunities for quick methodological innovation to support a large citizen science initiative. Alvaro Pabón of Finsocial Colombia, explained how he has set up a data science team in a Colombian company, the challenges to build this capacity and the support needed for it.

Eleven contributed talks and fourteen posters were presented during the event. The selection process had two stages: during the first one, the reviewers gave recommendations to the authors on how to improve their abstract, and during the second stage, the talks that had a satisfactory level were accepted. The posters were then reviewed by the chairs to ensure all of them had a satisfactory level. Two out of the eleven contributed talks and six out of the fourteen posters were presented by women.

The topics of the contributed talks followed the four themes of the conference. First government and citizen science, where we saw how shiny apps are used at the Costa Rican national comptroller's office. We also heard how the national statistics office is transitioning from SPSS to R. In the industry track a talk about the transition from Excel to R at the national insurance institute showed how this has lead to significant reduction in time spent on data processing. In academia the visualization and analysis of complex climate data took center stage in two separate talks. The teaching track included a fun example of how to predict the outcome of soccer matches, and showcased experiences from Mexico of the power of R as a didactic tool in statistics and mathematics.

After the last break of the first day, the poster session was opened and accompanied by the conference cocktail reception. As organizers we felt strongly about including sufficient opportunities for people to mingle and talk. The posters were well visited and led to spirited discussions. The conference dinner had a lower attendance than the reception, but served its purpose just as well in offering an opportunity for people in the community to connect and re-connect.

During the last day of the event four workshops were held, each of which managed to

attract full classrooms:

- Crear API's con código R usando plumber by Frans van Dunné.
- Documentos dinámicos, trabajo colaborativo y control de versiones con Rmarkdown y GitHub by Natalia da Silva.
- Análisis de texto by Riva Quiroga.
- Introducción al análisis Bayesiano con aplicaciones en STAN by Ignacio Álvarez-Castro.

ConectaR served to connect different communities, announce exciting projects and to create new ones. Examples are the visit and help of two of the three chairs of LatinR to teach workshops and their LatinR2019 announcement during the closing remarks of ConectaR. Also, Riva Quiroga explained details about the R4DS translation project (https://es.r4ds.hadley.nz) to the community, and Frans van Dunné asked for volunteers to start the (already advanced) Plumber translation project (https://github.com/fontanero-api/).

Communities outside R were also involved, such as Women in Engineering, who organized an introduction to R workshop for its members two months after the event, with the help of Marcela Alfaro Córdoba. DataLatam did several interviews for their podcast thanks to the connections established during ConectaR, and the School of Statistics made the first arrangements to invite Edgar Ruiz to give a week of workshops to its students and faculty in June.

Evaluation

An evaluation questionnaire was circulated after the event, and 70% of the participants filled it out. The results were overwhelmingly positive, having a median rate of 5 (on a scale from 1=bad job to 5=good job) for all the questions, with very small variability in each distribution. A word cloud of the comment section of the questionnaire was constructed and is shown in Figure 3, where it is clear that positive words such as excellent (excelente), good (bien, bueno) and quality (calidad) were among the most used in the comments.

Figure 3: Word cloud for the evaluation comments.

Corporate Sponsors

ConectaR was possible thanks to the main sponsors: INCAE Business School, R Consortium, RStudio, Inc., Hivos Latinoamerica, The Trust for the Americas. Also, a small job fair was organized parallel to the conference, in which the some of the main sponsors participated, along with companies like McKinsey & Company, Alteryx, Growth Acceleration and Partners, ThermoFisher Scientific and ixpantia.

Other Events and Future Steps

Future plans for the chairs of ConectaR include ConectaR 2021, where the expectation is to improve the network with communities in Mexico, Colombia, Panamá, and search for funding sources to overcome our most important limitation for this edition: lack of funding to cover travel expenses. Also, the organization of more local events such as a Datathon for 2020, is on the agenda. The idea is to gather momentum to get different R communities from the region to participate in a visualization competition, inspired in Open Data from the Costa Rican Government.

Further information

- ConectaR materials: https://github.com/ConectaR2019.
- Twitter account: @conecta_R, #ConectaR2019.
- Webpage: http://www.conectar2019.ucr.ac.cr
- Facebook account: conectar2019

Marcela Alfaro Córdoba
Escuela de Estadística, Facultad de Ciencias Económicas
Universidad de Costa Rica
Ciudad Universitaria Rodrigo Facio
Costa Rica
ORCiD: 0000-0002-7703-3578
marcela.alfarocordoba@ucr.ac.cr

Agustín Gómez Meléndez
Observatorio de Desarrollo
Escuela de Estadística, Facultad de Ciencias Económicas
Universidad de Costa Rica
Costa Rica
agustin.gomez@ucr.ac.cr

Frans van Dunné
ixpantia
Costa Rica
frans@ixpantia.com

Jacob van Etten
Bioversity International
Spain
j.vanetten@cgiar.org

R Foundation News

by Torsten Hothorn

Donations and members

Membership fees and donations received between 2019-09-05 and 2020-02-24.

Donations

Amy Tzu-Yu Chen (United States) Murat D (France) Charles Geyer (United States) Susan Gruber (United States) Francesco Maria Lo Russo (Italy) Søren Lophaven (Denmark) J+Brian Loria (United States) Heramb Modak (India) Nikola Motik (Croatia) Kem Phillips (United States) Nick Redell (United States) Ravinderpal Vaid (United States) Dr. Alfred Wagner (Germany) Merck Research Laboratories, Kenilwort (United States)

Supporting benefactors

Thomas Levine (United States) b-data GmbH, Winterthur (Switzerland)

Supporting institutions

Fumihiko Makiyama (Japan) Code Ocean, Jenkintown (United States)

Supporting members

Antoniade Ciprian Alexandru (Romania) Tim Appelhans (Germany) Srinivas B (India) Michael Blanks (United States) Gordon Blunt (United Kingdom) Shannon Callan (United States) Gilberto Camara (Brazil) Susan M Carlson (United States) Michael Chirico (United States) Gerard Conaghan (United Kingdom) Terry Cox (United States) Robin Crockett (United Kingdom) Robert Daly (Australia) Gergely Daroczi (Hungary) Steph de Silva (Australia) Ajit de Silva (United States) Elliott Deal (United States) Jasja Dekker (Netherlands) Michael Dorman (Israel) Johan Eklund (Sweden) Shalese Fitzgerald (United States) Neil Frazer (United States) Keita Fukasawa (Japan) Laura Gabrysiak (United States) J. Antonio García (Mexico) Brian Gramberg (Netherlands) Krushi Gurudu (United States) Hlynur Hallgrímsson (Iceland) Joe Harwood (United Kingdom) Bela Hausmann (Austria) Arnošt Komárek (Czechia) Miha Kosmac (United Kingdom) Jan Herman Kuiper (United Kingdom) Hoonjeong Kwon (Republic of Korea) Mauro Lepore (United States) Chin Soon Lim (Singapore) Daniel McNichol (United States) Tore Christian Michaelsen (Norway) Guido Möser (Germany) Yoshinobu Nakahashi (Japan) Dan Orsholits (Switzerland) George Ostrouchov (United States) Antonio Paez (Canada) Peter Perez (United States) Elgin Perry (United States) Fergus Reig Gracia (Spain) Ingo Ruczinski (United States) Antonio J. Saez-Castillo (Spain) Pieta Schofield (United Kingdom) Jagat Sheth (United States) Rachel Smith-Hunter (United States) Berthold Stegemann (Germany) Tobias Strapatsas (Germany) Robert Szabo (Sweden) Waldemar T Talen (United States) Koray Tascilar (Germany) Michael Tiefelsdorf (United States) Uku Vainik (Estonia) Marcus Vollmer (Germany) Jaap Walhout (Netherlands) Sandra Ware (Australia)

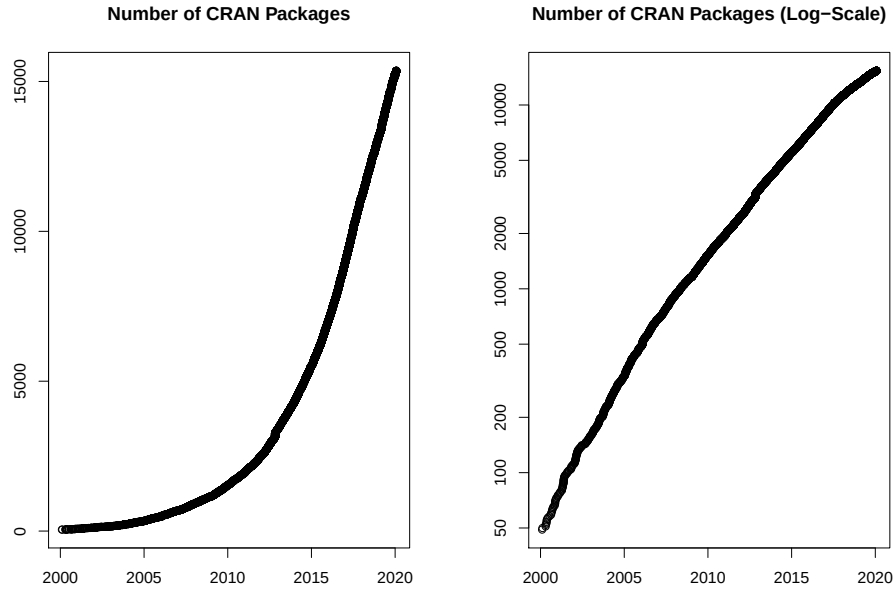
Torsten Hothorn
Universität Zürich, Switzerland Torsten.Hothorn@R-project.org

Changes on CRAN

2019-09-01 to 2019-12-31

by Kurt Hornik, Uwe Ligges and Achim Zeileis

In the past 4 months, 632 new packages were added to the CRAN package repository. 27 packages were unarchived and 182 were archived. The following shows the growth of the number of active packages in the CRAN package repository:



On 2019-12-31, the number of active packages was around 15227.

Changes in the CRAN Repository Policy

The Checklist for CRAN submissions now says the following:

- Make the Description as informative as possible for potential new users of your package. If in doubt, make the Description longer rather than shorter, but try to avoid redundancies such as repetitions of the package name.
- Write function names including parentheses as in `foo()` but without quotes.

CRAN package submissions

CRAN mirror security

Currently, there are 97 official CRAN mirrors, 67 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

Tracking Topic: Processing and Analysis of Tracking Data. Maintainer: Rocío Joo, Matthew E. Boone, Michael Sumner and Mathieu Basille. Packages: **BBMM**, **BayesianAnimalTracker**, **EMbC**, **FLightR**, **GGIR**, **GeoLight**, **PhysicalActivity**, **SDLfilter**, **SiM-**

Riv, **SimilarityMeasures**, **TrackReconstruction**, **TrajDataMining**, **VTrack**, **acc**, **accelerometry**, **adehabitatHR**, **adehabitatLT***, **amt**, **animalTrack**, **anipaths**, **argosfilter**, **bcpa**, **bsam**, **caribou**, **crawl**, **ctmcmove**, **ctmm**, **diveMove**, **foieGras**, **m2b**, **marcher**, **mkde**, **momentuHMM**, **move***, **moveHMM**, **moveVis**, **moveWindSpeed**, **nparACT**, **pawacc**, **recurse**, **rpostgisLT**, **rsMove**, **segclust2d**, **smam**, **spatsoc**, **trackdem**, **trackeR**, **trajectories**, **trajr**, **trip**, **tripEstimation**, **wildlifeDI**.

(* = core package)

New packages in CRAN task views

ChemPhys **RadData**, **radsafer**.

Distributions **MomTrunc**, **OwenQ**, **TruncatedNormal**, **distr6**, **distributions3**, **parameters**, **spam**, **truncdist**.

Econometrics **durmod**, **lpirfs**.

HighPerformanceComputing **RxODE**.

Hydrology **FedData**, **VICmodel**, **baseflow**, **echor**, **nasapower**, **openair**.

MachineLearning **mlr3**.

MetaAnalysis **BayesCombo**, **CopulaDTA**, **EValue**, **GENMETA**, **GMCM**, **HSROC**, **KenSyn**, **MBNMAdose**, **MBNMAtime**, **NMAoutlier**, **PRISMAstatement**, **PublicationBias**, **RBesT**, **RcmdrPlugin.MA**, **SingleCaseES**, **baggr**, **catmap**, **effectsize**, **harmonicmeanp**, **jarbes**, **mc.heterogeneity**, **metaBLUE**, **metacart**, **metapro**, **mixmeta**, **nmadb**.

MissingData **MatchThem**.

NumericalMathematics **HypergeoMat**, **SQUAREM**, **calculus**, **commonsMath**, **daarem**, **freealg**, **jack**, **kubik**, **matlib**, **mbend**, **turboEM**, **wedge**.

OfficialStatistics **MatchThem**, **RJDemetra**, **diyar**, **simPop**, **tidyqwi**.

Optimization **GPareto**, **Jaya**, **OOR**, **SCOR**, **nonneg.cg**, **roptim**.

Pharmacokinetics **RxODE**, **nlmixr**.

Psychometrics **conquestr**, **jrt**, **psychonetrics**.

ReproducibleResearch **DataPackageR**, **ProjectTemplate**, **RSuite**, **RepoGenerator**, **adapr**, **cabinets**, **drake**, **exreport**, **here**, **madrat**, **makeProject**, **orderly**, **prodigenr**, **projects**, **renv**, **repo**, **reports**, **represtools**, **storr**, **tinyProject**, **usethis**, **workflowr**, **zoon**.

SpatioTemporal **AtmRay**, **EMbC**, **SDLfilter**, **SiMRiv**, **SpaTimeClus**, **amt**, **anipaths**, **bsam**, **caribou**, **eyelinker**, **eyetracking**, **eyetrackingR**, **foieGras**, **gazepath**, **marcher**, **mdftracks**, **momentuHMM**, **mousetrack**, **mousetrap**, **moveVis**, **moveWindSpeed**, **movecost**, **oce**, **opentraj**, **psyosphere**, **rerddapXtracto**, **riverdist**, **rpostgisLT**, **rsMove**, **saccades**, **spatsoc**, **stampr**, **stplanr**, **trackdem**, **trackdf**, **trackeRapp**, **trajectories**, **trajr**.

TeachingStatistics **BetaBit**, **DALEX**, **HH**, **car**, **carData**, **effects**, **regtools**, **resampledata**.

TimeSeries **VARshrink**, **fable***, **feasts***, **forecastML**, **fpp3**, **nsarfima**, **sazedR**.

gR **bnclassify**, **mgm**, **qgraph**, **sna**.

(* = 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 3.10 was released on 30 October, 2019. It is compatible with R 3.6.1 and consists of 1823 software packages, 384 experiment data packages, 953 up-to-date annotation packages, and 27 workflows. The release announcement includes descriptions of 94 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", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()
```

Install additional packages and dependencies, e.g., **SingleCellExperiment**, with

```
BiocManager::install("SingleCellExperiment")
```

Docker and Amazon images provides a very 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 list of available software, linking to pages describing each package.
- A question-and-answer style user support site and developer-oriented mailing list.
- A community slack (sign up) for extended technical discussion.
- The F1000Research Bioconductor channel for peer-reviewed Bioconductor work flows.
- Our package submission repository for open technical review of new packages.

Our annual conference will be on July 29 - 31, 2020 in Boston, USA.

Bioconductor Core Team
Biostatistics and Bioinformatics
Roswell Park Cancer Institute, Buffalo, NY
USA maintainer@bioconductor.org

R News

by R Core Team

CHANGES IN R 3.6.2

NEW FEATURES

- `runmed(x,*)` gains a new option `na.action` determining *how* to handle `NaN` or `NA` in `x`.
- `dotchart()` gains new options `ann`, `xaxt`, `frame.plot` and `log`.

INSTALLATION on a UNIX-ALIKE

- Detection of the C stack direction has been moved from run-time to `configure`: this is safer with LTO builds and allows the detection to be overridden – see file 'config.site'.
- Source-code changes enable installation on platforms using `gcc -fno-common` (the expected default for `gcc 10.x`).

C-LEVEL FACILITIES

- `installTrChar` (which is nowadays is wrapped by `installChar`) is defined in 'Rinternals.h'. (Neither are part of the API.)

PACKAGE INSTALLATION

- Header 'Rconfig.h' contains the value of `FC_LEN_T` deduced at installation which is used by the prototypes in headers 'R_ext/BLAS.h' and 'R_ext/Lapack.h' but to avoid extensive breakage this is only exposed when `USE_FC_LEN_T` is defined.

  If a package's C/C++ calls to BLAS/LAPACK allow for the 'hidden' arguments used by most Fortran compilers to pass the lengths of Fortran character arguments, define `USE_FC_LEN_T` and include 'Rconfig.h' (possibly *via* 'R.h') before including 'R_ext/BLAS.h' or 'R_ext/Lapack.h'.

- A package with Fortran source code and perhaps C (but not C++) sources can request for its shared object/DLL to be linked by the Fortran compiler by including a line `USE_FC_TO_LINK=` in 'src/Makevars[.win]' and using `$(SHLIB_OPENMP_FFLAGS)` as part of `PKG_LIBS`.

  The known reason for doing so is a package which uses Fortran (only) OpenMP on a platform where the Fortran OpenMP runtime is incompatible with the C one (e.g. `gfortran` 9.x with `clang`).

UTILITIES

- `R CMD check` has a new option to mitigate checks leaving files/directories in '/tmp'. See the 'R Internals' manual – this is part of `--as-cran`.

Windows

- The default standard for C++ in package installation is C++11 (as it has been on other platforms where available since R 3.6.0: the default toolchain on Windows was defaulting to C++98).

DEPRECATED AND DEFUNCT

- Support for specifying C++98 in package installation is deprecated.
- Support in `R CMD config` for 'F77', 'FCPIFCPLAGS', 'CPP', 'CXXCPP' and 'CXX98' and similar is deprecated. ('CPP' is found from the system `make` and may well not be set.)

  Use '`$CC -E`' and '`$CXX -E`' instead of 'CPP' and 'CXXCPP'.

BUG FIXES

- `runmed(x,*)` when `x` contains missing values now works consistently for both `algorithm="Stuetzle"` and `"Turlach"`, and no longer segfaults for `"Turlach"`, as reported by Hilmar Berger.
- `apply(diag(3),2:3,mean)` now gives a helpful error message.
- `dgamma(x,shape,log=TRUE)` now longer overflows to `Inf` for shape < 1 and very small `x`, fixing PR#17577, reported by Jonathan Rougier.
- Buffer overflow in building error messages fixed. Reported by Benjamin Tremblay.
- `options(str = .)` is correctly initialized at package **utils** load time, now. A consequence is that `str()` in scripts now is more consistent to interactive use, e.g., when displaying function(**) argument lists.
- `as.numeric(<call>)` now gives correct error message.
- Printing `ls.str()` no longer wrongly shows `"<missing>"` in rare cases.
- Auto-printing S4 objects no longer duplicates the object, for faster speed and reduced memory consumption. Reported by Aaron Lun.
- `pchisq(<LRG>,<LRG>,ncp=100)` no longer takes practically forever in some cases. Hence ditto for corresponding `qchisq()` calls.
- `x %% L` for finite `x` no longer returns `NaN` when `L` is infinite, nor suffers from cancellation for large finite `L`, thanks to Long Qu's PR#17611.

  Analogously, `x %/% L` and `L %/% x` suffer less from cancellation and return values corresponding to limits for large `L`.
- `grepl(NA,*)` now returns `logical` as documented.
- `options(warn=1e11)` is an error now, instead of later leading to C stack overflow because of infinite recursion.
- `R_tryCatch` no longer transfers control for all conditions. Reported and patch provided by Lionel Henry in PR#17617.
- `format(object.size(.),digits=NULL)` now works, fixing PR#17628 reported by Jonathan Carroll.
- `get_all_vars(f,d)` now also works for cases, e.g. where `d` contains a matrix. Reported by Simon Wood in 2009 and patch provided by Ben Bolker in PR#13624.

  Additionally, it now also works when some variables are data frames, fixing PR#14905, reported by Patrick Breheny.
- `barplot()` could get spacings wrong if there were exactly two bars PR#15522. Patch by Michael Chirico.
- `power.t.test()` works in more cases when returning values of n smaller than 2.
- `dotchart(*,pch=.,groups=.)` now works better. Reported by Robert and confirmed by Nic Rochette in PR#16953.

- `canCoerce(obj,cl)` no longer assumes `length(class(obj)) == 1`.
- `plot.formula(*,subset = *)` now also works in a boundary case reported by Robert Schlicht (TU Dresden).
- `readBin()` and `writeBin()` of a `rawConnection()` now also work in large cases, thanks to a report and proposal by Taeke Harkema in PR#17665.

CHANGES IN R 3.6.1

INSTALLATION on a UNIX-ALIKE

- The default detection of the shell variable `libNN` is overridden for derivatives of Debian Linux, some of which have started to have a '/usr/lib64' directory. (E.g. Ubuntu 19.04.) As before, it can be specified in 'config.site'.

UTILITIES

- `R CMD config` knows the values of `AR` and `RANLIB`, often set for LTO builds.

DEPRECATED AND DEFUNCT

- The use of a character vector with `.Fortran()` is formally deprecated and gives a non-portability warning. (It has long been strongly discouraged in 'Writing R Extensions'.)

BUG FIXES

- On Windows, GUI package installation via `menuInstallPkgs()` works again, thanks to Len Weil's and Duncan Murdoch's PR#17556.
- `R CMD check` on `data()` fixing PR#17558 thanks to Duncan Murdoch.
- `quasi(*,variance = list(..))` now works more efficiently, and should work in all cases fixing PR#17560. Further, `quasi(var = mu(1-mu))` and `quasi(var = "mu ^ 3")` now work, and `quasi(variance = "log(mu)")` now gives a correct error message.
- Creation of lazy loading database during package installation is again robust to Rprofile changing the current working directory (PR#17559).
- `boxplot(y ~ f,horizontal=TRUE)` now produces correct x- and y-labels.
- `rbind.data.frame()` allows to keep `<NA>` levels from factor columns (PR#17562) via new option `factor.exclude`.

  Additionally, it works in one more case with matrix-columns which had been reported on 2017-01-16 by Krzysztof Banas.
- Correct messaging in C++ pragma checks in **tools** code for `R CMD check`, fixing PR#17566 thanks to Xavier Robin.
- print()ing and auto-printing no longer differs for `functions` with a user defined `print.function`, thanks to Bill Dunlap's report.
- On Windows, `writeClipboard(..,format = <n>)` now does correctly pass `format` to the underlying C code, thanks to a bug report (with patch) by Jenny Bryan.
- `as.data.frame()` treats 1D arrays the same as vectors, PR#17570.
- Improvements in `smoothEnds(x,*)` working with `NA`s (towards `runmed()` working in that case, in the next version of R).

- `vcov(glm(<quasi>),dispersion = *)` works correctly again, fixing PR#17571 thanks to Pavel Krivitsky.
- `R CMD INSTALL` of binary packages on Windows now works also with per-directory locking.
- `R CMD INSTALL` and `install.packages()` on Windows are now more robust against a locked file in an earlier installation of the package to be installed. The default value of option `install.lock` on Windows has been changed to `TRUE`.
- On Unix alikes (when `readline` is active), only expand tilde (~) file names starting with a tilde, instead of almost all tildes.
- In R documentation ('*.Rd') files, '`\item [..]`' is no longer treated specially when rendered in LaTeX and hence pdf, but rather shows the brackets in all cases.

CHANGES IN R 3.6.0

SIGNIFICANT USER-VISIBLE CHANGES

- Serialization format version 3 becomes the default for serialization and saving of the workspace (`save()`, `serialize()`, `saveRDS()`, `compiler::cmpfile()`). Serialized data in format 3 cannot be read by versions of R prior to version 3.5.0. Serialization format version 2 is still supported and can be selected by `version = 2` in the save/serialization functions. The default can be changed back for the whole R session by setting environment variables `R_DEFAULT_SAVE_VERSION` and `R_DEFAULT_SERIALIZE_VERSION` to `2`. For maximal back-compatibility, files 'vignette.rds' and 'partial.rdb' generated by `R CMD build` are in serialization format version 2, and resave by default produces files in serialization format version 2 (unless the original is already in format version 3).
- The default method for generating from a discrete uniform distribution (used in `sample()`, for instance) has been changed. This addresses the fact, pointed out by Ottoboni and Stark, that the previous method made `sample()` noticeably non-uniform on large populations. See PR#17494 for a discussion. The previous method can be requested using `RNGkind()` or `RNGversion()` if necessary for reproduction of old results. Thanks to Duncan Murdoch for contributing the patch and Gabe Becker for further assistance.

  The output of `RNGkind()` has been changed to also return the 'kind' used by `sample()`.

NEW FEATURES

- `Sys.setFileTime()` has been vectorized so arguments `path` and `time` of length greater than one are now supported.
- `axis()` gets new option `gap.axis = NA` for specifying a multiplication factor for the minimal "gap" (distance) between axis labels drawn. Its default is `1` for labels *parallel* to the axis, and `0.25` for perpendicular ones.

  Perpendicular labels no longer overlap, fixing bug PR#17384.
- The default method of `plot()` gains new arguments `xgap.axis = NA` and `ygap.axis = NA` to be passed to the x– and y– `axis(..,gap.axis=*)` calls.
- `removeSource()` now works not only for functions but also for some language objects.
- `as.call()`, `rep.int()`, `rep_len()` and `nchar()` dispatch internally.
- `is(object,class2)` looks for `class2` in the calling namespace after looking in the namespace of `class(object)`.

- `extendrange(..,f)` with a length-2 `f` now extends separately to the left and the right.
- `lengths()` dispatches internally to S4 methods.
- `download.file()` on Windows now uses `URLdecode()` to determine the file extension, and uses binary transfer (`mode = "wb"`) also for file extension '.rds'.

  The help page for `download.file()` now contains the same information on all platforms.
- Setting 'C' locale for collation *via* environment variables `LC_ALL` and `LC_COLLATE` and *via* a call to `Sys.setlocale()` now takes precedence over environment variable `R_ICU_LOCALE`.
- There is a new function, `nullfile()`, to give the file name of the null system device (e.g., '/dev/null') on the current platform.
- There are two new options, `keep.parse.data` and `keep.parse.data.pkgs`, which control whether parse data are included into sources when `keep.source` or `keep.source.pkgs` is `TRUE`. By default, `keep.parse.data.pkgs` is now `FALSE`, which changes previous behavior and significantly reduces space and time overhead when sources are kept when installing packages.
- In `rapply(x,..)`, `x` can also be "list-like" and of length $\geq 2^{31}$.
- `trimws()` gets new optional `whitespace` argument, allowing more extensive definitions of "space", such as including Unicode spaces (as wished in PR#17431).
- `weighted.mean()` no longer coerces the weights to a double/numeric vector, since `sum()` now handles integer overflow. This makes `weighted.mean()` more polymorphic and endomorphic, but be aware that the results are no longer guaranteed to be a vector of type 'double'.
- When loading namespaces, S3 method registrations which overwrite previous registrations are now noted by default (using `packageStartupMessage()`).
- `compiler::cmpfile()` gains a `version` argument, for use when the output file should be saved in serialization format 2.
- The axis labeling in the default method of `pairs()` may now be toggled by new options `horOdd` and `verOdd`.
- (Not Windows nor macOS.) Package **tcltk** now supports an environment variable `R_DONT_USE_TK` which if set disables Tk initialization. This is intended for use to circumvent errors in loading the package, e.g. with recent Linux running under an address sanitizer.
- The `numeric` method of `all.equal()` gets optional arguments `countEQ` and `formatFUN`. If `countEQ` is true, the mean error is more sensible when many entries are **eq**ual.
- `outer(x,y,FUN = "*")` is more efficient using `tcrossprod(u,v)` instead of `u %*% t(v)`.
- `vcov(<mlm>)` is more efficient via new optional arguments in `summary.mlm()`.
- The default method of `summary()` gets an option to choose the *kind* of `quantile()`s to use; wish of PR#17438.
- Fitting multiple linear models *via* `lm()` does work with *matrix* offsets, as suggested in PR#17407.
- The new functions `mem.maxVSize()` and `mem.maxMSize()` allow the maximal size of the vector heap and the maximal number of nodes allowed in the current `R` process to be queried and set.

- `news()` gains support for 'NEWS.md' files.
- An effort has been started to have our reference manuals, i.e., all help pages. show platform-independent information (rather than Windows or Unix-alike specifics visible only on that platform). Consequently, the Windows version of `X11()` / `x11()` got identical formal arguments to the Unix one.
- `sessionInfo()$running` has been factored out in a new variable `osVersion`.
- `slice.index()` now also works for multi-dimensional margins.
- `untar()` used with an external `tar` command assumes this supports decompression including `xz` and automagically detecting the compression type. This has been true of all mainstream implementations since 2009 (for GNU `tar`, since version 1.22): older implementations are still supported *via* the new argument `support_old_tars` whose default is controlled by environment variable `R_SUPPORT_OLD_TARS`. (It looks like NetBSD and OpenBSD have 'older' `tar` commands for this purpose.)
- The new function `asplit()` allow splitting an array or matrix by its margins.
- New functions `errorCondition()` and `warningCondition()` provide a convenient way to create structured error and warning objects.

  `.Deprecated()` now signals a warning of class `"deprecatedWarning"`, and `.Defunct()` now signals an error of class `"defunctError"`.
- Many 'package not found' errors are now signaled as errors of class `"packageNotFoundError"`.
- As an experimental feature, when `loadNamespace()` fails because the requested package is not available the error is initially signaled with a `retry_loadNamespace` restart available. This allows a calling handler to try to install the package and continue.
- `S3method()` directives in 'NAMESPACE' can now also be used to perform *delayed* S3 method registration.
- Experimentally, setting environment variable `_R_CHECK_LENGTH_1_LOGIC2_` will lead to warnings (or errors if the variable is set to a 'true' value) when `&&` or `||` encounter and use arguments of length more than one.
- Added `"lines"` and `"chars"` coordinate systems to `grconvertX()` and `grconvertY()`.
- `getOption()` is more efficient notably for the rare case when called with two arguments, from several contributors in PR#17394.
- In `.col(dim)` and `.row(dim)`, `dim` now may also be an integer-valued `"double"`.
- `sQuote()` and `dQuote()` get an explicit `q` argument with obvious default instead of using `getOption("fancyQuotes")` implicitly and unconditionally.
- `unzip()` can list archives with comments and with spaces in file names even using an external `unzip` command.
- Command line completion has a new setting `rc.settings(dots = FALSE)` to remove `...` from the list of possible function arguments.
- `library()` no longer checks packages with compiled code match '`R.version$platform`'. `loadNamespace()` never has, and increasingly the 'canonical name' does not reflect the important characteristics of compiled code.
- The primitive functions `drop()` and `unclass()` now avoid duplicating their data for atomic vectors that are large enough, by returning `ALTREP` wrapper objects with adjusted attributes. R-level assignments to change attributes will also use wrapper objects to avoid duplicating data for larger atomic vectors. R functions like `structure()` and `unname()` will therefore not duplicate data in these settings. Generic vectors as produced by `list()` are not yet covered by this optimization but may be in due course.

- In `formals()`, `envir` becomes an optional argument instead of being hardwired.
- Instead of signalling an error for an invalid S4 object `x`, `str(x)` now gives a warning and subsequently still shows most parts of `x`, e.g., when slots are missing.
- `gamma(x)` and `lgamma(x)` no longer warn when correctly returning `Inf` or underflowing to zero. This helps maximum likelihood and similar computations.
- `convertColor()` is now vectorized, so a lot faster for converting many colours at once. The new argument `vectorized` to `colorConverter()` ensures that non-vectorized colour converters still work. (Thanks to Brodie Gaslam.)
- `download.file()` and `url()` get new argument `headers` for custom HTTP headers, e.g., allowing to perform basic http authentication, thanks to a patch contributed by Gábor Csárdi.
- File-based connection functions `file()`, `gzfile()`, `bzfile()` and `xzfile()` now signal an error when used on a directory.
- For `approx()`, `splinefun()` *etc*, a new setting `ties = c("ordered",<fun>)` allows skipping the sorting and still treat ties.
- `format(x)` gives a more user friendly error message in the case where no method is defined. A minimal method is provided in `format.default(x)` when `isS4(x)` is true.
- `which(x)` now also works when `x` is a long vector, thanks to Suharto Anggono's PR#17201. **NB**: this may return a double result, breaking the previous guarantee of an integer result.
- `seq.default()` is more careful to return an `integer` (as opposed to `double`) result when its arguments are large and/or classed objects; see comment #9 of Suharto Anggono's PR#17497.
- The `plot()` method for `lm` and `glm` fits, `plot.lm()`, gains a new option `iter.smooth` with a default of `0` for binomial fits, no longer down-weighting when smoothing the residuals.
- `zip()` passes its list of files *via* standard input to the external command when too long for the command line (on some platforms).
- `data()` gains an `overwrite` argument.
- `t.test()` now also returns the standard error (in list component `stderr`).
- `model.matrix(*,contrasts.arg = CC)` now warns about invalid `contrasts.args`.
- Performance of `substr()` and `substring()` has been improved.
- `stopifnot()` has been simplified thanks to Suharto Anggono's proposals to become considerably faster for cheap expressions.
- The default 'user agent' has been changed when accessing '`http://`' and '`https://`' sites using '`libcurl`'. (A site was found which caused '`libcurl`' to infinite-loop with the previous default.)
- `sessionInfo()` now also contains `RNGkind()` and prints it when it differs from the default; based on a proposal and patch by Gabe Becker in PR#17535. Also, `RNGversion(getRversion())` works directly.
- `library()` and `require()` now allow more control over handling search path conflicts when packages are attached. The policy is controlled by the new `conflicts.policy` option.

- `barplot()` gets a formula method, thanks to a patch proposal by Arni Magnusson in PR#17521.
- `pmax()` and `pmin(x)` now also work for long vectors, thanks to Suharto Anggono's PR#17533.
- `bxp()` now warns when omitting duplicated arguments.
- New `hcl.colors()` function to provide wide range of HCL-based colour palettes with much better perceptual properties than the existing RGB/HSV-based palettes like `rainbow()`.

  Also a new `hcl.pals()` function to list available palette names for `hcl.colors()`.

  Contributed by Achim Zeileis.
- The default colours for `image()` and `filled.contour()` are now based on `hcl.colors()`.
- The palette-generating functions `rainbow()`, `gray.colors()`, etc. get a new `rev` argument to facilitate reversing the order of colors.
- New `str2lang()` and `str2expression()` as streamlined versions of `parse(text=.,keep.source=FALSE)` allow to abstract typical call constructions, e.g., in `formula` manipulations. (Somewhat experimental)
- Add `update_PACKAGES()` for incrementally updating a package repository index, instead of rebuilding the index from scratch. Thanks to Gabe Becker in PR#17544 for the patch, based on part of his **switchr** package.

INSTALLATION on a UNIX-ALIKE

- The options selected for the C++ compiler default to the C++11 standard if supported, otherwise to the C++98 standard.
- Visibility macros such as 'C_VISIBILITY' can now be user-set (including to empty), e.g. in 'config.site'.
- Macro 'FCLIBS', which has sometimes been needed on Solaris, has been renamed to 'FCLIBS_XTRA'.
- Macro 'F77' is always set to the value of 'FC', so the latter should be set to user-select the Fortran compiler for both fixed-form and free-form Fortran. In particular, `gfortran` is now the first choice for 'F77', not `f95`.

  Macros 'FFLAGS' and 'FCFLAGS' remain distinct to allow for a compiler which needs a flag to select free- or fixed-form Fortran (most use the source-file extension to choose: '.f' is fixed-form and '.f90' and '.f95' are free-form).

  If only one of them is set, its value is used for both.
- The special-casing of 'CFLAGS', 'CXXFLAGS' and 'FFLAGS' for Intel compilers on Linux has been removed: we do not have recent experience but the generic defaults now chosen are the same as those previously special-cased for 'x86_64'.

  If necessary, override the defaults on the `configure` command line or in file 'config.site'.
- Long-untested `configure` support for HP-UX and very old versions of Linux has been removed.
- `configure --with-blas` (without specifying a value) includes OpenBLAS in its search (before ATLAS and a generic BLAS). This follows recent versions of the 'ax_blas' autoconf macro.
- The `configure` macro 'MAKEINFO' has been updated to 'TEXI2ANY'.
- Support for `make install-strip` has been enhanced.

PACKAGE INSTALLATION

- Source package installation is by default 'staged': the package is installed into a temporary location under the final library directory and moved into place once the installation is complete. The benefit is that partially-installed packages are hidden from other R sessions.

  The overall default is set by environment variable `R_INSTALL_STAGED`. `R CMD INSTALL` has new options '`--staged-install`' and '`--no-staged-install`', and packages can use the '`StagedInstall`' field in their 'DESCRIPTION' file to opt out. (That opt-out is a temporary measure which may be withdrawn in future.)

  Staged installation requires either '`--pkglock`' or '`--lock`', one of which is used by default.

- The interpretation of source code with extension '.f' is changing. Previously this denoted FORTRAN 77 code, but current compilers no longer have a FORTRAN 77 mode and interpret it as 'fixed-form' Fortran 90 (or later where supported) code. Extensions '.f90' and '.f95' continue to indicate 'free-form' Fortran code.

  Legal FORTRAN 77 code is also legal fixed-form Fortran 9x; however this change legitimizes the use of later features, in particular to replace features marked 'obsolescent' in Fortran 90 and 'deleted' in Fortran 2018 which `gfortran` 8.x and later warn about.

- Packages containing files in the 'src' directory with extensions '.f90' or '.f95' are now linked using the C or C++ compiler rather than the Fortran 9x compiler. This is consistent with fixed-form Fortran code and allows mixing of C++ and free-form Fortran on most platforms.

  Consequentially, a package which includes free-form Fortran 9x code which uses OpenMP should include '`SHLIB_OPENMP_CFLAGS`' (or the '`CXXFLAGS`' version if they also include C++ code) in '`PKG_LIBS`' rather than '`SHLIB_OPENMP_FCFLAGS`' — fortunately on almost all current platforms they are the same flag.

- Macro '`PKG_FFLAGS`' will be used for the compilation of both fixed-form and free-form Fortran code unless '`PKG_FCFLAGS`' is also set (in 'src/Makevars' or 'src/Makevars.win').

- The make macro '`F_VISIBILITY`' is now preferred for both fixed-form and free-form Fortran, for use in 'src/Makevars' and similar.

- `R CMD INSTALL` gains a new option '`--strip`' which (where supported) strips installed shared object(s): this can also be achieved by setting the environment variable `_R_SHLIB_STRIP_` to a true value.

  The new option '`--strip-lib`' attempts stripping of static and shared libraries installed under 'lib'.

  These are most useful on platforms using GNU `binutils` (such as Linux) and compiling with '`-g`' flags.

- There is more support for installing UTF-8-encoded packages in a strict Latin-1 locale (and probably for other Latin locales): non-ASCII comments in R code (and 'NAMESPACE' files) are worked around better.

UTILITIES

- `R CMD check` now optionally checks makefiles for correct and portable use of the '`SHLIB_OPENMP_*FLAGS`' macros.

- `R CMD check` now evaluates `\Sexpr{}` expressions (including those in macros) before checking the contents of 'Rd' files and so detects issues both in evaluating the expressions and in the expanded contents.

- `R CMD check` now lists missing packages separated by commas and with regular quotes such as to be useful as argument in calling `install.packages(c(..))`; from a suggestion by Marcel Ramos.
- `tools::Rd2latex()` now uses UTF-8 as its default output encoding.
- `R CMD check` now checks line endings of files with extension '.hpp' and those under 'inst/include'. The check now includes that a non-empty file is terminated with a newline.

  `R CMD build` will correct line endings in such files.
- `R CMD check` now tries re-building all vignettes rather than stopping at the first error: whilst doing so it adds 'bookmarks' to the log. By default (see the 'R Internals' manual) it re-builds each vignette in a separate process.

  It now checks for duplicated vignette titles (also known as 'index entries'): they are used as hyperlinks on CRAN package pages and so do need to be unique.
- `R CMD check` has more comprehensive checks on the 'data' directory and the functioning of `data()` in a package.
- `R CMD check` now checks `autoconf`-generated 'configure' files have their corresponding source files, including optionally attempting to regenerate them on platforms with `autoreconf`.
- `R CMD build` has a new option '`--compression`' to select the compression used for the tarball.
- `R CMD build` now removes 'src/*.mod' files on all platforms.

C-LEVEL FACILITIES

- New pointer protection C functions `R_PreserveInMSet` and `R_ReleaseFromMSet` have been introduced to replace `UNPROTECT_PTR`, which is not safe to mix with `UNPROTECT` (and with `PROTECT_WITH_INDEX`). Intended for use in parsers only.
- `NAMEDMAX` has been raised to 7 to allow further protection of intermediate results from (usually ill-advised) assignments in arguments to `BUILTIN` functions. Properly written package code should not be affected.
- `R_unif_index` is now considered to be part of the C API.
- `R_GetCurrentEnv()` allows C code to retrieve the current environment.

DEPRECATED AND DEFUNCT

- Argument `compressed` of `untar()` is deprecated — it is only used for external `tar` commands which increasingly for extraction auto-detect compression and ignore their '`zjJ`' flags.
- `var(f)` and hence `sd(f)` now give an error for `factor` arguments; they gave a deprecation warning since R 3.2.3, PR#16564.
- Package **tools**' `vignetteDepends()` has been deprecated (it called a function deprecated since Feb 2016), being partly replaced by newly exported `vignetteInfo()`.
- The `f77_f2c` script has been removed: it no longer sufficed to compile the '.f' files in R.
- The deprecated legacy support of `make` macros such as '`CXX1X`' has been removed: use the '`CXX11`' forms instead.
- Make macro '`F77_VISIBILITY`' is deprecated in favour of '`F_VISIBILITY`'.

- Make macros 'F77', 'FCPIFCPLAGS' and 'SHLIB_OPENMP_FCFLAGS' are deprecated in favour of 'FC', 'FPICFLAGS' and 'SHLIB_OPENMP_FFLAGS' respectively.
- `$.data.frame` had become an expensive version of the default method, so has been removed. (Thanks to Radford Neal for picking this up and to Duncan Murdoch for providing a patch.)

BUG FIXES

- `replayPlot(r)` now also works in the same R session when `r` has been "reproduced" from serialization, typically after saving to and reading from an RDS file.
- `substr()` and `substring()` now signal an error when the input is invalid UTF-8.
- `file.copy()` now works also when its argument `to` is of length greater than one.
- `mantelhaen.test()` no longer suffers from integer overflow in largish cases, thanks to Ben Bolker's PR#17383.
- Calling `setGeneric("foo")` in a package no longer fails when the enclosing environment of the implicit generic `foo()` is `.GlobalEnv`.
- `untar(file("<some>.tar.gz"),*)` now gives a better error message, suggesting to use `gzfile()` instead.
- Method dispatch uses more relevant environments when looking up class definitions.
- The documentation for `identify()` incorrectly claimed that the indices of identified points were returned in the order that the points were selected. `identify()` now has a new argument `order` to allow the return value to include the order in which points were identified; the documentation has been updated. Reported by Richard Rowe and Samuel Granjeaud.
- `order(....,decreasing=c(TRUE,FALSE))` could fail in some cases. Reported from StackOverflow via Karl Nordström.
- User macros in Rd files now accept empty and multi-line arguments.
- Changes in `print.*()`, thanks to Lionel Henry's patches in PR#17398:
  - Printing lists, pairlists or attributes containing calls with S3 class no longer evaluate those.
  - Printing S4 objects within lists and pairlists dispatches with `show()` rather than `print()`, as with auto-printing.
  - The indexing tags (names or `[[<n>]]`) of recursive data structures are now printed correctly in complex cases.
  - Arguments supplied to `print()` are now properly forwarded to methods when printing lists, pairlists or attributes containing S3 objects.
  - The print parameters are now preserved when printing S3 objects or deparsing symbols and calls. Previously, printing lists containing S3 objects or expressions would reset these parameters.
  - Printing lists, pairlists or attributes containing functions now uses `srcref` attributes if present.
- Calling `install.packages()` with a length zero `pkgs` argument now is a no-op (PR#17422).
- `unlist(x)` now returns a correct `factor` when `x` is a nested list with factor leaves, fixing PR#12572 and PR#17419.
- The documentation `help(family)` gives more details about the `aic` component, thanks to Ben Bolker's prompting.

- The documentation for `attributes` and `` `attributes<-` `` now gives `x` as name of the first and main argument which the implementation has been requiring, fixing PR#17434. For consistency, the first argument name is also changed from `obj` to `x` for `` `mostattributes<-` ``.
- `strwidth()` now uses `par("font")` as default font face (PR#17352).
- `plot(<table>,log="x")` no longer warns about `log`.
- The `print()` method for `"htest"` objects now formats the test statistic and parameter directly and hence no longer rounds to units *before* the decimal point. Consequently, printing of `t.test()` results with a small number of digits now shows non-large `df`'s to the full precision (PR#17444).
- `kruskal.test()` and `fligner.test()` no longer erroneously insist on numeric `g` group arguments (PR#16719).
- Printing a news db via the browser now does a much better job (PR#17433).
- `print.aov()` missed column names in the multivariate case due to misspelling (reported by Chris Andrews).
- `axis()` now creates valid `at` locations also for small subnormal number ranges in log scale plots.
- `format.POSIXlt()` now also recycles the `zone` and `gmtoff` list components to full length when needed, and its internal C code detects `have_zone` in more cases. In some cases, this changes its output to become compatible with `format.POSIXct()`.
- On Windows, `detectCores()` in package **parallel** now detects processors in all processor groups, not just the group R is running in (impacts particularly systems with more than 64 logical processors). Reported by Arunkumar Srinivasan.
- On Windows, `socketSelect()` would hang with more than 64 sockets, and hence `parallel::clusterApplyLB()` would hang with more than 64 workers. Reported by Arunkumar Srinivasan.
- `as(1L,"double")` now does coerce (PR#17457).
- `lm.influence()`, `influence.measures()`, `rstudent()` etc now work (more) correctly for multivariate models (`"mlm"`), thanks to (anonymous) stackoverflow remarks.
- `sample.int(2.9,*,replace=TRUE)` again behaves as documented and as in R < 3.0.0, namely identically to `sample.int(2,..)`.
- Fixes to `convertColor()` for chromatic adaptation; thanks to Brodie Gaslam PR#17473.
- Using `\Sexpr[stage=install]{..}` to create an 'Rd' section no longer gives a warning in `R CMD check`; problem originally posted by Gábor Csárdi, then reported as PR#17479 with a partial patch by Duncan Murdoch.
- Parse data now include a special node for equal assignment.
- `split.default()` no longer relies on `[[<-()`, so it behaves as expected when splitting an object by a factor with the empty string as one of its levels. Thanks to Brad Friedman for the report.
- Line numbers in messages about '.Rd' files are now more reliable, thanks to a patch from Duncan Murdoch.
- In the `numeric` method for `all.equal()`, a numeric `scale` argument is now checked to be positive and allowed to be of length > 1. (The latter worked originally and with a warning in recent years).

- Deferred string conversions now record the `OutDec` option setting when not equal to the default. Reported by Michael Sannella.
- When `y` is numeric and `f` a `factor`, `plot(y ~ f)` nicely uses "y" and "f" as y- and x-labels. The more direct `boxplot(y ~ f)` now does too. The new argument `ann = FALSE` may be used to suppress these.
- Subassignment to no/empty rows of a data frame is more consistent and typically a no-op in all cases instead of sometimes an error; part of Emil Bode's PR#17483.
- Calls like `formatC(*,zero.print = "<0.001")` no longer give an error and are further improved via new optional argument `replace.zero`. Reported by David Hugh-Jones.
- `methods::formalArgs("<fn>")` now finds the same function as `formals("<fn>")`, fixing Emil Bode's PR#17499.
- The **methods** package better handles duplicated class names across packages.
- The default method of `seq()` now avoids integer overflow, thanks to the report and "cumsum" patch of Suharto Anggono's PR#17497.
- `sub()` no longer loses encodings for non-ASCII replacements (PR#17509).
- Fix for rotated raster image on X11 device. (Partial fix for PR#17148; thanks to Mikko Korpela).
- `formula(model.frame(frml,..))` now returns `frml` in all cases, thanks to Bill Dunlap. The previous behavior is available as `DF2formula(<model.frame>)`.
- `ar.ols()` also returns scalar `var.pred` in univariate case (PR#17517).
- `normalizePath()` now treats `NA` path as non-existent and normalizes it to `NA`. `file.access()` treats `NA` file name as non-existent. `file.edit()` and connection functions such as `file()` now treat `NA` file names as errors.
- The internal `regularize.values()` auxiliary of `approx()`, `splinefun()` etc now warns again when there are ties and the caller did not specify `ties`. Further, it no longer duplicates `x` and `y` unnecessarily when `x` is already sorted (PR#17515).
- `strtoi("",base)` now gives `NA` on all platforms, following its documentation. Reported by Michael Chirico.
- In the definition of an S4 class, prototype elements are checked against the slots of the class, with giving a prototype for an undefined slot now being an error. (Reported by Bill Dunlap.)
- From `setClassUnion()`, if environment variable `_R_METHODS_SHOW_CHECKSUBCLASSES` is set to true, the internal `.checkSubclasses()` utility prints debugging info to see where it is used.
- `max.col(m)` with an `m` of zero columns now returns integer `NA` (instead of `1`).
- `axTicks()` no longer returns small "almost zero" numbers (in exponential format) instead of zero, fixing Ilario Gelmetti's PR#17534.
- `isSymmetric(matrix(0,dimnames=list("A","b")))` is `FALSE` again, as always documented.
- The `cairo_pdf` graphics device (and other Cairo-based devices) now clip correctly to the right and bottom border.

  There was an off-by-one-pixel bug, reported by Lee Kelvin.

- `as.roman(3) <= 2:4` and all other comparisons now work, as do group `"Summary"` function calls such as `max(as.roman(sample(20)))` and `as.roman(NA)`. (Partly reported by Bill Dunlap in PR#17542.)
- `reformulate("x",response = "sin(y)")` no longer produces extra back quotes, PR#17359, and gains new optional argument `env`.
- When reading console input from 'stdin' with re-encoding (`R --encoding=enc <input`) the code on a Unix-alike now ensures that each converted input line is terminated with a newline even if re-encoding fails.
- `as.matrix.data.frame()` now produces better strings from logicals, thanks to PR#17548 from Gabe Becker.
- The S4 generic signature of `rowSums()`, `rowMeans()`, `colSums()` and `colMeans()` is restricted to `"x"`.
- `match(x,tab)` now works for long *character* vectors `x`, thanks to PR#17552 by Andreas Kersting.
- Class unions are unloaded when their namespace is unloaded (PR#17531, adapted from a patch by Brodie Gaslam).
- `selectMethod()` is robust to ANY-truncation of method signatures (thanks to Herve Pages for the report).

CHANGES IN R 3.5.3

INSTALLATION on a UNIX-ALIKE

- Detection of flags for C++98/11/14/17 has been improved: in particular if CXX??STD is set, it is tried first with no additional flags.

PACKAGE INSTALLATION

- New macro '`F_VISIBILITY`' as an alternative to '`F77_VISIBILITY`'. This will become the preferred form in R 3.6.0.

BUG FIXES

- `writeLines(readLines(fnam),fnam)` now works as expected, thanks to Peter Meissner's PR#17528.
- `setClassUnion()` no longer warns, but uses `message()` for now, when encountering "non local" subclasses of class members.
- `stopifnot(exprs = T)` no longer fails.

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.

CHANGES IN R 3.5.0

SIGNIFICANT USER-VISIBLE CHANGES

- All packages are by default byte-compiled on installation. This makes the installed packages larger (usually marginally so) and may affect the format of messages and tracebacks (which often exclude `.Call` and similar).

NEW FEATURES

- `factor()` now uses `order()` to sort its levels, rather than `sort.list()`. This allows `factor()` to support custom vector-like objects if methods for the appropriate generics are defined. It has the side effect of making `factor()` succeed on empty or length-one non-atomic vector(-like) types (e.g., `"list"`), where it failed before.
- `diag()` gets an optional `names` argument: this may require updates to packages defining S4 methods for it.
- `chooseCRANmirror()` and `chooseBioCmirror()` no longer have a `useHTTPS` argument, not needed now all R builds support '`https://`' downloads.
- New `summary()` method for `warnings()` with a (somewhat experimental) `print()` method.
- (**methods** package.) `.self` is now automatically registered as a global variable when registering a reference class method.
- `tempdir(check = TRUE)` recreates the `tempdir()` directory if it is no longer valid (e.g. because some other process has cleaned up the '/tmp' directory).
- New `askYesNo()` function and `"askYesNo"` option to ask the user binary response questions in a customizable but consistent way. (Suggestion of PR#17242.)
- New low level utilities `...elt(n)` and `...length()` for working with `...` parts inside a function.
- `isTRUE()` is more tolerant and now true in

```
x <- rlnorm(99)
isTRUE(median(x) == quantile(x)["50%"])
```

New function `isFALSE()` defined analogously to `isTRUE()`.

- The default symbol table size has been increased from 4119 to 49157; this may improve the performance of symbol resolution when many packages are loaded. (Suggested by Jim Hester.)
- `line()` gets a new option `iter = 1`.
- Reading from connections in text mode is buffered, significantly improving the performance of `readLines()`, as well as `scan()` and `read.table()`, at least when specifying `colClasses`.
- `order()` is smarter about picking a default sort `method` when its arguments are objects.
- `available.packages()` has two new arguments which control if the values from the per-session repository cache are used (default true, as before) and if so how old cached values can be used (default one hour).

  These arguments can be passed from `install.packages()`, `update.packages()` and functions calling that: to enable this `available.packages()`, `packageStatus()` and `download.file()` gain a `...` argument.
- `packageStatus()`'s `upgrade()` method no longer ignores its `...` argument but passes it to `install.packages()`.
- `installed.packages()` gains a `...` argument to allow arguments (including `noCache`) to be passed from `new.packages()`, `old.packages()`, `update.packages()` and `packageStatus()`.
- `factor(x,levels,labels)` now allows duplicated `labels` (not duplicated `levels`!). Hence you can map different values of `x` to the same level directly.
- Attempting to use `names<-()` on an S4 derivative of a basic type no longer emits a warning.
- The `list` method of `within()` gains an option `keepAttrs = FALSE` for some speed-up.
- `system()` and `system2()` now allow the specification of a maximum elapsed time ('timeout').
- `debug()` supports debugging of methods on any object of S4 class `"genericFunction"`, including group generics.
- Attempting to increase the length of a variable containing `NULL` using `length()<-` still has no effect on the target variable, but now triggers a warning.
- `type.convert()` becomes a generic function, with additional methods that operate recursively over list and `data.frame` objects. Courtesy of Arni Magnusson (PR#17269).
- `lower.tri(x)` and `upper.tri(x)` only needing `dim(x)` now work via new functions `.row()` and `.col()`, so no longer call `as.matrix()` by default in order to work efficiently for all kind of matrix-like objects.
- `print()` methods for `"xgettext"` and `"xngettext"` now use `encodeString()` which keeps, e.g. `"\n"`, visible. (Wish of PR#17298.)
- `package.skeleton()` gains an optional `encoding` argument.
- `approx()`, `spline()`, `splinefun()` and `approxfun()` also work for long vectors.

- `deparse()` and `dump()` are more useful for S4 objects, `dput()` now using the same internal C code instead of its previous imperfect workaround R code. S4 objects now typically deparse perfectly, i.e., can be recreated identically from deparsed code.

  `dput()`, `deparse()` and `dump()` now print the `names()` information only once, using the more readable `(tag = value)` syntax, notably for `list()`s, i.e., including data frames.

  These functions gain a new control option `"niceNames"` (see `.deparseOpts()`), which when set (as by default) also uses the `(tag = value)` syntax for atomic vectors. On the other hand, without deparse options `"showAttributes"` and `"niceNames"`, names are no longer shown also for lists. `as.character(list( c (one = 1)))` now includes the name, as `as.character(list(list(one = 1)))` has always done.

  `m:n` now also deparses nicely when $m > n$.

  The `"quoteExpressions"` option, also part of `"all"`, no longer `quote()`s formulas as that may not re-parse identically. (PR#17378)

- If the option `setWidthOnResize` is set and `TRUE`, R run in a terminal using a recent `readline` library will set the `width` option when the terminal is resized. Suggested by Ralf Goertz.

- If multiple `on.exit()` expressions are set using `add = TRUE` then all expressions will now be run even if one signals an error.

- `mclapply()` gets an option `affinity.list` which allows more efficient execution with heterogeneous processors, thanks to Helena Kotthaus.

- The `character` methods for `as.Date()` and `as.POSIXlt()` are more flexible *via* new arguments `tryFormats` and `optional`: see their help pages.

- `on.exit()` gains an optional argument `after` with default `TRUE`. Using `after = FALSE` with `add = TRUE` adds an exit expression before any existing ones. This way the expressions are run in a first-in last-out fashion. (From Lionel Henry.)

- On Windows, `file.rename()` internally retries the operation in case of error to attempt to recover from possible anti-virus interference.

- Command line completion on '`::`' now also includes lazy-loaded data.

- If the `TZ` environment variable is set when date-time functions are first used, it is recorded as the session default and so will be used rather than the default deduced from the OS if `TZ` is subsequently unset.

- There is now a `[` method for class `"DLLInfoList"`.

- `glm()` and `glm.fit` get the same `singular.ok = TRUE` argument that `lm()` has had forever. As a consequence, in `glm(*,method = <your_own>)`, user specified methods need to accept a `singular.ok` argument as well.

- `aspell()` gains a filter for Markdown ('.md' and '.Rmd') files.

- `intToUtf8(multiple = FALSE)` gains an argument to allow surrogate pairs to be interpreted.

- The maximum number of DLLs that can be loaded into R e.g. *via* `dyn.load()` has been increased up to 614 when the OS limit on the number of open files allows.

- `Sys.timezone()` on a Unix-alike caches the value at first use in a session: *inter alia* this means that setting `TZ` later in the session affects only the *current* time zone and not the *system* one.

  `Sys.timezone()` is now used to find the system timezone to pass to the code used when R is configured with '`--with-internal-tzcode`'.

- When `tar()` is used with an external command which is detected to be GNU `tar` or libarchive `tar` (aka `bsdtar`), a different command-line is generated to circumvent line-length limits in the shell.
- `system(*, intern = FALSE)`, `system2()` (when not capturing output), `file.edit()` and `file.show()` now issue a warning when the external command cannot be executed.
- The "default" (`"lm"` etc) methods of `vcov()` have gained new optional argument `complete = TRUE` which makes the `vcov()` methods more consistent with the `coef()` methods in the case of singular designs. The former (back-compatible) behavior is given by `vcov(*, complete = FALSE)`.
- `coef()` methods (for `lm` etc) also gain a `complete = TRUE` optional argument for consistency with `vcov()`.
  For `"aov"`, both `coef()` and `vcov()` methods remain back-compatibly consistent, using the *other* default, `complete = FALSE`.
- `attach(*, pos = 1)` is now an error instead of a warning.
- New function `getDefaultCluster()` in package **parallel** to get the default cluster set via `setDefaultCluster()`.
- `str(x)` for atomic objects `x` now treats both cases of `is.vector(x)` similarly, and hence much less often prints `"atomic"`. This is a slight non-back-compatible change producing typically both more informative and shorter output.
- `gc()` gets new argument `full`.
- `write.dcf()` gets optional argument `useBytes`.
- New, partly experimental `packageDate()` which tries to get a valid `"Date"` object from a package 'DESCRIPTION' file, thanks to suggestions in PR#17324.
- `tools::resaveRdaFiles()` gains a `version` argument, for use when packages should remain compatible with earlier versions of R.
- `ar.yw(x)` and hence by default `ar(x)` now work when `x` has `NA`s, mostly thanks to a patch by Pavel Krivitsky in PR#17366. The `ar.yw.default()`'s AIC computations have become more efficient by using `determinant()`.
- New `warnErrList()` utility (from package **nlme**, improved).
- By default the (arbitrary) signs of the loadings from `princomp()` are chosen so the first element is non-negative.
- If '`--default-packages`' is not used, then `Rscript` now checks the environment variable `R_SCRIPT_DEFAULT_PACKAGES`. If this is set, then it takes precedence over `R_DEFAULT_PACKAGES`. If default packages are not specified on the command line or by one of these environment variables, then `Rscript` now uses the same default packages as `R`. For now, the previous behavior of not including **methods** can be restored by setting the environment variable `R_SCRIPT_LEGACY` to '`yes`'.
- When a package is found more than once, the warning from `find.package(*, verbose=TRUE)` lists all library locations.
- POSIXt objects can now also be rounded or truncated to month or year.
- `stopifnot()` can be used alternatively via new argument `exprs` which is nicer and useful when testing several expressions in one call.
- The environment variable `R_MAX_VSIZE` can now be used to specify the maximal vector heap size. On macOS, unless specified by this environment variable, the maximal vector heap size is set to the maximum of 16GB and the available physical memory. This is to avoid having the `R` process killed when macOS over-commits memory.

- `sum(x)` and `sum(x1,x2,..,x<N>)` with many or long logical or integer vectors no longer overflows (and returns `NA` with a warning), but returns `double` numbers in such cases.
- Single components of `"POSIXlt"` objects can now be extracted and replaced via `[` indexing with 2 indices.
- S3 method lookup now searches the namespace registry after the top level environment of the calling environment.
- Arithmetic sequences created by `1:n`, `seq_along`, and the like now use compact internal representations via the `ALTREP` framework. Coercing integer and numeric vectors to character also now uses the `ALTREP` framework to defer the actual conversion until first use.
- Finalizers are now run with interrupts suspended.
- `merge()` gains new option `no.dups` and by default suffixes the second of two duplicated column names, thanks to a proposal by Scott Ritchie (and Gabe Becker).
- `scale.default(x,center,scale)` now also allows `center` or `scale` to be "numeric-alike", i.e., such that `as.numeric(.)` coerces them correctly. This also eliminates a wrong error message in such cases.
- `par*apply` and `par*applyLB` gain an optional argument `chunk.size` which allows to specify the granularity of scheduling.
- Some `as.data.frame()` methods, notably the `matrix` one, are now more careful in not accepting duplicated or `NA` row names, and by default produce unique non-NA row names. This is based on new function `.rowNamesDF(x,make.names = *) <-rNms` where the logical argument `make.names` allows to specify *how* invalid row names `rNms` are handled. `.rowNamesDF()` is a "workaround" compatible default.
- R has new serialization format (version 3) which supports custom serialization of `ALTREP` framework objects. These objects can still be serialized in format 2, but less efficiently. Serialization format 3 also records the current native encoding of unflagged strings and converts them when de-serialized in R running under different native encoding. Format 3 comes with new serialization magic numbers (RDA3, RDB3, RDX3). Format 3 can be selected by `version = 3` in `save()`, `serialize()` and `saveRDS()`, but format 2 remains the default for all serialization and saving of the workspace. Serialized data in format 3 cannot be read by versions of R prior to version 3.5.0.
- The `"Date"` and "date-time" classes `"POSIXlt"` and `"POSIXct"` now have a working `length<-()` method, as wished in PR#17387.
- `optim(*,control = list(warn.1d.NelderMead = FALSE))` allows to turn off the warning when applying the default `"Nelder-Mead"` method to 1-dimensional problems.
- `matplot(..,panel.first = .)` etc now work, as `log` becomes explicit argument and `...` is passed to `plot()` unevaluated, as suggested by Sebastian Meyer in PR#17386.
- Interrupts can be suspended while evaluating an expression using `suspendInterrupts`. Subexpression can be evaluated with interrupts enabled using `allowInterrupts`. These functions can be used to make sure cleanup handlers cannot be interrupted.
- R 3.5.0 includes a framework that allows packages to provide alternate representations of basic R objects (`ALTREP`). The framework is still experimental and may undergo changes in future R releases as more experience is gained. For now, documentation is provided in `https://svn.r-project.org/R/branches/ALTREP/ALTREP.html`.

UTILITIES

- `install.packages()` for source packages now has the possibility to set a 'timeout' (elapsed-time limit). For serial installs this uses the `timeout` argument of `system2()`: for parallel installs it requires the `timeout` utility command from GNU **coreutils**.
- It is now possible to set 'timeouts' (elapsed-time limits) for most parts of `R CMD check` *via* environment variables documented in the 'R Internals' manual.
- The 'BioC extra' repository which was dropped from Bioconductor 3.6 and later has been removed from `setRepositories()`. This changes the mapping for 6–8 used by `setRepositories(ind=)`.
- `R CMD check` now also applies the settings of environment variables `_R_CHECK_SUGGESTS_ONLY_` and `_R_CHECK_DEPENDS_ONLY_` to the re-building of vignettes.
- `R CMD check` with environment variable `_R_CHECK_DEPENDS_ONLY_` set to a true value makes test-suite-management packages available and (for the time being) works around a common omission of **rmarkdown** from the '`VignetteBuilder`' field.

INSTALLATION on a UNIX-ALIKE

- Support for a system Java on macOS has been removed — install a fairly recent Oracle Java (see 'R Installation and Administration' §C.3.2).
- `configure` works harder to set additional flags in '`SAFE_FFLAGS`' only where necessary, and to use flags which have little or no effect on performance.

  In rare circumstances it may be necessary to override the setting of '`SAFE_FFLAGS`'.
- C99 functions `expm1`, `hypot`, `log1p` and `nearbyint` are now required.
- `configure` sets a '`-std`' flag for the C++ compiler for all supported C++ standards (e.g., '`-std=gnu++11`' for the C++11 compiler). Previously this was not done in a few cases where the default standard passed the tests made (e.g. `clang 6.0.0` for C++11).

C-LEVEL FACILITIES

- 'Writing R Extensions' documents macros `MAYBE_REFERENCED`, `MAYBE_SHARED` and `MARK_NOT_MUTABLE` that should be used by package `C` code instead `NAMED` or `SET_NAMED`.
- The object header layout has been changed to support merging the `ALTREP` branch. This requires re-installing packages that use compiled code.
- 'Writing R Extensions' now documents the `R_tryCatch`, `R_tryCatchError`, and `R_UnwindProtect` functions.
- `NAMEDMAX` has been raised to 3 to allow protection of intermediate results from (usually ill-advised) assignments in arguments to `BUILTIN` functions. Package `C` code using `SET_NAMED` may need to be revised.

DEPRECATED AND DEFUNCT

- `Sys.timezone(location = FALSE)` is defunct, and is ignored (with a warning).
- `methods:::bind_activation()` is defunct now; it typically has been unneeded for years.

  The undocumented 'hidden' objects `.__H__.cbind` and `.__H__.rbind` in package **base** are deprecated (in favour of `cbind` and `rbind`).
- The declaration of `pythag()` in 'Rmath.h' has been removed — the entry point has not been provided since R 2.14.0.

BUG FIXES

- `printCoefmat()` now also works without column names.
- The S4 methods on `Ops()` for the `"structure"` class no longer cause infinite recursion when the structure is not an S4 object.
- `nlm(f,..)` for the case where `f()` has a `"hessian"` attribute now computes $LL' = H + \mu I$ correctly. (PR#17249).
- An S4 method that "rematches" to its generic and overrides the default value of a generic formal argument to `NULL` no longer drops the argument from its formals.
- `Rscript` can now accept more than one argument given on the '`#!`' line of a script. Previously, one could only pass a single argument on the '`#!`' line in Linux.
- Connections are now written correctly with encoding `"UTF-16LE"`. (PR#16737).
- Evaluation of `..0` now signals an error. When `..1` is used and `...` is empty, the error message is more appropriate.
- (Windows mainly.) Unicode code points which require surrogate pairs in UTF-16 are now handled. All systems should properly handle surrogate pairs, even those systems that do not need to make use of them. (PR#16098)
- `stopifnot(e,e2,...)` now evaluates the expressions sequentially and in case of an error or warning shows the relevant expression instead of the full `stopifnot(..)` call.
- `path.expand()` on Windows now accepts paths specified as UTF-8-encoded character strings even if not representable in the current locale. (PR#17120)
- `line(x,y)` now correctly computes the medians of the left and right group's x-values and in all cases reproduces straight lines.
- Extending S4 classes with slots corresponding to special attributes like `dim` and `dimnames` now works.
- Fix for `legend()` when `fill` has multiple values the first of which is `NA` (all colours used to default to `par(fg)`). (PR#17288)
- `installed.packages()` did not remove the cached value for a library tree that had been emptied (but would not use the old value, just waste time checking it).
- The documentation for `installed.packages(noCache = TRUE)` incorrectly claimed it would refresh the cache.
- `aggregate(<data.frame>)` no longer uses spurious names in some cases. (PR#17283)
- `object.size()` now also works for long vectors.
- `packageDescription()` tries harder to solve re-encoding issues, notably seen in some Windows locales. This fixes the `citation()` issue in PR#17291.
- `poly(<matrix>,3)` now works, thanks to prompting by Marc Schwartz.
- `readLines()` no longer segfaults on very large files with embedded `'\0'` (aka 'nul') characters. (PR#17311)
- `ns()` (package **splines**) now also works for a single observation. `interpSpline()` gives a more friendly error message when the number of points is less than four.
- `dist(x,method = "canberra")` now uses the correct definition; the result may only differ when `x` contains values of differing signs, e.g. not for 0-1 data.

- `methods:::cbind()` and `methods:::rbind()` avoid deep recursion, thanks to Suharto Anggono via PR#17300.
- Arithmetic with zero-column data frames now works more consistently; issue raised by Bill Dunlap.

  Arithmetic with data frames gives a data frame for `^` (which previously gave a numeric matrix).
- `pretty(x,n)` for large `n` or large `diff(range(x))` now works better (though it was never meant for large `n`); internally it uses the same rounding fuzz (1e-10) as `seq.default()` — as it did up to 2010-02-03 when both were 1e-7.
- Internal C-level `R_check_class_and_super()` and hence `R_check_class_etc()` now also consider non-direct super classes and hence return a match in more cases. This e.g., fixes behaviour of derived classes in package **Matrix**.
- Reverted unintended change in behavior of `return` calls in `on.exit` expressions introduced by stack unwinding changes in R 3.3.0.
- Attributes on symbols are now detected and prevented; attempt to add an attribute to a symbol results in an error.
- `fisher.test(*,workspace = <n>)` now may also increase the internal stack size which allows larger problem to be solved, fixing PR#1662.
- The **methods** package no longer directly copies slots (attributes) into a prototype that is of an "abnormal" (reference) type, like a symbol.
- The **methods** package no longer attempts to call `length<-()` on `NULL` (during the bootstrap process).
- The **methods** package correctly shows methods when there are multiple methods with the same signature for the same generic (still not fully supported, but at least the user can see them).
- `sys.on.exit()` is now always evaluated in the right frame. (From Lionel Henry.)
- `seq.POSIXt(*,by = "<n>DSTdays")` now should work correctly in all cases and is faster. (PR#17342)
- `.C()` when returning a logical vector now always maps values other than FALSE and NA to TRUE (as documented).
- Subassignment with zero length vectors now coerces as documented (PR#17344). Further, `x <-numeric(); x[1] <-character()` now signals an error 'replacement has length zero' (or a translation of that) instead of doing nothing.
- (Package **parallel**.) `mclapply()`, `pvec()` and `mcparallel()` (when `mccollect()` is used to collect results) no longer leave zombie processes behind.
- `R CMD INSTALL <pkg>` now produces the intended error message when, e.g., the `LazyData` field is invalid.
- `as.matrix(dd)` now works when the data frame `dd` contains a column which is a data frame or matrix, including a 0-column matrix/d.f. .
- `mclapply(X,mc.cores)` now follows its documentation and calls `lapply()` in case `mc.cores = 1` also in the case `mc.preschedule` is false. (PR#17373)
- `aggregate(<data.frame>,drop=FALSE)` no longer calls the function on <empty> parts but sets corresponding results to `NA`. (Thanks to Suharto Anggono's patches in PR#17280).

- The `duplicated()` method for data frames is now based on the `list` method (instead of string coercion). Consequently `unique()` is better distinguishing data frame rows, fixing PR#17369 and PR#17381. The methods for matrices and arrays are changed accordingly.
- Calling `names()` on an S4 object derived from `"environment"` behaves (by default) like calling `names()` on an ordinary environment.
- `read.table()` with a non-default separator now supports quotes following a non-whitespace character, matching the behavior of `scan()`.
- `parLapplyLB` and `parSapplyLB` have been fixed to do load balancing (dynamic scheduling). This also means that results of computations depending on random number generators will now really be non-reproducible, as documented.
- Indexing a list using dollar and empty string (`l$""`) returns NULL.
- Using `\usage{ data(<name>,package="<pkg>") }` no longer produces `R CMD check` warnings.
- `match.arg()` more carefully chooses the environment for constructing default `choices`, fixing PR#17401 as proposed by Duncan Murdoch.
- Deparsing of consecutive `!` calls is now consistent with deparsing unary `-` and `+` calls and creates code that can be reparsed exactly; thanks to a patch by Lionel Henry in PR#17397. (As a side effect, this uses fewer parentheses in some other deparsing involving `!` calls.)

CHANGES IN R 3.4.4

NEW FEATURES

- `Sys.timezone()` tries more heuristics on Unix-alikes and so is more likely to succeed (especially on Linux). For the slowest method, a warning is given recommending that `TZ` is set to avoid the search.
- The version of LAPACK included in the sources has been updated to 3.8.0 (for the routines used by R, a very minor bug-fix change).
- `parallel::detectCores(logical = FALSE)` is ignored on Linux systems, since the information is not available with virtualized OSes.

INSTALLATION on a UNIX-ALIKE

- `configure` will use `pkg-config` to find the flags to link to `jpeg` if available (as it should be for the recently-released `jpeg-9c` and `libjpeg-turbo`). (This amends the code added in R 3.3.0 as the module name in `jpeg-9c` is not what that tested for.)

DEPRECATED AND DEFUNCT

- `Sys.timezone(location = FALSE)` (which was a stop-gap measure for Windows long ago) is deprecated. It no longer returns the value of environment variable `TZ` (usually a location).
- Legacy support of `make` macros such as '`CXX1X`' is formally deprecated: use the '`CXX11`' forms instead.

BUG FIXES

- `power.prop.test()` now warns when it cannot solve the problem, typically because of impossible constraints. (PR#17345)
- `removeSource()` no longer erroneously removes `NULL` in certain cases, thanks to Dénes Tóth.
- `nls(` NO [mol/l]` ~ f(t))` and `nls(y ~ a)` now work. (Partly from PR#17367)
- `R CMD build` checks for GNU `cp` rather than assuming Linux has it. (PR#17370 says 'Alpine Linux' does not.)
- Non-UTF-8 multibyte character handling fixed more permanently (PR#16732).
- `sum(<large ints>,<stuff>)` is more consistent. (PR#17372)
- `rf()` and `rbeta()` now also work correctly when `ncp` is not scalar, notably when (partly) `NA`. (PR#17375)
- `is.na(NULL)` no longer warns. (PR#16107)
- `R CMD INSTALL` now correctly sets C++ compiler flags when all source files are in sub-directories of 'src'.

CHANGES IN R 3.4.3

INSTALLATION on a UNIX-ALIKE

- A workaround has been added for the changes in location of time-zone files in macOS 10.13 'High Sierra' and again in 10.13.1, so the default time zone is deduced correctly from the system setting when R is configured with '`--with-internal-tzcode`' (the default on macOS).
- `R CMD javareconf` has been updated to recognize the use of a Java 9 SDK on macOS.

BUG FIXES

- `raw(0) & raw(0)` and `raw(0) | raw(0)` again return `raw(0)` (rather than `logical(0)`).
- `intToUtf8()` converts integers corresponding to surrogate code points to `NA` rather than invalid UTF-8, as well as values larger than the current Unicode maximum of `0x10FFFF`. (This aligns with the current RFC3629.)
- Fix calling of methods on S4 generics that dispatch on `...` when the call contains `....`
- Following Unicode 'Corrigendum 9', the UTF-8 representations of U+FFFE and U+FFFF are now regarded as valid by `utf8ToInt()`.
- `range(c(TRUE,NA),finite = TRUE)` and similar no longer return `NA`. (Reported by Lukas Stadler.)
- The self starting function `attr(SSlogis,"initial")` now also works when the y values have exact minimum zero and is slightly changed in general, behaving symmetrically in the y range.
- The printing of named raw vectors is now formatted nicely as for other such atomic vectors, thanks to Lukas Stadler.

CHANGES IN R 3.4.2

NEW FEATURES

- Setting the `LC_ALL` category in `Sys.setlocale()` invalidates any cached locale-specific day/month names and the AM/PM indicator for `strptime()` (as setting `LC_TIME` has since R 3.1.0).
- The version of LAPACK included in the sources has been updated to 3.7.1, a bug-fix release.
- The default for `tools::write_PACKAGES(rds_compress=)` has been changed to `"xz"` to match the compression used by CRAN.
- `c()` and `unlist()` are now more efficient in constructing the `names(.)` of their return value, thanks to a proposal by Suharto Anggono. (PR#17284)

UTILITIES

- `R CMD check` checks for and `R CMD build` corrects CRLF line endings in shell scripts `configure` and `cleanup` (even on Windows).

INSTALLATION on a UNIX-ALIKE

- The order of selection of OpenMP flags has been changed: Oracle Developer Studio 12.5 accepts '`-fopenmp`' and '`-xopenmp`' but only the latter enables OpenMP so it is now tried first.

BUG FIXES

- `within(List,rm(x1,x2))` works correctly again, including when `List[["x2"]]` is `NULL`.
- `regexec(pattern,text,*)` now applies `as.character(.)` to its first two arguments, as documented.
- `write.table()` and related functions, `writeLines()`, and perhaps other functions writing text to connections did not signal errors when the writes failed, e.g. due to a disk being full. Errors will now be signalled if detected during the write, warnings if detected when the connection is closed. (PR#17243)
- `rt()` assumed the `ncp` parameter was a scalar. (PR#17306)
- `menu(choices)` with more than 10 choices which easily fit into one `getOption("width")`-line no longer erroneously repeats choices. (PR#17312)
- `length()<-` on a pairlist succeeds. (`https://stat.ethz.ch/pipermail/r-devel/2017-July/074680.html`)
- Language objects such as `quote(("\n"))` or R functions are correctly printed again, where R 3.4.1 accidentally duplicated the backslashes.
- Construction of `names()` for very large objects in `c()` and `unlist()` now works, thanks to Suharto Anggono's patch proposals in PR#17292.
- Resource leaks (and similar) reported by Steve Grubb fixed. (PR#17314, PR#17316, PR#17317, PR#17318, PR#17319, PR#17320)
- `model.matrix(~1,mf)` now gets the row names from `mf` also when they differ from `1:nrow(mf)`, fixing PR#14992 thanks to the suggestion by Sebastian Meyer.

- `sigma(fm)` now takes the correct denominator degrees of freedom for a fitted model with `NA` coefficients. (PR#17313)
- `hist(x,"FD")` no longer "dies" with a somewhat cryptic error message when `x` has extreme outliers or `IQR()` zero: `nclass.FD(x)` tries harder to find a robust bin width h in the latter case, and `hist.default(*,breaks)` now checks and corrects a too large `breaks` number. (PR#17274)
- `callNextMethod()` works for `...` methods.
- `qr.coef(qd,y)` now has correct names also when `qd` is a complex QR or stems from `qr(*,LAPACK=TRUE)`.
- Setting `options(device = *)` to an invalid function no longer segfaults when plotting is initiated. (PR#15883)
- `encodeString(<very large string>)` no longer segfaults. (PR#15885)
- It is again possible to use `configure --enable-maintainer-mode` without having installed `notangle` (it was required in R 3.4.[01]).
- S4 method dispatch on `...` calls the method by name instead of `.Method` (for consistency with default dispatch), and only attempts to pass non-missing arguments from the generic.
- `readRDS(textConnection(.))` works again. (PR#17325)
- `(1:n)[-n]` no longer segfaults for `n <-2.2e9` (on a platform with enough RAM).
- `x <-1:2; tapply(x,list(x,x),function(x) "")[1,2]` now correctly returns `NA`. (PR#17333)
- Running of finalizers after explicit GC request moved from the R interface `do_gc` to the `C` interface `R_gc`. This helps with reclaiming inaccessible connections.
- `help.search(topic)` and `??topic` matching topics in vignettes with multiple file name extensions (e.g., '*.md.rsp' but not '*.Rmd') failed with an error when using `options(help_type = "html")`.
- The X11 device no longer uses the Xlib backing store (PR#16497).
- `array(character(),1)` now gives (a 1D array with) `NA` as has been documented for a long time as in the other cases of zero-length array initialization and also compatibly with `matrix(character(),*)`. As mentioned there, this also fixes PR#17333.
- `splineDesign(..,derivs = 4)` no longer segfaults.
- `fisher.test(*,hybrid=TRUE)` now (again) will use the hybrid method when Cochran's conditions are met, fixing PR#16654.

CHANGES IN R 3.4.1

INSTALLATION on a UNIX-ALIKE

- The deprecated support for PCRE versions older than 8.20 has been removed.

BUG FIXES

- `getParseData()` gave incorrect column information when code contained multi-byte characters. (PR#17254)
- Asking for help using expressions like `?stats::cor()` did not work. (PR#17250)
- `readRDS(url(....))` now works.
- `R CMD Sweave` again returns '`status = 0`' on successful completion.
- Vignettes listed in '.Rbuildignore' were not being ignored properly. (PR#17246)
- `file.mtime()` no longer returns NA on Windows when the file or directory is being used by another process. This affected `installed.packages()`, which is now protected against this.
- `R CMD INSTALL` Windows .zip file obeys `--lock` and `--pkglock` flags.
- (Windows only) The `choose.files()` function could return incorrect results when called with `multi = FALSE`. (PR#17270)
- `aggregate(<data.frame>,drop = FALSE)` now also works in case of near-equal numbers in by. (PR#16918)
- `fourfoldplot()` could encounter integer overflow when calculating the odds ratio. (PR#17286)
- `parse()` no longer gives spurious warnings when extracting srcrefs from a file not encoded in the current locale.

  This was seen from `R CMD check` with 'inst/doc/*.R' files, and `check` has some additional protection for such files.
- `print.noquote(x)` now always returns its argument `x` (invisibly).
- Non-UTF-8 multibyte character sets were not handled properly in source references. (PR#16732)

CHANGES IN R 3.4.0

SIGNIFICANT USER-VISIBLE CHANGES

- (Unix-alike) The default methods for `download.file()` and `url()` now choose `"libcurl"` except for '`file://`' URLs. There will be small changes in the format and wording of messages, including in rare cases if an issue is a warning or an error. For example, when HTTP re-direction occurs, some messages refer to the final URL rather than the specified one.

  Those who use proxies should check that their settings are compatible (see `?download.file`: the most commonly used forms work for both `"internal"` and `"libcurl"`).
- `table()` has been amended to be more internally consistent and become back compatible to R $\leq$ 2.7.2 again. Consequently, `table(1:2,exclude = NULL)` no longer contains a zero count for `<NA>`, but `useNA = "always"` continues to do so.
- `summary.default()` no longer rounds, but its print method does resulting in less extraneous rounding, notably of numbers in the ten thousands.
- `factor(x,exclude = L)` behaves more rationally when `x` or `L` are character vectors. Further, `exclude = <factor>` now behaves as documented for long.

- Arithmetic, logic (`&`, `|`) and comparison (aka 'relational', e.g., `<`, `==`) operations with arrays now behave consistently, notably for arrays of length zero.

  Arithmetic between length-1 arrays and longer non-arrays had silently dropped the array attributes and recycled. This now gives a warning and will signal an error in the future, as it has always for logic and comparison operations in these cases (e.g., compare `matrix(1,1) + 2:3` and `matrix(1,1) <2:3`).

- The JIT ('Just In Time') byte-code compiler is now enabled by default at its level 3. This means functions will be compiled on first or second use and top-level loops will be compiled and then run. (Thanks to Tomas Kalibera for extensive work to make this possible.)

  For now, the compiler will not compile code containing explicit calls to `browser()`: this is to support single stepping from the `browser()` call.

  JIT compilation can be disabled for the rest of the session using `compiler::enableJIT(0)` or by setting environment variable `R_ENABLE_JIT` to `0`.

- `xtabs()` works more consistently with `NA`s, also in its result no longer setting them to `0`. Further, a new logical option `addNA` allows to count `NA`s where appropriate. Additionally, for the case `sparse = TRUE`, the result's `dimnames` are identical to the default case's.

- Matrix products now consistently bypass BLAS when the inputs have `NaN/Inf` values. Performance of the check of inputs has been improved. Performance when BLAS is used is improved for matrix/vector and vector/matrix multiplication (DGEMV is now used instead of DGEMM).

  One can now choose from alternative matrix product implementations *via* `options(matprod = )`. The `"internal"` implementation is not optimized for speed but consistent in precision with other summations in R (using `long double` accumulators where available). `"blas"` calls BLAS directly for best speed, but usually with undefined behavior for inputs with `NaN/Inf`.

NEW FEATURES

- User errors such as `integrate(f,0:1,2)` are now caught.

- Add `signature` argument to `debug()`, `debugonce()`, `undebug()` and `isdebugged()` for more conveniently debugging S3 and S4 methods. (Based on a patch by Gabe Becker.)

- Add `utils::debugcall()` and `utils::undebugcall()` for debugging the function that would be called by evaluating the given expression. When the call is to an S4 generic or standard S3 generic, `debugcall()` debugs the method that would be dispatched. A number of internal utilities were added to support this, most notably `utils::isS3stdGeneric()`. (Based on a patch by Gabe Becker.)

- Add `utils::strcapture()`. Given a character vector and a regular expression containing capture expressions, `strcapture()` will extract the captured tokens into a tabular data structure, typically a `data.frame`.

- `str()` and `strOptions()` get a new option `drop.deparse.attr` with improved but *changed* default behaviour for expressions. For `expression` objects `x`, `str(x)` now may remove extraneous white space and truncate long lines.

- `str(<looooooooong_string>)` is no longer very slow; inspired by Mikko Korpela's proposal in PR#16527.

- `str(x)`'s default method is more "accurate" and hence somewhat more generous in displaying character vectors; this will occasionally change R outputs (and need changes to some '*.Rout(.save)' files).

For a classed integer vector such as `x <-xtabs(~ c(1,9,9,9))`, `str(x)` now shows both the class and `"int"`, instead of only the latter.

- `isSymmetric(m)` is much faster for large asymmetric matrices `m` *via* pre-tests and a new option `tol1` (with which strict back compatibility is possible but not the default).
- The result of `eigen()` now is of class `"eigen"` in the default case when eigenvectors are computed.
- Zero-length date and date-time objects (of classes `"POSIX[cl]?t"`) now `print()` "recognizably".
- `xy.coords()` and `xyz.coords()` get a new `setLab` option.
- The `method` argument of `sort.list()`, `order()` and `sort.int()` gains an `"auto"` option (the default) which should behave the same as before when `method` was not supplied.
- `stopifnot(E,..)` now reports differences when `E` is a call to `all.equal()` and that is not true.
- `boxplot(<formula>,*)` gain optional arguments `drop`, `sep`, and `lex.order` to pass to `split.default()` which itself gains an argument `lex.order` to pass to `interaction()` for more flexibility.
- The `plot()` method for `ppr()` has enhanced default labels (`xmin` and `main`).
- `sample.int()` gains an explicit `useHash` option (with a back compatible default).
- `identical()` gains an `ignore.srcref` option which drops `"srcref"` and similar attributes when true (as by default).
- `diag(x,nrow = n)` now preserves `typeof(x)`, also for logical, integer and raw `x` (and as previously for complex and numeric).
- `smooth.spline()` now allows direct specification of `lambda`, gets a `hatvalues()` method and keeps `tol` in the result, and optionally parts of the internal matrix computations.
- `addNA()` is faster now, e.g. when applied twice. (Part of PR#16895.)
- New option `rstandard(<lm>,type = "predicted")` provides the "PRESS"–related leave-one-out cross-validation errors for linear models.
- After seven years of deprecation, duplicated factor levels now produce a warning when printed and an error in `levels<-` instead of a warning.
- Invalid factors, e.g., with duplicated levels (invalid but constructable) now give a warning when printed, *via* new function `.valid.factor()`.
- `sessionInfo()` has been updated for Apple's change in OS naming as from '10.12' ('macOS Sierra' *vs* 'OS X El Capitan').

  Its `toLatex()` method now includes the `running` component.
- `options(interrupt=)` can be used to specify a default action for user interrupts. For now, if this option is not set and the `error` option is set, then an unhandled user interrupt invokes the `error` option. (This may be dropped in the future as `interrupt` conditions are not `error` conditions.)
- In most cases user interrupt handlers will be called with a `"resume"` restart available. Handlers can invoke this restart to resume computation. At the browser prompt the `r` command will invoke a `"resume"` restart if one is available. Some read operations cannot be resumed properly when interrupted and do not provide a `"resume"` restart.

- Radix sort is now chosen by `method = "auto"` for `sort.int()` for double vectors (and hence used for `sort()` for unclassed double vectors), excluding 'long' vectors.

  `sort.int(method = "radix")` no longer rounds double vectors.

- The `default` and `data.frame` methods for `stack()` preserve the names of empty elements in the levels of the `ind` column of the return value. Set the new `drop` argument to `TRUE` for the previous behavior.

- Speedup in `simplify2array()` and hence `sapply()` and `mapply()` (for the case of names and common length > 1), thanks to Suharto Anggono's PR#17118.

- `table(x,exclude = NULL)` now sets `useNA = "ifany"` (instead of `"always"`). Together with the bug fixes for this case, this recovers more consistent behaviour compatible to older versions of R. As a consequence, `summary()` for a logical vector no longer reports (zero) counts for `NA` when there are no `NA`s.

- `dump.frames()` gets a new option `include.GlobalEnv` which allows to also dump the global environment, thanks to Andreas Kersting's proposal in PR#17116.

- `system.time()` now uses `message()` instead of `cat()` when terminated early, such that `suppressMessages()` has an effect; suggested by Ben Bolker.

- `citation()` supports 'inst/CITATION' files from package source trees, with `lib.loc` pointing to the directory containing the package.

- `try()` gains a new argument `outFile` with a default that can be modified *via* `options(try.outFile = .)`, useful notably for `Sweave`.

- The unexported low-level functions in package **parallel** for passing serialized R objects to and from forked children now support long vectors on 64-bit platforms. This removes some limits on higher-level functions such as `mclapply()` (but returning gigabyte results from forked processes *via* serialization should be avoided if at all possible).

- Connections now `print()` without error even if invalid, e.g. after having been destroyed.

- `apropos()` and `find(simple.words = FALSE)` no longer match object names starting with '`.`' which are known to be internal objects (such as `.__S3MethodsTable__.`).

- Convenience function `hasName()` has been added; it is intended to replace the common idiom `!is.null(x$name)` without the usually unintended partial name matching.

- `strcapture()` no longer fixes column names nor coerces strings to factors (suggested by Bill Dunlap).

- `strcapture()` returns `NA` for non-matching values in `x` (suggested by Bill Dunlap).

- `source()` gets new optional arguments, notably `exprs`; this is made use of in the new utility function `withAutoprint()`.

- `sys.source()` gets a new `toplevel.env` argument. This argument is useful for frameworks running package tests; contributed by Tomas Kalibera.

- `Sys.setFileTime()` and `file.copy(copy.date = TRUE)` will set timestamps with fractions of seconds on platforms/filesystems which support this.

- (Windows only.) `file.info()` now returns file timestamps including fractions of seconds; it has done so on other platforms since R 2.14.0. (NB: some filesystems do not record modification and access timestamps to sub-second resolution.)

- The license check enabled by `options(checkPackageLicense = TRUE)` is now done when the package's namespace is first loaded.

- `ppr()` and `supsmu()` get an optional `trace` argument, and `ppr(..,sm.method = ..spline)` is no longer limited to sample size $n \leq 2500$.
- The `POSIXct` method for `print()` gets optional `tz` and `usetz` arguments, thanks to a report from Jennifer S. Lyon.
- New function `check_packages_in_dir_details()` in package **tools** for analyzing package-check log files to obtain check details.
- Package **tools** now exports function `CRAN_package_db()` for obtaining information about current packages in the CRAN package repository, and several functions for obtaining the check status of these packages.
- The (default) Stangle driver `Rtangle` allows `annotate` to be a function and gets a new `drop.evalFALSE` option.
- The default method for `quantile(x,prob)` should now be monotone in `prob`, even in border cases, see PR#16672.
- `bug.report()` now tries to extract an email address from a '`BugReports`' field, and if there is none, from a '`Contacts`' field.
- The `format()` and `print()` methods for `object.size()` results get new options `standard` and `digits`; notably, `standard = "IEC"` and `standard = "SI"` allow more standard (but less common) abbreviations than the default ones, e.g. for kilobytes. (From contributions by Henrik Bengtsson.)
- If a reference class has a validity method, `validObject` will be called automatically from the default initialization method for reference classes.
- `tapply()` gets new option `default = NA` allowing to change the previously hardcoded value.
- `read.dcf()` now consistently interprets any 'whitespace' to be stripped to include newlines.
- The maximum number of DLLs that can be loaded into R e.g. *via* `dyn.load()` can now be increased by setting the environment variable `R_MAX_NUM_DLLS` before starting R.
- Assigning to an element of a vector beyond the current length now over-allocates by a small fraction. The new vector is marked internally as growable, and the true length of the new vector is stored in the `truelength` field. This makes building up a vector result by assigning to the next element beyond the current length more efficient, though pre-allocating is still preferred. The implementation is subject to change and not intended to be used in packages at this time.
- Loading the **parallel** package namespace no longer sets or changes the `.Random.seed`, even if `R_PARALLEL_PORT` is unset.

  NB: This can break reproducibility of output, and did for a CRAN package.
- Methods `"wget"` and `"curl"` for `download.file()` now give an R error rather than a non-zero return value when the external command has a non-zero status.
- Encoding name `"utf8"` is mapped to `"UTF-8"`. Many implementations of `iconv` accept `"utf8"`, but not GNU **libiconv** (including the late 2016 version 1.15).
- `sessionInfo()` shows the full paths to the library or executable files providing the BLAS/LAPACK implementations currently in use (not available on Windows).
- The binning algorithm used by bandwidth selectors `bw.ucv()`, `bw.bcv()` and `bw.SJ()` switches to a version linear in the input size `n` for `n >nb/2`. (The calculations are the same, but for larger `n/nb` it is worth doing the binning in advance.)

- There is a new option `PCRE_study` which controls when `grep(perl = TRUE)` and friends 'study' the compiled pattern. Previously this was done for 11 or more input strings: it now defaults to 10 or more (but most examples need many more for the difference from studying to be noticeable).

- `grep(perl = TRUE)` and friends can now make use of PCRE's Just-In-Time mechanism, for PCRE $\geq$ 8.20 on platforms where JIT is supported. It is used by default whenever the `pattern` is studied (see the previous item). (Based on a patch from Mikko Korpela.)

  This is controlled by a new option `PCRE_use_JIT`.

  Note that in general this makes little difference to the speed, and may take a little longer: its benefits are most evident on strings of thousands of characters. As a side effect it reduces the chances of C stack overflow in the PCRE library on very long strings (millions of characters, but see next item).

  Warning: segfaults were seen using PCRE with JIT enabled on 64-bit Sparc builds.

- There is a new option `PCRE_limit_recursion` for `grep(perl = TRUE)` and friends to set a recursion limit taking into account R's estimate of the remaining C stack space (or 10000 if that is not available). This reduces the chance of C stack overflow, but because it is conservative may report a non-match (with a warning) in examples that matched before. By default it is enabled if any input string has 1000 or more bytes. (PR#16757)

- `getGraphicsEvent()` now works on `X11(type = "cairo")` devices. Thanks to Frederick Eaton (for reviving an earlier patch).

- There is a new argument `onIdle` for `getGraphicsEvent()`, which allows an R function to be run whenever there are no pending graphics events. This is currently only supported on X11 devices. Thanks to Frederick Eaton.

- The `deriv()` and similar functions now can compute derivatives of `log1p()`, `sinpi()` and similar one-argument functions, thanks to a contribution by Jerry Lewis.

- `median()` gains a formal `...` argument, so methods with extra arguments can be provided.

- `strwrap()` reduces `indent` if it is more than half `width` rather than giving an error. (Suggested by Bill Dunlap.)

- When the condition `code` in `if(.)` or `while(.)` is not of length one, an error instead of a warning may be triggered by setting an environment variable, see the help page.

- Formatting and printing of bibliography entries (`bibentry`) is more flexible and better documented. Apart from setting `options(citation.bibtex.max = 99)` you can also use `print(<citation>,bibtex=TRUE)` (or `format(..)`) to get the BibTeX entries in the case of more than one entry. This also affects `citation()`. Contributions to enable `style = "html+bibtex"` are welcome.

C-LEVEL FACILITIES

- Entry points `R_MakeExternalPtrFn` and `R_ExternalPtrFn` are now declared in header 'Rinternals.h' to facilitate creating and retrieving an R external pointer from a C function pointer without ISO C warnings about the conversion of function pointers.

- There was an exception for the native Solaris C++ compiler to the dropping (in R 3.3.0) of legacy C++ headers from headers such as 'R.h' and 'Rmath.h' — this has now been removed. That compiler has strict C++98 compliance hence does not include extensions in its (non-legacy) C++ headers: some packages will need to request C++11 or replace non-C++98 calls such as `lgamma`: see §1.6.4 of 'Writing R Extensions'.

  Because it is needed by about 70 CRAN packages, headers 'R.h' and 'Rmath.h' still declare

```
use namespace std;
```

when included on Solaris.

- When included from C++, the R headers now use forms such as `std::FILE` directly rather than including the line

  ```
  using std::FILE;
  ```

  C++ code including these headers might be relying on the latter.

- Headers 'R_ext/BLAS.h' and 'R_ext/Lapack.h' have many improved declarations including `const` for double-precision complex routines. *Inter alia* this avoids warnings when passing 'string literal' arguments from C++11 code.

- Headers for Unix-only facilities 'R_ext/GetX11Image.h', 'R_ext/QuartzDevice.h' and 'R_ext/eventloop.h' are no longer installed on Windows.

- No-longer-installed headers 'GraphicsBase.h', 'RGraphics.h', 'Rmodules/RX11.h' and 'Rmodules/Rlapack.h' which had a LGPL license no longer do so.

- `HAVE_UINTPTR_T` is now defined where appropriate by `Rconfig.h` so that it can be included before `Rinterface.h` when `CSTACK_DEFNS` is defined and a C compiler (not C++) is in use. `Rinterface.h` now includes C header 'stdint.h' or C++11 header 'cstdint' where needed.

- Package **tools** has a new function `package_native_routine_registration_skeleton()` to assist adding native-symbol registration to a package. See its help and §5.4.1 of 'Writing R Extensions' for how to use it. (At the time it was added it successfully automated adding registration to over 90% of CRAN packages which lacked it. Many of the failures were newly-detected bugs in the packages, e.g. 50 packages called entry points with varying numbers of arguments and 65 packages called entry points not in the package.)

INSTALLATION on a UNIX-ALIKE

- `readline` headers (and not just the library) are required unless configuring with '`--with-readline=no`'.

- `configure` now adds a compiler switch for C++11 code, even if the compiler supports C++11 by default. (This ensures that `g++` 6.x uses C++11 mode and not its default mode of C++14 with 'GNU extensions'.)

  The tests for C++11 compliance are now much more comprehensive. For gcc < 4.8, the tests from R 3.3.0 are used in order to maintain the same behaviour on Linux distributions with long-term support.

- An alternative compiler for C++11 is now specified with '`CXX11`', not '`CXX1X`'. Likewise C++11 flags are specified with '`CXX11FLAGS`' and the standard (e.g., '`-std=gnu++11`') is specified with '`CXX11STD`'.

- `configure` now tests for a C++14-compliant compiler by testing some basic features. This by default tries flags for the compiler specified by '`CXX11`', but an alternative compiler, options and standard can be specified by variables '`CXX14`', '`CXX14FLAGS`' and '`CXX14STD`' (e.g., '`-std=gnu++14`').

- There is a new macro `CXXSTD` to help specify the standard for C++ code, e.g. '`-std=c++98`'. This makes it easier to work with compilers which default to a later standard: for example, with `CXX=g++6 CXXSTD=-std=c++98` `configure` will select commands for `g++` 6.x which conform to C++11 and C++14 where specified but otherwise use C++98.

- Support for the defunct IRIX and OSF/1 OSes and Alpha CPU has been removed.
- `configure` checks that the compiler specified by '`$CXX $CXXFLAGS`' is able to compile C++ code.
- `configure` checks for the required header 'sys/select.h' (or 'sys/time.h' on legacy systems) and system call `select` and aborts if they are not found.
- If available, the POSIX 2008 system call `utimensat` will be used by `Sys.setFileTime()` and `file.copy(copy.date = TRUE)`. This may result in slightly more accurate file times. (It is available on Linux and FreeBSD but not macOS.)
- The minimum version requirement for `libcurl` has been reduced to 7.22.0, although at least 7.28.0 is preferred and earlier versions are little tested. (This is to support Debian 7 'Wheezy' LTS and Ubuntu 'Precise' 12.04 LTS, although the latter is close to end-of-life.)
- `configure` tests for a C++17-compliant compiler. The tests are experimental and subject to change in the future.

INCLUDED SOFTWARE

- (Windows only) Tcl/Tk version 8.6.4 is now included in the binary builds. The 'tcltk*.chm' help file is no longer included; please consult the online help at http://www.tcl.tk/man/ instead.
- The version of LAPACK included in the sources has been updated to 3.7.0: no new routines have been added to R.

PACKAGE INSTALLATION

- There is support for compiling C++14 or C++17 code in packages on suitable platforms: see 'Writing R Extensions' for how to request this.
- The order of flags when '`LinkingTo`' other packages has been changed so their include directories come earlier, before those specified in `CPPFLAGS`. This will only have an effect if non-system include directories are included with '`-I`' flags in `CPPFLAGS` (and so not the default `-I/usr/local/include` which is treated as a system include directory on most platforms).
- Packages which register native routines for `.C` or `.Fortran` need to be re-installed for this version (unless installed with R-devel SVN revision r72375 or later).
- Make variables with names containing `CXX1X` are deprecated in favour of those using `CXX11`, but for the time being are still made available *via* file 'etc/Makeconf'. Packages using them should be converted to the new forms and made dependent on '`R (>= 3.4.0)`'.

UTILITIES

- Running `R CMD check --as-cran` with `_R_CHECK_CRAN_INCOMING_REMOTE_` false now skips tests that require remote access. The remaining (local) tests typically run quickly compared to the remote tests.
- `R CMD build` will now give priority to vignettes produced from files in the 'vignettes' directory over those in the 'inst/doc' directory, with a warning that the latter are being ignored.
- `R CMD config` gains a '`--all`' option for printing names and values of all basic configure variables.

  It now knows about all the variables used for the C++98, C++11 and C++14 standards.

- `R CMD check` now checks that output files in 'inst/doc' are newer than the source files in 'vignettes'.
- For consistency with other package subdirectories, files named '*.r' in the 'tests' directory are now recognized as tests by `R CMD check`. (Wish of PR#17143.)
- `R CMD build` and `R CMD check` now use the *union* of `R_LIBS` and `.libPaths()`. They may not be equivalent, e.g., when the latter is determined by `R_PROFILE`.
- `R CMD build` now preserves dates when it copies files in preparing the tarball. (Previously on Windows it changed the dates on all files; on Unix, it changed some dates when installing vignettes.)
- The new option `R CMD check --no-stop-on-test-error` allows running the remaining tests (under 'tests/') even if one gave an error.
- Check customization *via* environment variables to detect side effects of `.Call()` and `.External()` calls which alter their arguments is described in §8 of the 'R Internals' manual.
- `R CMD check` now checks any '`BugReports`' field to be non-empty and a suitable single URL.
- `R CMD check --as-cran` now NOTEs if the package does not register its native routines or does not declare its intentions on (native) symbol search. (This will become a WARNING in due course.)

DEPRECATED AND DEFUNCT

- (Windows only) Function `setInternet2()` is defunct.
- Installation support for `readline` emulations based on `editline` (aka `libedit`) is deprecated.
- Use of the C/C++ macro '`NO_C_HEADERS`' is defunct and silently ignored.
- `unix.time()`, a traditional synonym for `system.time()`, has been deprecated.
- `structure(NULL,..)` is now deprecated as you cannot set attributes on `NULL`.
- Header 'Rconfig.h' no longer defines '`SUPPORT_OPENMP`'; instead use '`_OPENMP`' (as documented for a long time).
- (C-level Native routine registration.) The deprecated `styles` member of the `R_CMethodDef` and `R_FortranMethodDef` structures has been removed. Packages using these will need to be re-installed for R 3.4.0.
- The deprecated support for PCRE versions older than 8.20 will be removed in R 3.4.1. (Versions 8.20–8.31 will still be accepted but remain deprecated.)

BUG FIXES

- Getting or setting `body()` or `formals()` on non-functions for now signals a warning and may become an error for setting.
- `match(x,t)`, `duplicated(x)` and `unique(x)` work as documented for complex numbers with `NA`s or `NaN`s, where all those containing `NA` do match, whereas in the case of `NaN`'s both real and imaginary parts must match, compatibly with how `print()` and `format()` work for complex numbers.
- `deparse(<complex>,options = "digits17")` prints more nicely now, mostly thanks to a suggestion by Richie Cotton.

- Rotated symbols in plotmath expressions are now positioned correctly on `x11(type = "Xlib")`. (PR#16948)
- `as<-()` avoids an infinite loop when a virtual class is interposed between a subclass and an actual superclass.
- Fix level propagation in `unlist()` when the list contains zero-length lists or factors.
- Fix S3 dispatch on S4 objects when the **methods** package is not attached.
- Internal S4 dispatch sets `.Generic` in the method frame for consistency with `standardGeneric()`. (PR#16929)
- Fix `order(x,decreasing = TRUE)` when `x` is an integer vector containing `MAX_INT`. Ported from a fix Matt Dowle made to **data.table**.
- Fix caching by `callNextMethod()`, resolves PR#16973 and PR#16974.
- `grouping()` puts NAs last, to be consistent with the default behavior of `order()`.
- Point mass limit cases: `qpois(-2,0)` now gives `NaN` with a warning and `qgeom(1,1)` is `0`. (PR#16972)
- `table()` no longer drops an `"NaN"` factor level, and better obeys `exclude = <chr>`, thanks to Suharto Anggono's patch for PR#16936. Also, in the case of `exclude = NULL` and `NA`s, these are tabulated correctly (again).

  Further, `table(1:2,exclude = 1,useNA = "ifany")` no longer erroneously reports `<NA>` counts.

  Additionally, all cases of empty `exclude` are equivalent, and `useNA` is not overwritten when specified (as it was by `exclude = NULL`).
- `wilcox.test(x,conf.int=TRUE)` no longer errors out in cases where the confidence interval is not available, such as for `x = 0:2`.
- `droplevels(f)` now keeps <NA> levels when present.
- In integer arithmetic, `NULL` is now treated as `integer(0)` whereas it was previously treated as `double(0)`.
- The radix sort considers `NA_real_` and `NaN` to be equivalent in rank (like the other sort algorithms).
- When `index.return=TRUE` is passed to `sort.int()`, the radix sort treats `NA`s like `sort.list()` does (like the other sort algorithms).
- When in `tabulate(bin,nbin)` `length(bin)` is larger than the maximal integer, the result is now of type `double` and hence no longer silently overflows to wrong values. (PR#17140)
- `as.character.factor()` respects S4 inheritance when checking the type of its argument. (PR#17141)
- The `factor` method for `print()` no longer sets the class of the factor to `NULL`, which would violate a basic constraint of an S4 object.
- `formatC(x,flag = f)` allows two new flags, and signals an error for invalid flags also in the case of character formatting.
- Reading from `file("stdin")` now also closes the connection and hence no longer leaks memory when reading from a full pipe, thanks to Gábor Csárdi, see thread starting at `https://stat.ethz.ch/pipermail/r-devel/2016-November/073360.html`.

- Failure to create file in `tempdir()` for compressed `pdf()` graphics device no longer errors (then later segfaults). There is now a warning instead of error and compression is turned off for the device. Thanks to Alec Wysoker (PR#17191).
- Asking for `methods()` on `"|"` returns only S3 methods. See https://stat.ethz.ch/pipermail/r-devel/2016-December/073476.html.
- `dev.capture()` using Quartz Cocoa device (macOS) returned invalid components if the back-end chose to use ARGB instead of RGBA image format. (Reported by Noam Ross.)
- `seq("2","5")` now works too, equivalently to `"2":"5"` and `seq.int()`.
- `seq.int(to = 1,by = 1)` is now correct, other cases are integer (instead of `double`) when `seq()` is integer too, and the "non-finite" error messages are consistent between `seq.default()` and `seq.int()`, no longer mentioning `NaN` etc.
- `rep(x,times)` and `rep.int(x,times)` now work when `times` is larger than the largest value representable in an integer vector. (PR#16932)
- `download.file(method = "libcurl")` does not check for URL existence before attempting downloads; this is more robust to servers that do not support HEAD or range-based retrieval, but may create empty or incomplete files for aborted download requests.
- Bandwidth selectors `bw.ucv()`, `bw.bcv()` and `bw.SJ()` now avoid integer overflow for large sample sizes.
- `str()` no longer shows `"list output truncated"`, in cases that list was not shown at all. Thanks to Neal Fultz (PR#17219)
- Fix for `cairo_pdf()` (and `svg()` and `cairo_ps()`) when replaying a saved display list that contains a mix of **grid** and **graphics** output. (Report by Yihui Xie.)
- The `str()` and `as.hclust()` methods for `"dendrogram"` now also work for deeply nested dendrograms thanks to non-recursive implementations by Bradley Broom.
- `sample()` now uses two uniforms for added precision when the uniform generator is `Knuth-TAOCP`, `Knuth-TAOCP-2002`, or a user-defined generator and the population size is 2^{25} or greater.
- If a vignette in the 'vignettes' directory is listed in '.Rbuildignore', `R CMD build` would not include it in the tarball, but would include it in the vignette database, leading to a check warning. (PR#17246)
- `tools::latexToUtf8()` infinite looped on certain inputs. (PR#17138)
- `terms.formula()` ignored argument names when determining whether two terms were identical. (PR#17235)
- `callNextMethod()` was broken when called from a method that augments the formal arguments of a primitive generic.
- Coercion of an S4 object to a vector during sub-assignment into a vector failed to dispatch through the `as.vector()` generic (often leading to a segfault).
- Fix problems in command completion: Crash (PR#17222) and junk display in Windows, handling special characters in filenames on all systems.

CHANGES IN R 3.3.3

NEW FEATURES

- Changes when redirection of a 'http://' URL to a 'https://' URL is encountered:
  - The internal methods of `download.file()` and `url()` now report that they cannot follow this (rather than failing silently).
  - (Unix-alike) `download.file(method = "auto")` (the default) re-tries with `method = "libcurl"`.
  - (Unix-alike) `url(method = "default")` with an explicit open argument re-tries with `method = "libcurl"`. This covers many of the usages, e.g. `readLines()` with a URL argument.

INSTALLATION on a UNIX-ALIKE

- The `configure` check for the `zlib` version is now robust to versions longer than 5 characters, including `1.2.11`.

UTILITIES

- Environmental variable `_R_CHECK_TESTS_NLINES_` controls how `R CMD check` reports failing tests (see §8 of the 'R Internals' manual).

DEPRECATED AND DEFUNCT

- (C-level Native routine registration.) The undocumented `styles` field of the components of `R_CMethodDef` and `R_FortranMethodDef` is deprecated.

BUG FIXES

- `vapply(x,*)` now works with long vectors `x`. (PR#17174)
- `isS3method("is.na.data.frame")` and similar are correct now. (PR#17171)
- `grepRaw(<long>,<short>,fixed = TRUE)` now works, thanks to a patch by Mikko Korpela. (PR#17132)
- Package installation into a library where the package exists *via* symbolic link now should work wherever `Sys.readlink()` works, resolving PR#16725.
- `"Cincinnati"` was missing an `"n"` in the `precip` dataset.
- Fix buffer overflow vulnerability in `pdf()` when loading an encoding file. Reported by Talos (TALOS-2016-0227).
- `getDLLRegisteredRoutines()` now produces its warning correctly when multiple DLLs match, thanks to Matt Dowle's PR#17184.
- `Sys.timezone()` now returns non-NA also on platforms such as 'Ubuntu 14.04.5 LTS', thanks to Mikko Korpela's PR#17186.
- `format(x)` for an illegal `"POSIXlt"` object `x` no longer segfaults.
- `methods(f)` now also works for `f "("` or `"{"`.
- (Windows only) `dir.create()` did not check the length of the path to create, and so could overflow a buffer and crash R. (PR#17206)

- On some systems, very small hexadecimal numbers in hex notation would underflow to zero. (PR#17199)
- `pmin()` and `pmax()` now work again for `ordered` factors and 0-length S3 classed objects, thanks to Suharto Anggono's PR#17195 and PR#17200.
- `bug.report()` did not do any validity checking on a package's 'BugReports' field. It now ignores an empty field, removes leading whitespace and only attempts to open 'http://' and 'https://' URLs, falling back to emailing the maintainer.
- Bandwidth selectors `bw.ucv()` and `bw.SJ()` gave incorrect answers or incorrectly reported an error (because of integer overflow) for inputs longer than 46341. Similarly for `bw.bcv()` at length 5793.

  Another possible integer overflow is checked and may result in an error report (rather than an incorrect result) for much longer inputs (millions for a smooth distribution).
- `findMethod()` failed if the active signature had expanded beyond what a particular package used. (Example with packages **XR** and **XRJulia** on CRAN.)
- `qbeta()` underflowed too early in some very asymmetric cases. (PR#17178)
- `R CMD Rd2pdf` had problems with packages with non-ASCII titles in '.Rd' files (usually the titles were omitted).

CHANGES IN R 3.3.2

NEW FEATURES

- `extSoftVersion()` now reports the version (if any) of the `readline` library in use.
- The version of LAPACK included in the sources has been updated to 3.6.1, a bug-fix release including a speedup for the non-symmetric case of `eigen()`.
- Use `options(deparse.max.lines=)` to limit the number of lines recorded in `.Traceback` and other deparsing activities.
- `format(<AsIs>)` looks more regular, also for non-character atomic matrices.
- `abbreviate()` gains an option `named = TRUE`.
- The online documentation for package **methods** is extensively rewritten. The goals are to simplify documentation for basic use, to note old features not recommended and to correct out-of-date information.
- Calls to `setMethod()` no longer print a message when creating a generic function in those cases where that is natural: S3 generics and primitives.

INSTALLATION and INCLUDED SOFTWARE

- Versions of the `readline` library >= 6.3 had been changed so that terminal window resizes were not signalled to `readline`: code has been added using a explicit signal handler to work around that (when R is compiled against readline >= 6.3). (PR#16604)
- `configure` works better with Oracle Developer Studio 12.5.

UTILITIES

- `R CMD check` reports more dubious flags in files 'src/Makevars[.in]', including '`-w`' and '`-g`'.
- `R CMD check` has been set up to filter important warnings from recent versions of `gfortran` with '`-Wall -pedantic`': this now reports non-portable GNU extensions such as out-of-order declarations.
- `R CMD config` works better with paths containing spaces, even those of home directories (as reported by Ken Beath).

DEPRECATED AND DEFUNCT

- Use of the C/C++ macro '`NO_C_HEADERS`' is deprecated (no C headers are included by R headers from C++ as from R 3.3.0, so it should no longer be needed).

BUG FIXES

- The check for non-portable flags in `R CMD check` could be stymied by 'src/Makevars' files which contained targets.
- (Windows only) When using certain desktop themes in Windows 7 or higher, `Alt-Tab` could cause `Rterm` to stop accepting input. (PR#14406; patch submitted by Jan Gleixner.)
- `pretty(d,..)` behaves better for date-time `d` (PR#16923).
- When an S4 class name matches multiple classes in the S4 cache, perform a dynamic search in order to obey namespace imports. This should eliminate annoying messages about multiple hits in the class cache. Also, pass along the package from the `ClassExtends` object when looking up superclasses in the cache.
- `sample(NA_real_)` now works.
- Packages using non-ASCII encodings in their code did not install data properly on systems using different encodings.
- `merge(df1,df2)` now also works for data frames with column names `"na.last"`, `"decreasing"`, or `"method"`. (PR#17119)
- `contour()` caused a segfault if the `labels` argument had length zero. (Reported by Bill Dunlap.)
- `unique(warnings())` works more correctly, thanks to a new `duplicated.warnings()` method.
- `findInterval(x,vec = numeric(),all.inside = TRUE)` now returns `0`s as documented. (Reported by Bill Dunlap.)
- (Windows only) `R CMD SHLIB` failed when a symbol in the resulting library had the same name as a keyword in the '.def' file. (PR#17130)
- `pmax()` and `pmin()` now work with (more ?) classed objects, such as `"Matrix"` from the **Matrix** package, as documented for a long time.
- `axis(side,x = D)` and hence `Axis()` and `plot()` now work correctly for `"Date"` and time objects `D`, even when "time goes backward", e.g., with decreasing `xlim`. (Reported by William May.)
- `str(I(matrix(..)))` now looks as always intended.

- `plot.ts()`, the `plot()` method for time series, now respects `cex`, `lwd` and `lty`. (Reported by Greg Werbin.)
- `parallel::mccollect()` now returns a named list (as documented) when called with `wait = FALSE`. (Reported by Michel Lang.)
- If a package added a class to a class union in another package, loading the first package gave erroneous warnings about "undefined subclass".
- `c()`'s argument `use.names` is documented now, as belonging to the (C internal) default method. In "parallel", argument `recursive` is also moved from the generic to the default method, such that the formal argument list of **base** generic `c()` is just `(...)`.
- `rbeta(4,NA)` and similarly `rgamma()` and `rnbinom()` now return `NaN`'s with a warning, as other r<dist>(), and as documented. (PR#17155)
- Using `options(checkPackageLicense = TRUE)` no longer requires acceptance of the licence for non-default standard packages such as **compiler**. (Reported by Mikko Korpela.)
- `split(<very_long>,*)` now works even when the split off parts are long. (PR#17139)
- `min()` and `max()` now also work correctly when the argument list starts with `character(0)`. (PR#17160)
- Subsetting very large matrices (`prod(dim(.)) >= 2^31`) now works thanks to Michael Schubmehl's PR#17158.
- `bartlett.test()` used residual sums of squares instead of variances, when the argument was a list of `lm` objects. (Reported by Jens Ledet Jensen).
- `plot(<lm>,which = *)` now correctly labels the contour lines for the standardized residuals for `which = 6`. It also takes the correct p in case of singularities (also for `which = 5`). (PR#17161)
- `xtabs(~ exclude)` no longer fails from wrong scope, thanks to Suharto Anggono's PR#17147.
- Reference class calls to `methods()` did not re-analyse previously defined methods, meaning that calls to methods defined later would fail. (Reported by Charles Tilford).
- `findInterval(x,vec,left.open = TRUE)` misbehaved in some cases. (Reported by Dmitriy Chernykh.)

CHANGES IN R 3.3.1

BUG FIXES

- `R CMD INSTALL` and hence `install.packages()` gave an internal error installing a package called **description** from a tarball on a case-insensitive file system.
- `match(x,t)` (and hence `x %in% t`) failed when `x` was of length one, and either `character` and `x` and `t` only differed in their `Encoding` or when `x` and `t` where `complex` with `NA`s or `NaN`s. (PR#16885.)
- `unloadNamespace(ns)` also works again when `ns` is a 'namespace', as from `getNamespace()`.
- `rgamma(1,Inf)` or `rgamma(1,0,0)` no longer give `NaN` but the correct limit.
- `length(baseenv())` is correct now.

- `pretty(d,..)` for date-time `d` rarely failed when `"halfmonth"` time steps were tried (PR#16923) and on 'inaccurate' platforms such as 32-bit Windows or a configuration with `--disable-long-double`; see comment #15 of PR#16761.
- In `text.default(x,y,labels)`, the rarely(?) used default for `labels` is now correct also for the case of a 2-column matrix `x` and missing `y`.
- `as.factor(c(a = 1L))` preserves `names()` again as in R < 3.1.0.
- `strtrim(""[0],0[0])` now works.
- Use of `Ctrl-C` to terminate a reverse incremental search started by `Ctrl-R` in the `readline`-based Unix terminal interface is now supported when R was compiled against `readline` >= 6.0 (`Ctrl-G` always worked). (PR#16603)
- `diff(<difftime>)` now keeps the `"units"` attribute, as subtraction already did, PR#16940.

CHANGES IN R 3.3.0

SIGNIFICANT USER-VISIBLE CHANGES

- `nchar(x,*)`'s argument `keepNA` governing how the result for `NA`s in `x` is determined, gets a new default `keepNA = NA` which returns `NA` where `x` is `NA`, except for `type = "width"` which still returns `2`, the formatting / printing width of `NA`.
- All builds have support for '`https:`' URLs in the default methods for `download.file()`, `url()` and code making use of them.

  Unfortunately that cannot guarantee that any particular '`https:`' URL can be accessed. For example, server and client have to successfully negotiate a cryptographic protocol (TLS/SSL, ...) and the server's identity has to be verifiable *via* the available certificates. Different access methods may allow different protocols or use private certificate bundles: we encountered a '`https:`' CRAN mirror which could be accessed by one browser but not by another nor by `download.file()` on the same Linux machine.

NEW FEATURES

- The `print` method for `methods()` gains a `byclass` argument.
- New functions `validEnc()` and `validUTF8()` to give access to the validity checks for inputs used by `grep()` and friends.
- Experimental new functionality for S3 method checking, notably `isS3method()`.

  Also, the names of the R 'language elements' are exported as character vector `tools::langElts`.
- `str(x)` now displays `"Time-Series"` also for matrix (multivariate) time-series, i.e. when `is.ts(x)` is true.
- (Windows only) The GUI menu item to install local packages now accepts '*.tar.gz' files as well as '*.zip' files (but defaults to the latter).
- New programmeR's utility function `chkDots()`.
- `D()` now signals an error when given invalid input, rather than silently returning `NA`. (Request of John Nash.)
- `formula` objects are slightly more "first class": e.g., `formula()` or `new("formula",y ~ x)` are now valid. Similarly, for `"table"`, `"ordered"` and `"summary.table"`. Packages defining S4 classes with the above S3/S4 classes as slots should be reinstalled.
- New function `strrep()` for repeating the elements of a character vector.

- `rapply()` preserves attributes on the list when `how = "replace"`.
- New S3 generic function `sigma()` with methods for extracting the estimated standard deviation aka "residual standard deviation" from a fitted model.
- `news()` now displays R and package news files within the HTML help system if it is available. If no news file is found, a visible `NULL` is returned to the console.
- `as.raster(x)` now also accepts `raw` arrays `x` assuming values in `0:255`.
- Subscripting of matrix/array objects of type `"expression"` is now supported.
- `type.convert("i")` now returns a factor instead of a complex value with zero real part and missing imaginary part.
- Graphics devices `cairo_pdf()` and `cairo_ps()` now allow non-default values of the cairographics 'fallback resolution' to be set.

  This now defaults to 300 on all platforms: that is the default documented by cairographics, but apparently was not used by all system installations.
- `file()` gains an explicit `method` argument rather than implicitly using `getOption("url.method","default")`.
- Thanks to a patch from Tomas Kalibera, `x[x != 0]` is now typically faster than `x[which(x != 0)]` (in the case where `x` has no NAs, the two are equivalent).
- `read.table()` now always uses the names for a named `colClasses` argument (previously names were only used when `colClasses` was too short). (In part, wish of PR#16478.)
- (Windows only) `download.file()` with default `method = "auto"` and a 'ftps://' URL chooses `"libcurl"` if that is available.
- The out-of-the box Bioconductor mirror has been changed to one using 'https://': use `chooseBioCmirror()` to choose a 'http://' mirror if required.
- The data frame and formula methods for `aggregate()` gain a `drop` argument.
- `available.packages()` gains a `repos` argument.
- The undocumented switching of methods for `url()` on 'https:' and 'ftps:' URLs is confined to `method = "default"` (and documented).
- `smoothScatter()` gains a `ret.selection` argument.
- `qr()` no longer has a `...` argument to pass additional arguments to methods.
- `[` has a method for class `"table"`.
- It is now possible (again) to `replayPlot()` a display list snapshot that was created by `recordPlot()` in a different R session.

  It is still not a good idea to use snapshots as a persistent storage format for R plots, but it is now not completely silly to use a snapshot as a format for transferring an R plot between two R sessions.

  The underlying changes mean that packages providing graphics devices (e.g., **Cairo**, **RSvgDevice**, **cairoDevice**, **tikzDevice**) will need to be reinstalled.

  Code for restoring snapshots was contributed by Jeroen Ooms and JJ Allaire.

  Some testing code is available at https://github.com/pmur002/R-display-list.
- `tools::undoc(dir = D)` and `codoc(dir = D)` now also work when `D` is a directory whose `normalizePath()`ed version does not end in the package name, e.g. from a symlink.

- `abbreviate()` has more support for multi-byte character sets – it no longer removes bytes within characters and knows about Latin vowels with accents. It is still only really suitable for (most) European languages, and still warns on non-ASCII input.

  `abbreviate(use.classes = FALSE)` is now implemented, and that is more suitable for non-European languages.

- `match(x,table)` is faster (sometimes by an order of magnitude) when `x` is of length one and `incomparables` is unchanged, thanks to Peter Haverty (PR#16491).

- More consistent, partly not back-compatible behavior of `NA` and `NaN` coercion to complex numbers, operations less often resulting in complex `NA` (`NA_complex_`).

- `lengths()` considers methods for `length` and `[[` on `x`, so it should work automatically on any objects for which appropriate methods on those generics are defined.

- The logic for selecting the default screen device on OS X has been simplified: it is now `quartz()` if that is available even if environment variable `DISPLAY` has been set by the user.

  The choice can easily be overridden *via* environment variable `R_INTERACTIVE_DEVICE`.

- On Unix-like platforms which support the `getline` C library function, `system(*,intern = TRUE)` no longer truncates (output) lines longer than 8192 characters, thanks to Karl Millar. (PR#16544)

- `rank()` gains a `ties.method = "last"` option, for convenience (and symmetry).

- `regmatches(invert = NA)` can now be used to extract both non-matched and matched substrings.

- `data.frame()` gains argument `fix.empty.names`; `as.data.frame.list()` gets new `cut.names`, `col.names` and `fix.empty.names`.

- `plot(x ~ x,*)` now warns that it is the same as `plot(x ~ 1,*)`.

- `recordPlot()` has new arguments `load` and `attach` to allow package names to be stored as part of a recorded plot. `replayPlot()` has new argument `reloadPkgs` to load/attach any package names that were stored as part of a recorded plot.

- S4 dispatch works within calls to `.Internal()`. This means explicit S4 generics are no longer needed for `unlist()` and `as.vector()`.

- Only font family names starting with '`"Hershey"`' (and not '`"Her"`' as before) are given special treatment by the graphics engine.

- S4 values are automatically coerced to vector (*via* `as.vector`) when subassigned into atomic vectors.

- `findInterval()` gets a `left.open` option.

- The version of LAPACK included in the sources has been updated to 3.6.0, including those 'deprecated' routines which were previously included. *Ca* 40 double-complex routines have been added at the request of a package maintainer.

  As before, the details of what is included are in 'src/modules/lapack/README' and this now gives information on earlier additions.

- `tapply()` has been made considerably more efficient without changing functionality, thanks to proposals from Peter Haverty and Suharto Anggono. (PR#16640)

- `match.arg(arg)` (the one-argument case) is faster; so is `sort.int()`. (PR#16640)

- The `format` method for `object_size` objects now also accepts "binary" units such as `"KiB"` and e.g., `"Tb"`. (Partly from PR#16649.)

- Profiling now records calls of the form `foo::bar` and some similar cases directly rather than as calls to `<Anonymous>`. Contributed by Winston Chang.
- New string utilities `startsWith(x,prefix)` and `endsWith(x,suffix)`. Also provide speedups for some `grepl("^...",*)` uses (related to proposals in PR#16490).
- Reference class finalizers run at exit, as well as on garbage collection.
- Avoid **parallel** dependency on **stats** for port choice and random number seeds. (PR#16668)
- The radix sort algorithm and implementation from **data.table** (`forder`) replaces the previous radix (counting) sort and adds a new method for `order()`. Contributed by Matt Dowle and Arun Srinivasan, the new algorithm supports logical, integer (even with large values), real, and character vectors. It outperforms all other methods, but there are some caveats (see `?sort`).
- The `order()` function gains a `method` argument for choosing between `"shell"` and `"radix"`.
- New function `grouping()` returns a permutation that stably rearranges data so that identical values are adjacent. The return value includes extra partitioning information on the groups. The implementation came included with the new radix sort.
- `rhyper(nn,m,n,k)` no longer returns `NA` when one of the three parameters exceeds the maximal integer.
- `switch()` now warns when no alternatives are provided.
- `parallel::detectCores()` now has default `logical = TRUE` on all platforms – as this was the default on Windows, this change only affects Sparc Solaris.

  Option `logical = FALSE` is now supported on Linux and recent versions of OS X (for the latter, thanks to a suggestion of Kyaw Sint).
- `hist()` for `"Date"` or `"POSIXt"` objects would sometimes give misleading labels on the breaks, as they were set to the day before the start of the period being displayed. The display format has been changed, and the shift of the start day has been made conditional on `right = TRUE` (the default). (PR#16679)
- R now uses a new version of the logo (donated to the R Foundation by RStudio). It is defined in '.svg' format, so will resize without unnecessary degradation when displayed on HTML pages—there is also a vector PDF version. Thanks to Dirk Eddelbuettel for producing the corresponding X11 icon.
- New function `.traceback()` returns the stack trace which `traceback()` prints.
- `lengths()` dispatches internally.
- `dotchart()` gains a `pt.cex` argument to control the size of points separately from the size of plot labels. Thanks to Michael Friendly and Milan Bouchet-Valat for ideas and patches.
- `as.roman(ch)` now correctly deals with more diverse character vectors `ch`; also arithmetic with the resulting roman numbers works in more cases. (PR#16779)
- `prcomp()` gains a new option `rank.` allowing to directly aim for less than `min(n,p)` PC's. The `summary()` and its `print()` method have been amended, notably for this case.
- `gzcon()` gains a new option `text`, which marks the connection as text-oriented (so e.g. `pushBack()` works). It is still always opened in binary mode.

- The `import()` namespace directive now accepts an argument `except` which names symbols to exclude from the imports. The `except` expression should evaluate to a character vector (after substituting symbols for strings). See Writing R Extensions.
- New convenience function `Rcmd()` in package **tools** for invoking `R CMD` tools from within R.
- New functions `makevars_user()` and `makevars_site()` in package **tools** to determine the location of the user and site specific 'Makevars' files for customizing package compilation.

UTILITIES

- `R CMD check` has a new option '`--ignore-vignettes`' for use with non-Sweave vignettes whose '`VignetteBuilder`' package is not available.
- `R CMD check` now by default checks code usage (*via* **codetools**) with only the base package attached. Functions from default packages other than **base** which are used in the package code but not imported are reported as undefined globals, with a suggested addition to the `NAMESPACE` file.
- `R CMD check --as-cran` now also checks DOIs in package 'CITATION' and Rd files.
- `R CMD Rdconv` and `R CMD Rd2pdf` each have a new option '`--RdMacros=pkglist`' which allows Rd macros to be specified before processing.

DEPRECATED AND DEFUNCT

- The previously included versions of `zlib`, `bzip2`, `xz` and PCRE have been removed, so suitable external (usually system) versions are required (see the 'R Installation and Administration' manual).
- The unexported and undocumented Windows-only devices `cairo_bmp()`, `cairo_png()` and `cairo_tiff()` have been removed. (These devices should be used as e.g. `bmp(type = "cairo")`.)
- (Windows only) Function `setInternet2()` has no effect and will be removed in due course. The choice between methods `"internal"` and `"wininet"` is now made by the `method` arguments of `url()` and `download.file()` and their defaults can be set *via* options. The out-of-the-box default remains `"wininet"` (as it has been since R 3.2.2).
- `[<-` with an S4 value into a list currently embeds the S4 object into its own list such that the end result is roughly equivalent to using `[[<-`. That behavior is deprecated. In the future, the S4 value will be coerced to a list with `as.list()`.
- Package **tools**' functions `package.dependencies()`, `pkgDepends()`, etc are deprecated now, mostly in favor of `package_dependencies()` which is both more flexible and efficient.

INSTALLATION and INCLUDED SOFTWARE

- Support for very old versions of `valgrind` (e.g., 3.3.0) has been removed.
- The included `libtool` script (generated by `configure`) has been updated to version 2.4.6 (from 2.2.6a).
- `libcurl` version 7.28.0 or later with support for the `https` protocol is required for installation (except on Windows).
- BSD networking is now required (except on Windows) and so `capabilities("http/ftp")` is always true.

- `configure` uses `pkg-config` for PNG, TIFF and JPEG where this is available. This should work better with multiple installs and with those using static libraries.
- The minimum supported version of OS X is 10.6 ('Snow Leopard'): even that has been unsupported by Apple since 2012.
- The `configure` default on OS X is '`--disable-R-framework`': enable this if you intend to install under '/Library/Frameworks' and use with `R.app`.
- The minimum preferred version of PCRE has since R 3.0.0 been 8.32 (released in Nov 2012). Versions 8.10 to 8.31 are now deprecated (with warnings from `configure`), but will still be accepted until R 3.4.0.
- `configure` looks for C functions `__cospi`, `__sinpi` and `__tanpi` and uses these if `cospi` *etc* are not found. (OS X is the main instance.)
- (Windows) R is now built using `gcc` 4.9.3. This build will require recompilation of at least those packages that include C++ code, and possibly others. A build of R-devel using the older toolchain will be temporarily available for comparison purposes.

  During the transition, the environment variable `R_COMPILED_BY` has been defined to indicate which toolchain was used to compile R (and hence, which should be used to compile code in packages). The `COMPILED_BY` variable described below will be a permanent replacement for this.
- (Windows) A `make` and `R CMD config` variable named `COMPILED_BY` has been added. This indicates which toolchain was used to compile R (and hence, which should be used to compile code in packages).

PACKAGE INSTALLATION

- The `make` macro `AWK` which used to be made available to files such as 'src/Makefile' is no longer set.

C-LEVEL FACILITIES

- The API call `logspace_sum` introduced in R 3.2.0 is now remapped as an entry point to `Rf_logspace_sum`, and its first argument has gained a `const` qualifier. (PR#16470)

  Code using it will need to be reinstalled.

  Similarly, entry point `log1pexp` also defined in 'Rmath.h' is remapped there to `Rf_log1pexp`
- `R_GE_version` has been increased to 11.
- New API call `R_orderVector1`, a faster one-argument version of `R_orderVector`.
- When R headers such as 'R.h' and 'Rmath.h' are called from C++ code in packages they include the C++ versions of system headers such as '<cmath>' rather than the legacy headers such as '<math.h>'. (Headers 'Rinternals.h' and 'Rinterface.h' already did, and inclusion of system headers can still be circumvented by defining `NO_C_HEADERS`, including as from this version for those two headers.)

  The manual has long said that R headers should **not** be included within an `extern "C"` block, and almost all the packages affected by this change were doing so.
- Including header 'S.h' from C++ code would fail on some platforms, and so gives a compilation error on all.
- The deprecated header 'Rdefines.h' is now compatible with defining `R_NO_REMAP`.
- The connections interface now includes a function `R_GetConnection()` which allows packages implementing connections to convert R `connection` objects to `Rconnection` handles. Code which previously used the low-level R-internal `getConnection()` entry point should switch.

BUG FIXES

- C-level `asChar(x)` is fixed for when `x` is not a vector, and it returns `"TRUE"/"FALSE"` instead of `"T"/"F"` for logical vectors.
- The first arguments of `.colSums()` etc (with an initial dot) are now named `x` rather than `X` (matching `colSums()`): thus error messages are corrected.
- A `coef()` method for class `"maov"` has been added to allow `vcov()` to work with multivariate results. (PR#16380)
- `method = "libcurl"` connections signal errors rather than retrieving HTTP error pages (where the ISP reports the error).
- `xpdrows.data.frame()` was not checking for unique row names; in particular, this affected assignment to non-existing rows *via* numerical indexing. (PR#16570)
- `tail.matrix()` did not work for zero rows matrices, and could produce row "labels" such as `"[1e+05,]"`.
- Data frames with a column named `"stringsAsFactors"` now format and print correctly. (PR#16580)
- `cor()` is now guaranteed to return a value with absolute value less than or equal to 1. (PR#16638)
- Array subsetting now keeps `names(dim(.))`.
- Blocking socket connection selection recovers more gracefully on signal interrupts.
- The `data.frame` method of `rbind()` construction `row.names` works better in borderline integer cases, but may change the names assigned. (PR#16666)
- (X11 only) `getGraphicsEvent()` miscoded buttons and missed mouse motion events. (PR#16700)
- `methods(round)` now also lists `round.POSIXt`.
- `tar()` now works with the default `files = NULL`. (PR#16716)
- Jumps to outer contexts, for example in error recovery, now make intermediate jumps to contexts where `on.exit()` actions are established instead of trying to run all `on.exit()` actions before jumping to the final target. This unwinds the stack gradually, releases resources held on the stack, and significantly reduces the chance of a segfault when running out of C stack space. Error handlers established using `withCallingHandlers()` and `options("error")` specifications are ignored when handling a C stack overflow error as attempting one of these would trigger a cascade of C stack overflow errors. (These changes resolve PR#16753.)
- The spacing could be wrong when printing a complex array. (Report and patch by Lukas Stadler.)
- `pretty(d,n,min.n,*)` for date-time objects `d` works again in border cases with large `min.n`, returns a `labels` attribute also for small-range dates and in such cases its returned length is closer to the desired `n`. (PR#16761) Additionally, it finally does cover the range of `d`, as it always claimed.
- `tsp(x) <-NULL` did not handle correctly objects inheriting from both `"ts"` and `"mts"`. (PR#16769)
- `install.packages()` could give false errors when `options("pkgType")` was `"binary"`. (Reported by Jose Claudio Faria.)

- A bug fix in R 3.0.2 fixed problems with `locator()` in X11, but introduced problems in Windows. Now both should be fixed. (PR#15700)
- `download.file()` with `method = "wininet"` incorrectly warned of download file length difference when reported length was unknown. (PR#16805)
- `diag(NULL,1)` crashed because of missed type checking. (PR#16853)

CHANGES IN R 3.2.5

BUG FIXES

- `format.POSIXlt()` behaved incorrectly in R 3.2.4. E.g. the output of `format(as.POSIXlt(paste0(1940:2000,"-01-01"), tz = "CET"),usetz = TRUE)` ended in two `"CEST"` time formats.

CHANGES IN R 3.2.4

NEW FEATURES

- `install.packages()` and related functions now give a more informative warning when an attempt is made to install a base package.
- `summary(x)` now prints with less rounding when `x` contains infinite values. (Request of PR#16620.)
- `provideDimnames()` gets an optional `unique` argument.
- `shQuote()` gains `type = "cmd2"` for quoting in `cmd.exe` in Windows. (Response to PR#16636.)
- The `data.frame` method of `rbind()` gains an optional argument `stringsAsFactors` (instead of only depending on `getOption("stringsAsFactors")`).
- `smooth(x,*)` now also works for long vectors.
- `tools::texi2dvi()` has a workaround for problems with the `texi2dvi` script supplied by **texinfo 6.1**.

  It extracts more error messages from the LaTeX logs when in emulation mode.

UTILITIES

- `R CMD check` will leave a log file 'build_vignettes.log' from the re-building of vignettes in the '.Rcheck' directory if there is a problem, and always if environment variable `_R_CHECK_ALWAYS_LOG_VIGNETTE_OUTPUT_` is set to a true value.

DEPRECATED AND DEFUNCT

- Use of 'SUPPORT_OPENMP' from header 'Rconfig.h' is deprecated in favour of the standard OpenMP define '_OPENMP'.

  (This has been the recommendation in the manual for a while now.)
- The make macro `AWK` which is long unused by R itself but recorded in file 'etc/Makeconf' is deprecated and will be removed in R 3.3.0.
- The C header file 'S.h' is no longer documented: its use should be replaced by 'R.h'.

BUG FIXES

- `kmeans(x,centers = <1-row>)` now works. (PR#16623)
- `Vectorize()` now checks for clashes in argument names. (PR#16577)
- `file.copy(overwrite = FALSE)` would signal a successful copy when none had taken place. (PR#16576)
- `ngettext()` now uses the same default domain as `gettext()`. (PR#14605)
- `array(..,dimnames = *)` now warns about non-`list` dimnames and, from R 3.3.0, will signal the same error for invalid dimnames as `matrix()` has always done.
- `addmargins()` now adds dimnames for the extended margins in all cases, as always documented.
- `heatmap()` evaluated its `add.expr` argument in the wrong environment. (PR#16583)
- `require()` etc now give the correct entry of `lib.loc` in the warning about an old version of a package masking a newer required one.
- The internal deparser did not add parentheses when necessary, e.g. before `[]` or `[[]]`. (Reported by Lukas Stadler; additional fixes included as well).
- `as.data.frame.vector(*,row.names=*)` no longer produces 'corrupted' data frames from row names of incorrect length, but rather warns about them. This will become an error.
- `url` connections with `method = "libcurl"` are destroyed properly. (PR#16681)
- `withCallingHandler()` now (again) handles warnings even during S4 generic's argument evaluation. (PR#16111)
- `deparse(...,control = "quoteExpressions")` incorrectly quoted empty expressions. (PR#16686)
- `format()`ting datetime objects (`"POSIX[cl]?t"`) could segfault or recycle wrongly. (PR#16685)
- `plot.ts(<matrix>,las = 1)` now does use `las`.
- `saveRDS(*,compress = "gzip")` now works as documented. (PR#16653)
- (Windows only) The `Rgui` front end did not always initialize the console properly, and could cause R to crash. (PR#16698)
- `dummy.coef.lm()` now works in more cases, thanks to a proposal by Werner Stahel (PR#16665). In addition, it now works for multivariate linear models (`"mlm"`, `manova`) thanks to a proposal by Daniel Wollschlaeger.
- The `as.hclust()` method for `"dendrogram"`s failed often when there were ties in the heights.
- `reorder()` and `midcache.dendrogram()` now are non-recursive and hence applicable to somewhat deeply nested dendrograms, thanks to a proposal by Suharto Anggono in PR#16424.
- `cor.test()` now calculates very small p values more accurately (affecting the result only in extreme not statistically relevant cases). (PR#16704)
- `smooth(*,do.ends=TRUE)` did not always work correctly in R versions between 3.0.0 and 3.2.3.

- `pretty(D)` for date-time objects `D` now also works well if `range(D)` is (much) smaller than a second. In the case of only one unique value in `D`, the pretty range now is more symmetric around that value than previously.
  Similarly, `pretty(dt)` no longer returns a length 5 vector with duplicated entries for `Date` objects `dt` which span only a few days.

- The figures in help pages such as `?points` were accidentally damaged, and did not appear in R 3.2.3. (PR#16708)

- `available.packages()` sometimes deleted the wrong file when cleaning up temporary files. (PR#16712)

- The `X11()` device sometimes froze on Red Hat Enterprise Linux 6. It now waits for `MapNotify` events instead of `Expose` events, thanks to Siteshwar Vashisht. (PR#16497)

- `[dpqr]nbinom(*,size=Inf,mu=.)` now works as limit case, for 'dpq' as the Poisson. (PR#16727)
  `pnbinom()` no longer loops infinitely in border cases.

- `approxfun(*,method="constant")` and hence `ecdf()` which calls the former now correctly "predict" `NaN` values as `NaN`.

- `summary.data.frame()` now displays `NA`s in `Date` columns in all cases. (PR#16709)

CHANGES IN R 3.2.3

NEW FEATURES

- Some recently-added Windows time zone names have been added to the conversion table used to convert these to Olson names. (Including those relating to changes for Russia in Oct 2014, as in PR#16503.)

- (Windows) Compatibility information has been added to the manifests for 'Rgui.exe', 'Rterm.exe' and 'Rscript.exe'. This should allow `win.version()` and `Sys.info()` to report the actual Windows version up to Windows 10.

- Windows `"wininet"` FTP first tries EPSV / PASV mode rather than only using active mode (reported by Dan Tenenbaum).

- `which.min(x)` and `which.max(x)` may be much faster for logical and integer `x` and now also work for long vectors.

- The 'emulation' part of `tools::texi2dvi()` has been somewhat enhanced, including supporting `quiet = TRUE`. It can be selected by `texi2dvi = "emulation"`.

  (Windows) MiKTeX removed its `texi2dvi.exe` command in Sept 2015: `tools::texi2dvi()` tries `texify.exe` if it is not found.

- (Windows only) Shortcuts for printing and saving have been added to menus in `Rgui.exe`. (Request of PR#16572.)

- `loess(...,iterTrace=TRUE)` now provides diagnostics for robustness iterations, and the `print()` method for `summary(<loess>)` shows slightly more.

- The included version of PCRE has been updated to 8.38, a bug-fix release.

- `View()` now displays nested data frames in a more friendly way. (Request with patch in PR#15915.)

INSTALLATION and INCLUDED SOFTWARE

- The included configuration code for `libintl` has been updated to that from `gettext` version 0.19.5.1 — this should only affect how an external library is detected (and the only known instance is under OpenBSD). (Wish of PR#16464.)
- `configure` has a new argument '`--disable-java`' to disable the checks for Java.
- The `configure` default for `MAIN_LDFLAGS` has been changed for the FreeBSD, NetBSD and Hurd OSes to one more likely to work with compilers other than `gcc` (FreeBSD 10 defaults to `clang`).
- `configure` now supports the OpenMP flags '`-fopenmp=libomp`' (clang) and '`-qopenmp`' (Intel C).
- Various macros can be set to override the default behaviour of `configure` when detecting OpenMP: see file 'config.site'.
- Source installation on Windows has been modified to allow for MiKTeX installations without `texi2dvi.exe`. See file 'MkRules.dist'.

BUG FIXES

- `regexpr(pat,x,perl = TRUE)` with Python-style named capture did not work correctly when `x` contained `NA` strings. (PR#16484)
- The description of dataset `ToothGrowth` has been improved/corrected. (PR#15953)
- `model.tables(type = "means")` and hence `TukeyHSD()` now support `"aov"` fits without an intercept term. (PR#16437)
- `close()` now reports the status of a `pipe()` connection opened with an explicit `open` argument. (PR#16481)
- Coercing a list without names to a data frame is faster if the elements are very long. (PR#16467)
- (Unix-only) Under some rare circumstances piping the output from `Rscript` or `R -f` could result in attempting to close the input file twice, possibly crashing the process. (PR#16500)
- (Windows) `Sys.info()` was out of step with `win.version()` and did not report Windows 8.
- `topenv(baseenv())` returns `baseenv()` again as in R 3.1.0 and earlier. This also fixes `compilerJIT(3)` when used in '.Rprofile'.
- `detach()`ing the **methods** package keeps `.isMethodsDispatchOn()` true, as long as the methods namespace is not unloaded.
- Removed some spurious warnings from `configure` about the preprocessor not finding header files. (PR#15989)
- `rchisq(*,df=0,ncp=0)` now returns `0` instead of `NaN`, and `dchisq(*,df=0,ncp=*)` also no longer returns `NaN` in limit cases (where the limit is unique). (PR#16521)
- `pchisq(*,df=0,ncp >0,log.p=TRUE)` no longer underflows (for ncp > ~60).
- `nchar(x,"w")` returned -1 for characters it did not know about (e.g. zero-width spaces): it now assumes 1. It now knows about most zero-width characters and a few more double-width characters.
- Help for `which.min()` is now more precise about behavior with logical arguments. (PR#16532)

- The print width of character strings marked as "latin1" or "bytes" was in some cases computed incorrectly.
- abbreviate() did not give names to the return value if minlength was zero, unlike when it was positive.
- (Windows only) dir.create() did not always warn when it failed to create a directory. (PR#16537)
- When operating in a non-UTF-8 multibyte locale (e.g. an East Asian locale on Windows), grep() and related functions did not handle UTF-8 strings properly. (PR#16264)
- read.dcf() sometimes misread lines longer than 8191 characters. (Reported by Hervé Pagès with a patch.)
- within(df,..) no longer drops columns whose name start with a ".".
- The built-in HTTP server converted entire Content-Type to lowercase including parameters which can cause issues for multi-part form boundaries (PR#16541).
- Modifying slots of S4 objects could fail when the **methods** package was not attached. (PR#16545)
- splineDesign(*,outer.ok=TRUE) (**splines**) is better now (PR#16549), and interpSpline() now allows sparse=TRUE for speedup with non-small sizes.
- If the expression in the traceback was too long, traceback() did not report the source line number. (Patch by Kirill Müller.)
- The browser did not truncate the display of the function when exiting with options("deparse.max.lines") set. (PR#16581)
- When bs(*,Boundary.knots=) had boundary knots inside the data range, extrapolation was somewhat off. (Patch by Trevor Hastie.)
- var() and hence sd() warn about factor arguments which are deprecated now. (PR#16564)
- loess(*,weights = *) stored wrong weights and hence gave slightly wrong predictions for newdata. (PR#16587)
- aperm(a,*) now preserves names(dim(a)).
- poly(x,..) now works when either raw=TRUE or coef is specified. (PR#16597)
- data(package=*) is more careful in determining the path.
- prettyNum(*,decimal.mark,big.mark): fixed bug introduced when fixing PR#16411.

CHANGES IN R 3.2.2

SIGNIFICANT USER-VISIBLE CHANGES

- It is now easier to use secure downloads from 'https://' URLs on builds which support them: no longer do non-default options need to be selected to do so. In particular, packages can be installed from repositories which offer 'https://' URLs, and those listed by setRepositories() now do so (for some of their mirrors).

  Support for 'https://' URLs is available on Windows, and on other platforms if support for libcurl was compiled in and if that supports the https protocol (system installations can be expected to do). So 'https://' support can be expected except on rather old OSes (an example being OS X 'Snow Leopard', where a non-system version of libcurl can be used).

(Windows only) The default method for accessing URLs *via* `download.file()` and `url()` has been changed to be `"wininet"` using Windows API calls. This changes the way proxies need to be set and security settings made: there have been some reports of 'ftp:' sites being inaccessible under the new default method (but the previous methods remain available).

NEW FEATURES

- `cmdscale()` gets new option `list.` for increased flexibility when a list should be returned.
- `configure` now supports `texinfo` version 6.0, which (unlike the change from 4.x to 5.0) is a minor update. (Wish of PR#16456.)
- (Non-Windows only) `download.file()` with default `method = "auto"` now chooses `"libcurl"` if that is available and a 'https://' or 'ftps://' URL is used.
- (Windows only) `setInternet2(TRUE)` is now the default. The command-line option `--internet2` and environment variable `R_WIN_INTERNET2` are now ignored.

  Thus by default the `"internal"` method for `download.file()` and `url()` uses the `"wininet"` method: to revert to the previous default use `setInternet2(FALSE)`.

  This means that 'https://' URLs can be read by default by `download.file()` (they have been readable by `file()` and `url()` since R 3.2.0).

  There are implications for how proxies need to be set (see `?download.file`).
- `chooseCRANmirror()` and `chooseBioCmirror()` now offer HTTPS mirrors in preference to HTTP mirrors. This changes the interpretation of their `ind` arguments: see their help pages.
- `capture.output()` gets optional arguments `type` and `split` to pass to `sink()`, and hence can be used to capture messages.

C-LEVEL FACILITIES

- Header 'Rconfig.h' now defines `HAVE_ALLOCA_H` if the platform has the 'alloca.h' header (it is needed to define `alloca` on Solaris and AIX, at least: see 'Writing R Extensions' for how to use it).

INSTALLATION and INCLUDED SOFTWARE

- The `libtool` script generated by `configure` has been modified to support FreeBSD >= 10 (PR#16410).

BUG FIXES

- The HTML help page links to demo code failed due to a change in R 3.2.0. (PR#16432)
- If the `na.action` argument was used in `model.frame()`, the original data could be modified. (PR#16436)
- `getGraphicsEvent()` could cause a crash if a graphics window was closed while it was in use. (PR#16438)
- `matrix(x,nr,nc,byrow = TRUE)` failed if `x` was an object of type `"expression"`.
- `strptime()` could overflow the allocated storage on the C stack when the timezone had a non-standard format much longer than the standard formats. (Part of PR#16328.)

- `options(OutDec = s)` now signals a warning (which will become an error in the future) when `s` is not a string with exactly one character, as that has been a documented requirement.
- `prettyNum()` gains a new option `input.d.mark` which together with other changes, e.g., the default for `decimal.mark`, fixes some `format()`ting variants with non-default `getOption("OutDec")` such as in PR#16411.
- `download.packages()` failed for `type` equal to either `"both"` or `"binary"`. (Reported by Dan Tenenbaum.)
- The `dendrogram` method of `labels()` is much more efficient for large dendrograms, now using `rapply()`. (Comment #15 of PR#15215)
- The `"port"` algorithm of `nls()` could give spurious errors. (Reported by Radford Neal.)
- Reference classes that inherited from reference classes in another package could invalidate methods of the inherited class. Fixing this requires adding the ability for methods to be "external", with the object supplied explicitly as the first argument, named `.self`. See "Inter-Package Superclasses" in the documentation.
- `readBin()` could fail on the SPARC architecture due to alignment issues. (Reported by Radford Neal.)
- `qt(*,df=Inf,ncp=.)` now uses the natural `qnorm()` limit instead of returning `NaN`. (PR#16475)
- Auto-printing of S3 and S4 values now searches for `print()` in the base namespace and `show()` in the **methods** namespace instead of searching the global environment.
- `polym()` gains a `coefs = NULL` argument and returns class `"poly"` just like `poly()` which gets a new `simple=FALSE` option. They now lead to correct `predict()`ions, e.g., on subsets of the original data. (PR#16239)
- `rhyper(nn,<large>)` now works correctly. (PR#16489)
- `ttkimage()` did not (and could not) work so was removed. Ditto for `tkimage.cget()` and `tkimage.configure()`. Added two Ttk widgets and missing subcommands for Tk's `image` command: `ttkscale()`, `ttkspinbox()`, `tkimage.delete()`, `tkimage.height()`, `tkimage.inuse()`, `tkimage.type()`, `tkimage.types()`, `tkimage.width()`. (PR#15372, PR#16450)
- `getClass("foo")` now also returns a class definition when it is found in the cache more than once.

CHANGES IN R 3.2.1

NEW FEATURES

- `utf8ToInt()` now checks that its input is valid UTF-8 and returns `NA` if it is not.
- `install.packages()` now allows `type = "both"` with `repos = NULL` if it can infer the type of file.
- `nchar(x,*)` and `nzchar(x)` gain a new argument `keepNA` which governs how the result for `NA`s in `x` is determined. For `nzchar()` in general and `nchar()` in the R 3.2.x series, the default remains `FALSE` which is fully back compatible. From R 3.3.0, `nchar()`'s default will change to `keepNA = NA` and you are advised to consider this for code portability.
- `news()` more flexibly extracts dates from package 'NEWS.Rd' files.

- `lengths(x)` now also works (trivially) for atomic `x` and hence can be used more generally as an efficient replacement of `sapply(x,length)` and similar.
- The included version of PCRE has been updated to 8.37, a bug-fix release.
- `diag()` no longer duplicates a matrix when extracting its diagonal.
- `as.character.srcref()` gains an argument to allow characters corresponding to a range of source references to be extracted.

BUG FIXES

- `acf()` and `ccf()` now guarantee values strictly in $[-1, 1]$ (instead of sometimes very slightly outside). PR#15832.
- `as.integer("11111111111")` now gives NA (with a warning) as it does for the corresponding numeric or negative number coercions. Further, `as.integer(M + 0.1)` now gives `M` (instead of NA) when M is the maximal representable integer.
- On some platforms `nchar(x,"c")` and `nchar(x,"w")` would return values (possibly `NA`) for inputs which were declared to be UTF-8 but were not, or for invalid strings without a marked encoding in a multi-byte locale, rather than give an error. Additional checks have been added to mitigate this.
- `apply(a,M,function(u) c(X = .,Y = .))` again has dimnames containing "X" and "Y" (as in R < 3.2.0).
- (Windows only) In some cases, the `--clean` option to `R CMD INSTALL` could fail. (PR#16178)
- (Windows only) `choose.files()` would occasionally include characters from the result of an earlier call in the result of a later one. (PR#16270)
- A change in `RSiteSearch()` in R 3.2.0 caused it to submit invalid URLs. (PR#16329)
- `Rscript` and command line `R` silently ignored incomplete statements at the end of a script; now they are reported as parse errors. (PR#16350)
- Parse data for very long strings was not stored. (PR#16354)
- `plotNode()`, the workhorse of the `plot` method for `"dendrogram"`s is no longer recursive, thanks to Suharto Anggono, and hence also works for deeply nested dendrograms. (PR#15215)
- The parser could overflow internally when given numbers in scientific format with extremely large exponents. (PR#16358)
- If the CRAN mirror was not set, `install.packages(type = "both")` and related functions could repeatedly query the user for it. (Part of PR#16362)
- The low-level functions `.rowSums()` etc. did not check the length of their argument, so could segfault. (PR#16367)
- The `quietly` argument of `library()` is now correctly propagated from `.getRequiredPackages2()`.
- Under some circumstances using the internal PCRE when building R from source would cause external libs such as `-llzma` to be omitted from the main link.
- The .Primitive default methods of the logic operators, i.e., `!`, `&` and `|`, now give correct error messages when appropriate, e.g., for `` `&`(TRUE) `` or `` `!`() ``. (PR#16385)
- `cummax(x)` now correctly propagates `NA`s also when `x` is of type `integer` and begins with an `NA`.

- `summaryRprof()` could fail when the profile contained only two records. (PR#16395)
- HTML vignettes opened using `vignette()` did not support links into the rest of the HTML help system. (Links worked properly when the vignette was opened using `browseVignettes()` or from within the help system.)
- `arima(*,xreg = .)` (for $d \geq 1$) computes estimated variances based on a the number of effective observations as in R version 3.0.1 and earlier. (PR#16278)
- `slotNames(.)` is now correct for `"signature"` objects (mostly used internally in **methods**).
- On some systems, the first string comparison after a locale change would result in `NA`.

CHANGES IN R 3.2.0

NEW FEATURES

- `anyNA()` gains a `recursive` argument.
- When `x` is missing and `names` is not false (including the default value), `Sys.getenv(x,names)` returns an object of class `"Dlist"` and hence prints tidily.
- (Windows.) `shell()` no longer consults the environment variable `SHELL`: too many systems have been encountered where it was set incorrectly (usually to a path where software was compiled, not where it was installed). `R_SHELL`, the preferred way to select a non-default shell, can be used instead.
- Some unusual arguments to `embedFonts()` can now be specified as character vectors, and the defaults have been changed accordingly.
- Functions in the `Summary` group duplicate less. (PR#15798)
- (Unix-alikes.) `system(cmd,input = )` now uses 'shell-execution-environment' redirection, which will be more natural if `cmd` is not a single command (but requires a POSIX-compliant shell). (Wish of PR#15508)
- `read.fwf()` and `read.DIF()` gain a `fileEncoding` argument, for convenience.
- Graphics devices can add attributes to their description in `.Device` and `.Devices`. Several of those included with R use a `"filepath"` attribute.
- `pmatch()` uses hashing in more cases and so is faster at the expense of using more memory. (PR#15697)
- `pairs()` gains new arguments to select sets of variables to be plotted against each other.
- `file.info(,extra_cols = FALSE)` allows a minimal set of columns to be computed on Unix-alikes: on some systems without properly-configured caching this can be significantly faster with large file lists.
- New function `dir.exists()` in package **base** to test efficiently whether one or more paths exist and are directories.
- `dput()` and friends gain new controls 'hexNumeric' and 'digits17' which output double and complex quantities as, respectively, binary fractions (exactly, see `sprintf("%a")`) and as decimals with up to 17 significant digits.
- `save()`, `saveRDS()` and `serialize()` now support `ascii = NA` which writes ASCII files using `sprintf("%a")` for double/complex quantities. This is read-compatible with `ascii = TRUE` but avoids binary->decimal->binary conversions with potential loss of precision. Unfortunately the Windows C runtime's lack of C99 compliance means that the format cannot be read correctly there in R before 3.1.2.

- The default for `formatC(decimal.mark =)` has been changed to be `getOption("OutDec")`; this makes it more consistent with `format()` and suitable for use in print methods, e.g. those for classes `"density"`, `"ecdf"`, `"stepfun"` and `"summary.lm"`.

  `getOption("OutDec")` is now consulted by the print method for class `"kmeans"`, by `cut()`, `dendrogram()`, `plot.ts()` and `quantile()` when constructing labels and for the report from `legend(trace = TRUE)`.

  (In part, wish of PR#15819.)

- `printNum()` and hence `format()` and `formatC()` give a warning if `big.mark` and `decimal.mark` are set to the same value (period and comma are not uncommonly used for each, and this is a check that conventions have not got mixed).

- `merge()` can create a result which uses long vectors on 64-bit platforms.

- `dget()` gains a new argument `keep.source` which defaults to `FALSE` for speed (`dput()` and `dget()` are most often used for data objects where this can make `dget()` many times faster).

- Packages may now use a file of common macro definitions in their help files, and may import definitions from other packages.

- A number of macros have been added in the new 'share/Rd' directory for use in package overview help pages, and `promptPackage()` now makes use of them.

- `tools::parse_Rd()` gains a new `permissive` argument which converts unrecognized macros into text. This is used by `utils:::format.bibentry` to allow LaTeX markup to be ignored.

- `options(OutDec =)` can now specify a multi-byte character, e.g., `options(OutDec = "\u00b7")` in a UTF-8 locale.

- `is.recursive(x)` is no longer true when `x` is an external pointer, a weak reference or byte code; the first enables `all.equal(x,x)` when `x <-getClass(.)`.

- `ls()` (aka `objects()`) and `as.list.environment()` gain a new argument `sorted`.

- The `"source"` attribute (which has not been added to functions by R since before R version 2.14.0) is no longer treated as special.

- Function `returnValue()` has been added to give `on.exit()` code access to a function's return value for debugging purposes.

- `crossprod(x,y)` allows more matrix coercions when `x` or `y` are vectors, now equalling `t(x) %*% y` in these cases (also reported by Radford Neal). Similarly, `tcrossprod(x,y)` and `%*%` work in more cases with vector arguments.

- Utility function `dynGet()` useful for detecting cycles, aka infinite recursions.

- The byte-code compiler and interpreter include new instructions that allow many scalar subsetting and assignment and scalar arithmetic operations to be handled more efficiently. This can result in significant performance improvements in scalar numerical code.

- `apply(m,2,identity)` is now the same as the matrix `m` when it has *named* row names.

- A new function `debuggingState()` has been added, allowing to temporarily turn off debugging.

- `example()` gets a new optional argument `run.donttest` and `tools::Rd2ex()` a corresponding `commentDonttest`, with a default such that `example(..)` in help examples will run `\donttest` code only if used interactively (a change in behaviour).

- `rbind.data.frame()` gains an optional argument `make.row.names`, for potential speedup.

- New function `extSoftVersion()` to report on the versions of third-party software in use in this session. Currently reports versions of `zlib`, `bzlib`, the `liblzma` from `xz`, PCRE, ICU, TRE and the `iconv` implementation.

  A similar function `grSoftVersion()` in package **grDevices** reports on third-party graphics software.

  Function `tcltk::tclVersion()` reports the Tcl/Tk version.

- Calling `callGeneric()` without arguments now works with primitive generics to some extent.

- `vapply(x,FUN,FUN.VALUE)` is more efficient notably for large `length(FUN.VALUE)`; as extension of PR#16061.

- `as.table()` now allows tables with one or more dimensions of length 0 (such as `as.table(integer())`).

- `names(x) <-NULL` now clears the names of call and `...` objects.

- `library()` will report a warning when an insufficient dependency version is masking a sufficient one later on the library search path.

- A new `plot()` method for class `"raster"` has been added.

- New `check_packages_in_dir_changes()` function in package **tools** for conveniently analyzing how changing sources impacts the check results of their reverse dependencies.

- Speed-up from Peter Haverty for `ls()` and `methods:::.requirePackage()` speeding up package loading. (PR#16133)

- New `get0()` function, combining `exists()` and `get()` in one call, for efficiency.

- `match.call()` gains an `envir` argument for specifying the environment from which to retrieve the `...` in the call, if any; this environment was wrong (or at least undesirable) when the `definition` argument was a function.

- `topenv()` has been made `.Internal()` for speedup, based on Peter Haverty's proposal in PR#16140.

- `getOption()` no longer calls `options()` in the main case.

- Optional use of `libcurl` (version 7.28.0 from Oct 2012 or later) for Internet access:

  - `capabilities("libcurl")` reports if this is available.
  - `libcurlVersion()` reports the version in use, and other details of the `"libcurl"` build including which URL schemes it supports.
  - `curlGetHeaders()` retrieves the headers for '`http://`', '`https://`', '`ftp://`' and '`ftps://`' URLs: analysis of these headers can provide insights into the 'existence' of a URL (it might for example be permanently redirected) and is so used in `R CMD check --as-cran`.
  - `download.file()` has a new optional method `"libcurl"` which will handle more URL schemes, follow redirections, and allows simultaneous downloads of multiple URLs.
  - `url()` has a new method `"libcurl"` which handles more URL schemes and follows redirections. The default method is controlled by a new option `url.method`, which applies also to the opening of URLs *via* `file()` (which happens implicitly in functions such as `read.table`.)
  - When `file()` or `url()` is invoked with a `https://` or `ftps://` URL which the current method cannot handle, it switches to a suitable method if one is available.

- (Windows.) The DLLs 'internet.dll' and 'internet2.dll' have been merged. In this version it is safe to switch (repeatedly) between the internal and Windows internet functions within an R session.

  The Windows internet functions are still selected by flag '`--internet2`' or `setInternet2()`. This can be overridden for an `url()` connection *via* its new `method` argument.

  `download.file()` has new method `"wininet"`, selected as the default by '`--internet2`' or `setInternet2()`.

- `parent.env<-` can no longer modify the parent of a locked namespace or namespace imports environment. Contributed by Karl Millar.

- New function `isNamespaceLoaded()` for readability and speed.

- `names(env)` now returns all the object names of an `environment` `env`, equivalently to `ls(env,all.names = TRUE,sorted = FALSE)` and also to the names of the corresponding list, `names(as.list(env,all.names = TRUE))`. Note that although `names()` returns a character vector, the names have no particular ordering.

- The memory manager now grows the heap more aggressively. This reduces the number of garbage collections, in particular while data or code are loaded, at the expense of slightly increasing the memory footprint.

- New function `trimws()` for removing leading/trailing whitespace.

- `cbind()` and `rbind()` now consider S4 inheritance during S3 dispatch and also obey `deparse.level`.

- `cbind()` and `rbind()` will delegate recursively to `methods::cbind2` (`methods::rbind2`) when at least one argument is an S4 object and S3 dispatch fails (due to ambiguity).

- (Windows.) `download.file(quiet = FALSE)` now uses text rather than Windows progress bars in non-interactive use.

- New function `hsearch_db()` in package **utils** for building and retrieving the help search database used by `help.search()`, along with functions for inspecting the concepts and keywords in the help search database.

- New function `.getNamespaceInfo()`, a no-check version of `getNamespaceInfo()` mostly for internal speedups.

- The help search system now takes '`\keyword`' entries in Rd files which are not standard keywords (as given in 'KEYWORDS' in the R documentation directory) as concepts. For standard keyword entries the corresponding descriptions are additionally taken as concepts.

- New `lengths()` function for getting the lengths of all elements in a list.

- New function `toTitleCase()` in package **tools**, tailored to package titles.

- The matrix methods of `cbind()` and `rbind()` allow matrices as inputs which have 2^{31} or more elements. (For `cbind()`, wish of PR#16198.)

- The default method of `image()` has an explicit check for a numeric or logical matrix (which was always required).

- `URLencode()` will not by default encode further URLs which appear to be already encoded.

- `BIC(mod)` and `BIC(mod,mod2)` now give non-NA numbers for `arima()` fitted models, as `nobs(mod)` now gives the number of "used" observations for such models. This fixes PR#16198, quite differently than proposed there.

- The `print()` methods for `"htest"`, `"pairwise.htest"` and `"power.htest"` objects now have a `digits` argument defaulting to (a function of) `getOption("digits")`, and influencing all printed numbers coherently. Unavoidably, this changes the display of such test results in some cases.
- Code completion for namespaces now recognizes all loaded namespaces, rather than only the ones that are also attached.
- The code completion mechanism can now be replaced by a user-specified completer function, for (temporary) situations where the usual code completion is inappropriate.
- `unzip()` will now warn if it is able to detect truncation when unpacking a file of 4GB or more (related to PR#16243).
- `methods()` reports S4 in addition to S3 methods; output is simplified when the `class` argument is used. `.S3methods()` and `methods::.S4methods()` report S3 and S4 methods separately.
- Higher order functions such as the `apply` functions and `Reduce()` now force arguments to the functions they apply in order to eliminate undesirable interactions between lazy evaluation and variable capture in closures. This resolves PR#16093.

INSTALLATION and INCLUDED SOFTWARE

- The `\donttest` sections of R's help files can be tested by `make check TEST_DONTTEST=TRUE`.
- It is possible to request the use of system `valgrind` headers *via* `configure` option '`--with-system-valgrind-headers`': note the possible future incompatibility of such headers discussed in the 'R Installation and Administration' manual. (Wish of PR#16068.)
- The included version of `liblzma` has been updated to `xz-utils` 5.0.7 (minor bug fixes from 5.0.5).
- `configure` options '`--with-system-zlib`', '`--with-system-bzlib`' and '`--with-system-pcre`' are now the default. For the time being there is fallback to the versions included in the R sources if no system versions are found or (unlikely) if they are too old.

  Linux users should check that the `-devel` or `-dev` versions of packages **zlib**, **bzip2**/**libbz2** and **pcre** as well as **xz-devel**/**liblzma-dev** (or similar names) are installed.
- `configure` by default looks for the `texi2any` script from **texinfo** 5.1 or later, rather than the `makeinfo` program. (`makeinfo` is a link to the Perl script `texi2any` in **texinfo** 5.x.)
- `R CMD INSTALL` gains an option '`--built-timestamp=STAMP`' allowing 100% reproducible package building, thanks to Dirk Eddelbuettel.

UTILITIES

- There is support for testing the `\dontrun` and `\donttest` parts of examples in packages.

  `tools::testInstalledPackage()` accepts new arguments `commentDontrun = FALSE` and `commentDonttest = FALSE`.

  `R CMD check` gains options '`--run-dontrun`' and '`--run-donttest`'.
- The HTML generated by `tools::Rd2HTML()` and `tools::toHTML()` methods is now 'XHTML 1.0 Strict'.
- The **compiler** package's utility function `setCompilerOptions()` now returns the old values invisibly. The initial optimization level can also be set with the environment variable `R_COMPILER_OPTIMIZE`.

- `R CMD build` adds a '`NeedsCompilation`' field if one is not already present in the 'DESCRIPTION' file.
- `R CMD check` gains option '`--test-dir`' to specify an alternative set of tests to run.
- `R CMD check` will now by default continue with testing after many types of errors, and will output a summary count of errors at the end if any have occurred.
- `R CMD check` now checks that the '`Title`' and '`Description`' fields are correctly terminated.
- `R CMD check --as-cran` now:
  - checks a 'README.md' file can be processed: this needs `pandoc` installed.
  - checks the existence and accessibility of URLs in the 'DESCRIPTION', 'CITATION', 'NEWS.Rd' and 'README.md' files and in the help files (provided the build has `libcurl` support).
  - reports non-ASCII characters in R source files when there is no package encoding declared in the 'DESCRIPTION' file.
  - reports (apparent) S3 methods exported but not registered.
  - reports overwriting registered S3 methods from base/recommended packages. (Such methods are replaced in the affected package for the rest of the session, even if the replacing namespace is unloaded.)
  - reports if the `Title` field does not appear to be in title case (see 'Writing R Extensions': there may be false positives, but note that technical words should be single-quoted and will then be accepted).

  Most of these checks can also be selected by environment variables: see the 'R Internals' manual.

C-LEVEL FACILITIES

- New C API utility `logspace_sum(logx[],n)`.
- Entry points `rbinom_mu`, `rnbinom_mu` and `rmultinom` are remapped (by default) to `Rf_rbinom_mu` etc. This requires packages using them to be re-installed.
- `.C(DUP = FALSE)` and `.Fortran(DUP = FALSE)` are now ignored, so arguments are duplicated if `DUP = TRUE` would do so. As their help has long said, `.Call()` is much preferred.
- New entry point `R_allocLD`, like `R_alloc` but guaranteed to have sufficient alignment for `long double` pointers.
- `isPairList()` now returns `TRUE` for DOTSXP.

WINDOWS BUILD CHANGES

A number of changes to the Windows build system are in development. The following are currently in place.

- Installation using external binary distributions of **zlib**, **bzip2**, **liblzma**, **pcre**, **libpng**, **jpeglib** and **libtiff** is now required, and the build instructions have been revised.
- A new `make` target `rsync-extsoft` has been added to obtain copies of the external libraries from CRAN.

- Building the manuals now requires `texi2any` from **texinfo** 5.1 or later. CRAN binary builds include the manuals, but by default builds from source will not, and they will be accessed from CRAN. See the comments in 'src/gnuwin32/MkRules.dist' for how to specify the location of `texi2any`.
- (Windows) Changes have been made to support an experimental Windows toolchain based on GCC 4.9.2. The default toolchain continues to be based on GCC 4.6.3, as the new toolchain is not yet stable enough. A change to a new toolchain is expected during the R 3.2.x lifetime.

PACKAGE INSTALLATION

- (Windows) The use of macro `ZLIB_LIBS` in file 'src/Makevars.win' (which has not been documented for a long time) now requires an external 'libz.a' to be available (it is part of the 'goodies' used to compile Windows binary packages). It would be simpler to use `-lz` instead.
- The default for option `pkgType` on platforms using binary packages is now `"both"`, so source packages will be tried if binary versions are not available or not up to date.

  There are options for what `install.packages(type = "both")` (possibly called *via* `update.packages()`) will do if compilation of a source package is desirable: see `?options` (under **utils**).

  If you intend not to accept updates as source packages, you should use `update.packages(type = "binary")`.

DEPRECATED AND DEFUNCT

- `download.file(method = "lynx")` is defunct.
- Building R using the included versions of `zlib`, `bzip2`, `xz` and PCRE is deprecated: these are frozen (bar essential bug-fixes) and will be removed for R 3.3.0.
- The `configure` option '`--with-valgrind-instrumentation=3`' has been withdrawn, as it did not work with recent `valgrind` headers: it is now treated as level `2`.
- The `MethodsList` class in package **methods** had been deprecated in R 2.11.0 and is defunct now. Functions using it are defunct if they had been deprecated in R 2.11.0, and are deprecated now, otherwise.

BUG FIXES

- Fixed two obscure bugs in pairlist subassignment, reported by Radford Neal as part of pqR issue 16.
- Fixes for bugs in handling empty arguments and argument matching by name in `log()`.
- `all.equal()` gains methods for `environments` and `refClasses`.
- `[<-` and `[[<-` gain S4 `data.frame` methods to avoid corruption of S4 class information by the S3 methods.
- `callNextMethod()` should now work within a `.local` call when `...` is absent from `formals(.local)`.
- `dput(pairlist(x))` generates a call to the `pairlist` constructor instead of the `list` constructor.
- Fix `missing()` when arguments are propagated through `...`. (PR#15707)

- `eigen(m)` now defaults to `symmetric = TRUE` even when the dimnames are asymmetric if the matrix is otherwise symmetric. (PR#16151)
- Fix issues with forwarding `...` through `callGeneric()` and `callNextMethod()`. (PR#16141)
- `callGeneric()` now works after a `callNextMethod()`.
- Subclass information is kept consistent when replacing an ordinary S4 class with an "old class" *via* the `S4Class` argument to `setOldClass()`. Thus, for example, a `data.frame` is valid for a `list` argument in the signature, and a `factor` is valid for `vector` arguments.
- In `qbeta()` the inversion of `pbeta()` is much more sophisticated. This works better in corner cases some of which failed completely previously (PR#15755), or were using too many iterations.
- Auto-printing no longer duplicates objects when printing is dispatched to a method.
- `kmeans(x,k)` would fail when `nrow(x) >= 42949673`. (Comment 6 of PR#15364)
- 'Abbreviated' locale-specific day and month names could have been truncated in those rare locales where there are the same as the full names.
- An irrelevant warning message from updating subclass information was silenced (the namespace would not be writable in this case).

CHANGES IN R 3.1.3

NEW FEATURES

- The internal method of `download.file()` can now handle files larger than 2GB on 32-bit builds which support such files (tested on 32-bit R running on 64-bit Windows).
- `kruskal.test()` warns on more types of suspicious input.
- The `as.dendrogram()` method for `"hclust"` objects gains a `check` argument protecting against memory explosion for invalid inputs.
- `capabilities()` has a new item `long.double` which indicates if the build uses a `long double` type which is longer than `double`.
- `nlm()` no longer modifies the callback argument in place (a new vector is allocated for each invocation, which mimics the implicit duplication that occurred in R < 3.1.0); note that this is a change from the previously documented behavior. (PR#15958)
- `icuSetCollate()` now accepts `locale = "ASCII"` which uses the basic C function `strcmp` and so collates strings byte-by-byte in numerical order.
- `sessionInfo()` tries to report the OS version in use (not just that compiled under, and including details of Linux distributions).
- `model.frame()` (used by `lm()` and many other modelling functions) now warns when it drops contrasts from factors. (Wish of PR#16119)
- `install.packages()` and friends now accept the value `type = "binary"` as a synonym for the native binary type on the platform (if it has one).
- Single source or binary files can be supplied for `install.packages(type = "both")` and the appropriate type and `repos = NULL` will be inferred.
- New function `pcre_config()` to report on some of the configuration options of the version of PCRE in use. In particular, this reports if regular expressions using '`\p{xx}`' are supported.

- (Windows.) `download.file(cacheOK = FALSE)` is now supported when 'internet2.dll' is used.
- `browseURL()` has been updated to work with Firefox 36.0 which has dropped support for the '`-remote`' interface.

INSTALLATION and INCLUDED SOFTWARE

- The included version of PCRE has been updated to 8.36.
- `configure` accepts '`MAKEINFO=texi2any`' as another way to ensure **texinfo** 5.x is used when both 5.x and 4.x are installed.

UTILITIES

- `R CMD check` now checks the packages used in `\donttest` sections of the examples are specified in the 'DESCRIPTION' file. (These are needed to run the examples interactively.)
- `R CMD check` checks for the undeclared use of GNU extensions in Makefiles, and for Makefiles with a missing final linefeed.

  `R CMD build` will correct line endings in all Makefiles, not just those in the 'src' directory.
- `R CMD check` notes uses of `library()` and `require()` in package code: see the section 'Suggested packages' of 'Writing R Extensions' for good practice.

DEPRECATED AND DEFUNCT

- The `configure` option '`--with-valgrind-instrumentation=3`' is deprecated and will be removed in R 3.2.0.

BUG FIXES

- (Windows.) `Rscript.exe` was missing a manifest specifying the modern style for common controls (e.g., the download progress bar).
- If a package had extra documentation files but no vignette, the HTML help system produced an empty index page.
- The parser now gives an error if a null character is included in a string using Unicode escapes. (PR#16046)
- `qr.Q()` failed on complex arguments due to pre-3.0(!) typo. (PR#16054)
- `abs()` failed with named arguments when the argument was complex. (PR#16047)
- `"noquote"` objects may now be used as columns in data frames. (PR#15997)
- Some values with extremely long names were printed incorrectly. (PR#15999)
- Extremely large exponents on zero expressed in scientific notation (e.g. `0.0e50000`) could give `NaN`. (PR#15976)
- `download.file()` reported downloaded sizes as 0KB if less than 1MB, only for R 3.1.2 and only on big-endian platforms.
- `prompt()` did not escape percent signs in the automatically generated usage section of help files.

- `drop.terms()` dropped some of the attributes of the object it was working with. (PR#16029)
- (Windows.) The command completion in `Rgui.exe` messed up the console. (PR#15791)
- (Windows.) The `choose.files()` command returned a blank string when the user asked for a single file but cancelled the request. (PR#16074)
- `Math2` S4 group generics failed to correctly dispatch `"structure"`- and `"nonStructure"`-derived classes.
- `loadNamespace()` imposed undocumented restrictions on the `versionCheck` parameter. (Reported by Geoff Lee.)
- Rare over-runs detected by AddressSanitizer in `substr()` and its replacement version have been avoided.

  Inter alia that fix gives the documented behaviour for `substr(x,1,2) <-""` (subsequently reported as PR#16214).
- Loading packages incorrectly defining an S4 generic followed by a function of the same name caused an erroneous cyclic namespace dependency error.
- Declared vignette encodings are now always passed to the vignette engine.
- Port Tomas Kalibera's fix from R-devel that restores the `loadMethod()` fast path, effectively doubling the speed of S4 dispatch.
- `power.t.test()` and `power.prop.test()` now make use of the `extendInt` option of `uniroot()` and hence work in more extreme cases. (PR#15792)
- If a package was updated and attached when its namespace was already loaded, it could end up with parts from one version and parts from the other. (PR#16120)
- `tools:::.Rdconv()` didn't accept `--encoding=` due to a typo. (PR#16121)
- Unix-alike builds without a suitable `makeinfo` were documented to link the missing HTML manuals to CRAN, but did not.
- `save(*,ascii=TRUE)` and `load()` now correctly deal with `NaN`'s. (PR#16137)
- `split.Date()` retains fractional representations while avoiding incomplete class propagation.
- 'R_ext/Lapack.h' had not been updated for changes made by LAPACK to the argument lists of its (largely internal) functions `dlaed2` and `dlaed3`. (PR#16157)
- `RShowDoc("NEWS","txt")` had not been updated for the layout changes of R 3.1.0.
- The `xtfrm()` method for class `"Surv"` has been corrected and its description expanded.
- `mode(x) <-y` would incorrectly evaluate `x` before changing its mode. (PR#16215)
- `besselJ(1,2^64)` and `besselY(..)` now signal a warning, returning `NaN` instead of typically segfaulting. (Issue 3 of PR#15554)
- HTML conversion of '`\href`' markup in '.Rd' files did not remove the backslash from '`\%`' and so gave an invalid URL. In a related change, the '`\`' escape is now required in such URLs.

CHANGES IN R 3.1.2

NEW FEATURES

- `embedFonts()` now defaults to `format = "ps2write"` for '.ps' and '.eps' files. This is available in Ghostscript 9.x (since 2010) whereas the previous default, `format = "pswrite"`, was removed in Ghostscript 9.10.

- For consistency with `[dpqr]norm()`, `[dp]lnorm(sdlog = 0)` model a point mass at `exp(mulog)` rather than return `NaN` (for an error).

- `capabilities()` now reports if ICU is compiled in for use for collation (it is only actually used if a suitable locale is set for collation, and never for a `C` locale).

- (OS X only.) Package **tcltk** checks when loaded if it is linked against the CRAN X11-based Tcl/Tk and if so that the Tcl/Tk component and the X11 libraries are installed. This allows more informative error messages to be given advising the installation of the missing component or of XQuartz.

  The `X11()` device and X11-based versions of the data editor and viewer (invoked by `edit()` and `View()` for data frames and matrices from command-line R) check that the X11 libraries are installed and if not advises installing XQuartz.

- `icuSetCollate()` allows `locale = "default"`, and `locale = "none"` to use OS services rather than ICU for collation.

  Environment variable `R_ICU_LOCALE` can be used to set the default ICU locale, in case the one derived from the OS locale is inappropriate (this is currently necessary on Windows).

- New function `icuGetCollate()` to report on the ICU collation locale in use (if any).

- `utils::URLencode()` was updated to use unreserved and reserved characters from RFC 3986 (http://tools.ietf.org/html/rfc3986) instead of RFC 1738.

- `unique(warnings())` and `c(warnings())` are now supported.

- The Bioconductor 'version' used by `setRepositories()` now defaults to `3.0`. (It can be set at runtime *via* environment variable `R_BIOC_VERSION`.)

- Omegahat is no longer listed as providing Windows binary packages, e.g. by `setRepositories()`. It has no binary packages available for R 3.1.x and those for earlier versions were 32-bit only.

INSTALLATION and INCLUDED SOFTWARE

- The `configure` script reports on the more important capabilities/options which will not be compiled in.

  More types of external BLAS are recognized by name in that report.

- When building R as a shared library, the '`-L${R_HOME}/lib${R_ARCH}`' flag is placed earlier in the link commands used during installation and when packages are installed: this helps ensure that the current build has priority if an R shared library has already been installed by e.g. `install-libR` in a library mentioned in `LDFLAGS` (and not in 'your system's library directory' as documented). (Wish of PR#15790.)

- LaTeX package **upquote** is no longer required for R's use of **inconsolata**.

- (Windows only) If both 32- and 64-bit versions of R are installed, the 'bin/R.exe' and 'bin/Rscript.exe' executables now run 64-bit R. (To run 32-bit R, overwrite these files with copies of 'bin/i386/Rfe.exe'.)

UTILITIES

- Running `R CMD check` with `_R_CHECK_DEPENDS_ONLY_` true now makes the '`VignetteBuilder`' packages available even if they are listed in '`Suggests`', since they are needed to recognise and process non-Sweave vignettes.
- `R CMD check` now reports empty `importFrom` declarations in a 'NAMESPACE' file, as these are common errors (writing `importFrom(`*Pkg*`)` where `import(`*Pkg*`)` was intended).
- `R CMD check` now by default checks code usage directly on the package namespace without loading and attaching the package and its suggests and enhances. For good practice with packages in the '`Suggests`' field, see §1.1.3.1 of 'Writing R Extensions'. For use of lazy-data objects in the package's own code, see `?data`.

BUG FIXES

- `dmultinom()` did not handle non-finite probabilities correctly.
- `prettyNum(x,zero.print=*)` now also works when `x` contains `NA`s.
- A longstanding bug exhibited by `nlminb()` on Windows was traced to a compiler bug in gcc 4.6.3; a workaround has been put in place. (PR#15244 and PR#15914).
- Rendering of `\command` in HTML versions of help pages has been improved: this is particularly evident on the help page for `INSTALL`.
- `as.hexmode(x)` and `as.octmode(x)` now behave correctly for some numeric `x`, e.g., `c(NA,1)` or `c(1,pi)`.
- `drop1()` failed if the `scope` argument had no variables to drop. (PR#15935)
- `edit()` (and hence `fix()`) failed if an object had a non-character attribute named `"source"` (an attribute that had been used in R prior to version 2.14.0).
- `callGeneric()` could fail if the generic had `...` as a formal argument. (PR#15937).
- Forking in package **parallel** called C entry point `exit` in the child. This was unsafe (`_exit` should have been called), and could flush `stdin` of the main R process (seen most often on Solaris).

  As good practice, `stdout` is now flushed before forking a child.
- R objects such as `list(`a\b` = 1)` now print correctly.
- `getAnywhere("C_pbinom")` now returns correctly a single object (rather than unlisting it).
- The `confint()` method for `nls()` fits failed it these has specified parameter limits despite using an algorithm other than `"port"`. (PR#15960)
- Subclassing an S4 class failed if the class required arguments to the generator, through its `initialize()` method.
- `removeSource()` did not properly handle expressions containing arguments that were supplied as missing, e.g. `x[i,]`. (PR#15957)
- `as.environment(list())` now works, and `as.list()` of such an environment is now the same as `list()`. (PR#15926)
- Several **tcltk** functions failed when run in unusual environments. (PR#15970)
- `options(list())` now works (trivially). (PR#15979)

- `merge(<dendrogram>,..)` now works correctly for two 'independent' dendrograms (PR#15648), and still compatibly *via* `adjust = "auto"` e.g. for two branches of an existing dendrogram.
- The `plot` method for `"hclust"` objects gets an optional argument `check`; when that is true (the default) it checks more carefully for valid input.
- (Windows only) If a user chose to install 64 bit R but not 32 bit R, the 'bin/R' and 'bin/Rscript' executables failed to run. (PR#15981)
- Various possible buffer overruns have been prevented, and missed memory protection added. (PR#15990)
- `Rscript` no longer passes `--args` to `R` when there are no extra ("user") arguments.
- objects like `getClass("refClass")@prototype` now `print()` and `str()` without error.
- `identical()` now also looks at the S4 bit.
- `hist(x,breaks)` is more robust in adding a small fuzz to few breaks when some are very large. (PR#15988)
- `sub()` and `gsub()` did not handle regular expressions like `"\s{2,}"` properly if the text contained `NA` or non-ASCII elements in a UTF-8 locale. Part of this was due to a bug in the TRE library. (PR#16009)
- `RShowDoc("NEWS")` now displays the PDF version.
- Matrices and arrays with last dimension zero did not print at all or incompletely. (PR#16012)
- `plot.histogram()` and hence `hist()` now respect the `xaxs`, `yaxs` and `lab` graphics parameters. (PR#16021)
- `bw.SJ(x)` and other bw.*() no longer segfault when `x` contains non-finite values. (PR#16024)
- `R CMD Rd2pdf` unintentionally ignored its '`--os`' option.
- The internal method of `download.file()` was not reporting file sizes and progress correctly on files larger than 2GB (inherited from `libxml2`). This is corrected for 64-bit builds (32-bit platforms may not support such files, but where possible will be supported in future versions of R).
- Work around a bug in OS X Yosemite where key environment variables may be duplicated causing issues in subprocesses. The duplicates are now removed on R startup (*via* Rprofile). (PR#16042)
- Adjust X11 auto-launch detection in DISPLAY on OS X to recognize latest XQuartz.

CHANGES IN R 3.1.1

NEW FEATURES

- When `attach()` reports conflicts, it does so compatibly with `library()` by using `message()`.
- `R CMD Sweave` no longer cleans any files by default, compatibly with versions of R prior to 3.1.0. There are new options '`--clean`', '`--clean=default`' and '`--clean=keepOuts`'.
- `tools::buildVignette()` and `tools::buildVignettes()` with `clean = FALSE` no longer remove any created files. `buildvignette()` gains a `keep` argument for more cleaning customization.

- The Bioconductor 'version' used by `setRepositories()` can now be set by environment variable `R_BIOC_VERSION` at runtime, not just when R is installed. (It has been stated that Bioconductor will switch from 'version' 2.14 to 'version' 3.0 during the lifetime of the R 3.1 series.)
- Error messages from bugs in embedded '`Sexpr`' code in Sweave documents now report the source location.
- `type.convert()`, `read.table()` and similar `read.*()` functions get a new `numerals` argument, specifying how numeric input is converted when its conversion to double precision loses accuracy. The default value, `"allow.loss"` allows accuracy loss, as in R versions before 3.1.0.
- For some compilers, integer addition could overflow without a warning. R's internal code for both integer addition and subtraction is more robust now. (PR#15774)
- The function determining the default number of knots for `smooth.spline()` is now exported, as `.nknots.smspl()`.
- `dbeta(,a,b)`, `pbeta()`, `qbeta()` and `rbeta()` are now defined also for $a = 0$, $b = 0$, or infinite a and b (where they typically returned `NaN` before).
- Many package authors report that the RStudio graphics device does not work correctly with their package's use of `dev.new()`. The new option `dev.new(noRStudioGD = TRUE)` replaces the RStudio override by the default device as selected by R itself, still respecting environment variables `R_INTERACTIVE_DEVICE` and `R_DEFAULT_DEVICE`.
- `readRDS()` now returns visibly.
- Modifying internal logical scalar constants now results in an error instead of a warning.
- `install.packages(repos = NULL)` now accepts '`http://`' or '`ftp://`' URLs of package archives as well as file paths, and will download as required. In most cases `repos = NULL` can be deduced from the extension of the URL.
- The warning when using partial matching with the `$` operator on data frames is now only given when `options("warnPartialMatchDollar")` is `TRUE`.
- Package help requests like `package?foo` now try the package `foo` whether loaded or not.
- General help requests now default to trying all loaded packages, not just those on the search path.
- Added a new function `promptImport()`, to generate a help page for a function that was imported from another package (and presumably re-exported, or help would not be needed).

INSTALLATION and INCLUDED SOFTWARE

- `configure` option '`--with-internal-tzcode`' can now be used with variable `rsharedir`.
- The included version of PCRE has been updated to 8.35.
- There is a new target `make uninstall-libR` to remove an installed shared/static 'libR'.

  `make install-libR` now works if a sub-architecture is used, although the user will need to specify `libdir` differently for different sub-architectures.
- There is more extensive advice on which LaTeX packages are required to install R or to make package manuals (as done by `R CMD check`) in the 'Writing R Extensions' manual.

- Compilers/linkers were handling the visibility control in 'src/extra/xz' inconsistently (and apparently in some cases incorrectly), so it has been simplified. (PR#15327)
- (Windows) There is updated support for the use of ICU for collation: see the 'R Installation and Administration Manual'.

BUG FIXES

- `dbinom(x,n)`, `pbinom()`, `dpois()`, etc, are slightly less restrictive in checking if `n` is integer-valued. (Wish of PR#15734.)
- `pchisq(x,df,ncp,log.p = TRUE)` is more accurate and no longer underflows for small `x` and `ncp <80`, e.g, for `pchisq(1e-5,df = 100,ncp = 1,log = TRUE)`. (Based on PR#15635 and a suggestion by Roby Joehanes.)
- The `s` ("step into") command in the debugger would cause R to step into expressions evaluated there, not just into functions being debugged. (PR#15770)
- The C code used by `strptime()` rejected time-zone offsets of more than `+1200` (`+1245`, `+1300` and `+1400` can occur). (PR#15768)
- (Windows only.) `png(type = "cairo",antialias = "gray")` was not accepted. (PR#15760)
- Use of `save(...,envir=)` with named objects could fail. (PR#15758)
- `Sweave()` mis-parsed 'Sexpr' expressions that contained backslashes. (PR#15779)
- The return value from `options(foo = NULL)` was not the previous value of the option. (PR#15781)
- `enc2utf8()` and `enc2native()` did not always mark the encoding of the return values when it was known.
- `dnbinom(x,size = <large>,mu,log = TRUE)` no longer underflows to -Inf for large `mu`, thanks to a suggestion from Alessandro Mammana (MPI MolGen, Berlin).
- `pbeta(x,a,b,log = TRUE)` no longer behaves discontinuously (in a small x-region) because of denormalized numbers. Also, `pbeta(1-1e-12,1e30,1.001,log=TRUE)` now terminates "in real time".
- The `"CRAN"` filter (see `available.packages()`) no longer removes duplicates other than of packages on CRAN, and does not fail if there is no CRAN repository in `getOption("repos")`.
- The device listing from `dev2bitmap()` and `bitmap()` was truncated to 1000 characters: modern versions of GhostScript on most platforms have many more devices.
- (Windows.) Commands such as `Sys.which()` and `pipe()` which needed to find the full path to a command could segfault if the 'long' path name was much longer than the 'short' path name (which `Sys.which()` returns), as the behaviour of the Windows API call had changed.
- `R CMD build` will fail with an error if one of the packages specified in the `'VignetteBuilder'` field is not installed. (Without loading those packages it cannot be ascertained which files are intended to be vignettes. This means that the `'VignetteBuilder'` packages have to be installed for package checking too.) (Wish of PR#15775.)
- Misguided attempts to use `chull()` with non-finite points now give an error (related to PR#15777).

- For a formula with exactly 32 variables the 32nd variable was aliased to the intercept in some C-level computations of terms, so that for example attempting to remove it would remove the intercept instead (and leave a corrupt internal structure). (PR#15735)
- `anyDuplicated()` silently returned wrong values when the first duplicate was at an index which was too large to be stored in an integer vector (although a lot of RAM and patience would have been needed to encounter this).
- `tools::Rd2ex(commentDontrun = FALSE)` failed if the block had only one line.
- Hexadecimal constants such as `0x110p-5L` which were incorrectly qualified by `L` were parsed incorrectly since R 3.0.0, with a slightly garbled warning. (PR#15753)
- `system()` returned success on some platforms even if the system was unable to launch a process. (PR#15796)
- (Windows `Rgui` console.) Unbuffered output was sometimes not output immediately if the prompt was not on the last line of the console.
- The built-in help server did not declare the encoding for the 'DESCRIPTION' or other text files to be the package encoding, so non-ASCII characters could be displayed incorrectly.
- R is now trying harder to not cleanup child processes that were not spawned by `mcparallel()` on platforms that provide information about the source process of the `SIGCHLD` signal. This allows 3rd party libraries to manage the exit status of children that they spawn without R interfering.
- `mcmapply()` was only parallelizing if the number of jobs was bigger than the number of cores. It now parallelizes if the number of jobs is more than one.
- Auto-printing would re-evaluate its argument when trying to dispatch to a print method. This is now avoided when possible.
- Unserializing (including `load()` and `readRDS()`) could silently return incorrect numeric values from ASCII saves if there was a read error.
- `getParseData()` could return incorrect values for the parents of some elements. (Reported by Andrew Redd.)
- Attempting to use data frames of 2^31 or more rows with `merge()` or to create a merged data frame of that size now gives a clearer error message.
- `parse()` did not check its `file` argument was a connection if it was not a character string, so e.g. `parse(FALSE)` attempted to read from `stdin`.

  Nor did `dump()` and `dput()`.
- The `"help.try.all.packages"` option was ignored when the shortcut syntax for help was used, e.g. `?foo`.
- A potential segfault in string allocation has been fixed. (Found by Radford Neal.)
- Potential memory protection errors in `sort()` and `D()` have been fixed. (Found by Radford Neal.)
- Fixed a lack of error checking in graphics event functions. (Found by Radford Neal; a different patch used here than the one in pqR.)
- `numericDeriv()` sometimes miscalculated the gradient. (PR#15849, reported originally by Radford Neal)

CHANGES IN R 3.1.0

NEW FEATURES

- `type.convert()` (and hence by default `read.table()`) returns a character vector or factor when representing a numeric input as a double would lose accuracy. Similarly for complex inputs.

  If a file contains numeric data with unrepresentable numbers of decimal places that are intended to be read as numeric, specify `colClasses` in `read.table()` to be `"numeric"`.

- `tools::Rdiff(useDiff = FALSE)` is closer to the POSIX definition of `diff -b` (as distinct from the description in the `man` pages of most systems).

- New function `anyNA()`, a version of `any(is.na(.))` which is fast for atomic vectors, based on a proposal by Tim Hesterberg. (Wish of PR#15239.)

- `arrayInd(*,useNames = TRUE)` and, analogously, `which(*,arr.ind = TRUE)` now make use of `names(.dimnames)` when available.

- `is.unsorted()` now also works for `raw` vectors.

- The `"table"` method for `as.data.frame()` (also useful as `as.data.frame.table()`) now passes `sep` and `base` arguments to `provideDimnames()`.

- `uniroot()` gets new optional arguments, notably `extendInt`, allowing to auto-extend the search interval when needed. The return value has an extra component, `init.it`.

- `switch(f,...)` now warns when `f` is a factor, as this typically happens accidentally where the useR meant to pass a character string, but `f` is treated as integer (as always documented).

- The parser has been modified to use less memory.

- The way the unary operators (`+ -!`) handle attributes is now more consistent. If there is no coercion, all attributes (including class) are copied from the input to the result: otherwise only names, dims and dimnames are.

- `colorRamp()` and `colorRampPalette()` now allow non-opaque colours and a ramp in opacity *via* the new argument `alpha = TRUE`. (Suggested by Alberto Krone-Martins, but optionally as there are existing uses which expect only RGB values.)

- `grid.show.layout()` and `grid.show.viewport()` get an optional `vp.ex` argument.

- There is a new function `find_gs_cmd()` in the **tools** package to locate a GhostScript executable. (This is an enhanced version of a previously internal function there.)

- `object.size()` gains a `format()` method.

- There is a new family, `"ArialMT"`, for the `pdf()` and `postscript()` devices. This will only be rendered correctly on viewers which have access to Monotype TrueType fonts (which are sometimes requested by journals).

- The text and PDF news files, including 'NEWS' and 'NEWS.2', have been moved to the 'doc' directory.

- `combn(x,simplify = TRUE)` now gives a factor result for factor input `x` (previously user error). (Related to PR#15442.)

- Added `utils::fileSnapshot()` and `utils::changedFiles()` functions to allow snapshots and comparison of directories of files.

- `make.names(names,unique=TRUE)` now tries to preserve existing names. (Suggestion of PR#15452.)

- New functions `cospi(x)`, `sinpi(x)`, and `tanpi(x)`, for more accurate computation of `cos(pi*x)`, etc, both in R and the C API. Using these gains accuracy in some cases, e.g., inside `lgamma()` or `besselI()`. (Suggested by Morten Welinder in PR#15529.)

- `print.table(x,zero.print = ".")` now also has an effect when `x` is not integer-valued.

- There is more support to explore the system's idea of time-zone names. `Sys.timezone()` tries to give the current system setting by name (and succeeds at least on Linux, OS X, Solaris and Windows), and `OlsonNames()` lists the names in the system's Olson database. `Sys.timezone(location = FALSE)` gives the previous behaviour.

- Platforms with a 64-bit `time_t` type are allowed to handle conversions between the `"POSIXct"` and `"POSIXlt"` classes for date-times outside the 32-bit range (before 1902 or after 2037): the existing workarounds are used on other platforms. (Note that time-zone information for post-2037 is speculative at best, and the OS services are tested for known errors and so not used on OS X.)

  Currently `time_t` is usually `long` and hence 64-bit on Unix-alike 64-bit platforms: however in several cases the time-zone database is 32-bit. For R for Windows it is 64-bit (for both architectures as from this version).

- The `"save.defaults"` option can include a value for `compression_level`. (Wish of PR#15579.)

- `colSums()` and friends now have support for arrays and data-frame columns with 2^{31} or more elements.

- `as.factor()` is faster when `f` is an unclassed integer vector (for example, when called from `tapply()`).

- `fft()` now works with longer inputs, from the 12 million previously supported up to 2 billion. (PR#15593)

- Complex `svd()` now uses LAPACK subroutine `ZGESDD`, the complex analogue of the routine used for the real case.

- Sweave now outputs '.tex' files in UTF-8 if the input encoding is declared to be UTF-8, regardless of the local encoding. The UTF-8 encoding may now be declared using a LaTeX comment containing the string `%\SweaveUTF8` on a line by itself.

- `file.copy()` gains a `copy.date` argument.

- Printing of date-times will make use of the time-zone abbreviation in use at the time, if known. For example, for Paris pre-1940 this could be '`LMT`', '`PMT`', '`WET`' or '`WEST`'. To enable this, the `"POSIXlt"` class has an optional component `"zone"` recording the abbreviation for each element.

  For platforms which support it, there is also a component `"gmtoff"` recording the offset from GMT where known.

- (On Windows, by default on OS X and optionally elsewhere.) The system C function `strftime` has been replaced by a more comprehensive version with closer conformance to the POSIX 2008 standard.

- `dnorm(x,log = FALSE)` is more accurate (but somewhat slower) for $|x| > 5$; as suggested in PR#15620.

- Some versions of the `tiff()` device have further compression options.

- `read.table()`, `readLines()` and `scan()` have a new argument to influence the treatment of embedded nuls.

- Avoid duplicating the right hand side values in complex assignments when possible. This reduces copying of replacement values in expressions such as `Z$a <-a0` and `ans[[i]] <-tmp`: some package code has relied on there being copies.

  Also, a number of other changes to reduce copying of objects; all contributed by or based on suggestions by Michael Lawrence.

- The `fast` argument of `KalmanLike()`, `KalmanRun()` and `KalmanForecast()` has been replaced by `update`, which instead of updating `mod` in place, optionally returns the updated model in an attribute `"mod"` of the return value.

- `arima()` and `makeARIMA()` get a new optional argument `SSinit`, allowing the choice of a different **s**tate **s**pace initialization which has been observed to be more reliable close to non-stationarity: see PR#14682.

- `warning()` has a new argument `noBreaks.`, to simplify post-processing of output with `options(warn = 1)`.

- `pushBack()` gains an argument `encoding`, to support reading of UTF-8 characters using `scan()`, `read.table()` and related functions in a non-UTF-8 locale.

- `all.equal.list()` gets a new argument `use.names` which by default labels differing components by names (if they match) rather than by integer index. Saved R output in packages may need to be updated.

- The methods for `all.equal()` and `attr.all.equal()` now have argument `check.attributes` after `...` so it cannot be partially nor positionally matched (as it has been, unintentionally).

  A side effect is that some previously undetected errors of passing empty arguments (no object between commas) to `all.equal()` are detected and reported.

  There are explicit checks that `check.attributes` is logical, `tolerance` is numeric and `scale` is `NULL` or numeric. This catches some unintended positional matching.

  The message for `all.equal.numeric()` reports a `"scaled difference"` only for `scale != 1`.

- `all.equal()` now has a `"POSIXt"` method replacing the `"POSIXct"` method.

- The `"Date"` and `"POSIXt"` methods of `seq()` allows `by = "quarter"` for completeness (`by = "3 months"` always worked).

- `file.path()` removes any trailing separator on Windows, where they are invalid (although sometimes accepted). This is intended to enhance the portability of code written by those using POSIX file systems (where a trailing `/` can be used to confine path matching to directories).

- New function `agrepl()` which like `grepl()` returns a logical vector.

- `fifo()` is now supported on Windows. (PR#15600)

- `sort.list(method = "radix")` now allows negative integers (wish of PR#15644).

- Some functionality of `print.ts()` is now available in `.preformat.ts()` for more modularity.

- `mcparallel()` gains an option `detach = TRUE` which allows execution of code independently of the current session. It is based on a new `estranged = TRUE` argument to `mcfork()` which forks child processes such that they become independent of the parent process.

- The `pdf()` device omits circles and text at extremely small sizes, since some viewers were failing on such files.

- The rightmost break for the "months", "quarters" and "years" cases of hist.POSIXlt() has been increased by a day. (Inter alia, fixes PR#15717.)
- The handling of DF[i,] <-a where i is of length 0 is improved. (Inter alia, fixes PR#15718.)
- hclust() gains a new method "ward.D2" which implements Ward's method correctly. The previous "ward" method is "ward.D" now, with the old name still working. Thanks to research and proposals by Pierre Legendre.
- The sunspot.month dataset has been amended and updated from the official source, whereas the sunspots and sunspot.year datasets will remain immutable. The documentation and source links have been updated correspondingly.
- The summary() method for "lm" fits warns if the fit is essentially perfect, as most of the summary may be computed inaccurately (and with platform-dependent values).

  Programmers who use summary() in order to extract just a component which will be reliable (e.g., $cov.unscaled) should wrap their calls in suppressWarnings().

INSTALLATION and INCLUDED SOFTWARE

- The included version of LAPACK has been updated to 3.5.0.
- There is some support for parallel testing of an installation, by setting TEST_MC_CORES to an integer greater than one to indicate the maximum number of cores to be used in parallel. (It is worth specifying at least 8 cores if available.) Most of these require a make program (such as GNU make and dmake) which supports the $MAKE -j nproc syntax.

  Except on Windows: the tests of standard package examples in make check are done in parallel. This also applies to running tools::testInstalledPackages().

  The more time-consuming regression tests are done in parallel.

  The package checks in make check-devel and make check-recommended are done in parallel.
- More of make check will work if recommended packages are not installed: but recommended packages remain needed for thorough checking of an R build.
- The version of 'tzcode' included in 'src/extra/tzone' has been updated. (Formerly used only on Windows.)
- The included (64-bit) time-zone conversion code and Olson time-zone database can be used instead of the system version: use configure option '--with-internal-tzcode'. This is the default on Windows and OS X. (Note that this does not currently work if a non-default rsharedir configure variable is used.)

  (It might be necessary to set environment variable TZ on OSes where this is not already set, although the system timezone is deduced correctly on at least Linux, OS X and Windows.)

  This option also switches to the version of strftime included in directory 'src/extra/tzone'.
- configure now tests for a C++11-compliant compiler by testing some basic features. This by default tries flags for the compiler specified by 'CXX', but an alternative compiler, options and standard can be specified by variables 'CXX1X', 'CXX1XFLAGS' and 'CXX1XSTD' (e.g., '-std=gnu++11').
- R can now optionally be compiled to use reference counting instead of the NAMED mechanism by defining SWITCH_TO_REFCNT in 'Rinternals.h'. This may become the default in the future.

- There is a new option '`--use-system-tre`' to use a suitable system **tre** library: at present this means a version from their `git` repository, after corrections. (Wish of PR#15660.)

PACKAGE INSTALLATION

- The `CRANextra` repository is no longer a default repository on Windows: all the binary versions of packages from CRAN are now on CRAN, although `CRANextra` contains packages from Omegahat and elsewhere used by CRAN packages.
- Only vignettes sources in directory 'vignettes' are considered to be vignettes and hence indexed as such.
- In the 'DESCRIPTION' file,

  ```
  License: X11
  ```

  is no longer recognized as valid. Use '`MIT`' or '`BSD_2_clause`' instead, both of which need '`+ file LICENSE`'.
- For consistency, entries in '.Rinstignore' are now matched case-insensitively on all platforms.
- Help for S4 methods with very long signatures now tries harder to split the description in the '`Usage`' field to no more than 80 characters per line (some packages had over 120 characters).
- `R CMD INSTALL --build` (not Windows) now defaults to the internal `tar()` unless `R_INSTALL_TAR` is set.
- There is support for compiling C++11 code in packages on suitable platforms: see 'Writing R Extensions'.
- Fake installs now install the contents of directory 'inst': some packages use this to install e.g. C++ headers for use by other packages that are independent of the package itself. Option '`--no-inst`' can be used to get the previous behaviour.

DEBUGGING

- The behaviour of the code browser has been made more consistent, in part following the suggestions in PR#14985.
- Calls to `browser()` are now consistent with calls to the browser triggered by `debug()`, in that `Enter` will default to `n` rather than `c`.
- A new browser command `s` has been added, to "step into" function calls.
- A new browser command `f` has been added, to "finish" the current loop or function.
- Within the browser, the command `help` will display a short list of available commands.

UTILITIES

- Only vignettes sources in directory 'vignettes' are considered to be vignettes by `R CMD check`. That has been the preferred location since R 2.14.0 and is now obligatory.
- For consistency, `R CMD build` now matches entries in '.Rbuildignore' and 'vignettes/.install_extras' case-insensitively on all platforms (not just on Windows).

- checkFF() (called by R CMD check by default) can optionally check foreign function calls for consistency with the registered type and argument count. This is the default for R CMD check --as-cran or can be enabled by setting environment variable _R_CHECK_FF_CALLS_ to 'registration' (but is in any case suppressed by '--install=no'). Because this checks calls in which .NAME is an R object and not just a literal character string, some other problems are detected for such calls.

  Functions suppressForeignCheck() and dontCheck() have been added to allow package authors to suppress false positive reports.

- R CMD check --as-cran warns about a false value of the 'DESCRIPTION' field 'BuildVignettes' for Open Source packages, and ignores it. (An Open Source package needs to have complete sources for its vignettes which should be usable on a suitably well-equipped system).

- R CMD check --no-rebuild-vignettes is defunct:
  R CMD check --no-build-vignettes has been preferred since R 3.0.0.

- R CMD build --no-vignettes is defunct:
  R CMD build --no-build-vignettes has been preferred since R 3.0.0.

- R CMD Sweave and R CMD Stangle now process both Sweave and non-Sweave vignettes. The tools::buildVignette() function has been added to do the same tasks from within R.

- The flags returned by R CMD config --ldflags and (where installed) pkg-config --libs libR are now those needed to link a front-end against the (shared or static) R library.

- 'Sweave.sty' has a new option '[inconsolata]'.

- R CMD check customizations such as _R_CHECK_DEPENDS_ONLY_ make available packages only in 'LinkingTo' only for installation, and not for loading/runtime tests.

- tools::checkFF() reports on .C and .Fortran calls with DUP = FALSE if argument check_DUP is true. This is selected by R CMD check by default.

- R CMD check --use-gct can be tuned to garbage-collect less frequently using gctorture2() *via* the setting of environment variable _R_CHECK_GCT_N_.

- Where supported, tools::texi2dvi() limits the number of passes tried to 20.

C-LEVEL FACILITIES

- (Windows only) A function R_WaitEvent() has been added (with declaration in header'R.h') to block execution until the next event is received by R.

- Remapping in the 'Rmath.h' header can be suppressed by defining 'R_NO_REMAP_RMATH'.

- The remapping of rround() in header 'Rmath.h' has been removed: use fround() instead.

- ftrunc() in header 'Rmath.h' is now a wrapper for the C99 function trunc(), which might as well be used in C code: ftrunc() is still needed for portable C++ code.

- The never-documented remapping of prec() to fprec() in header 'Rmath.h' has been removed.

- The included LAPACK subset now contains ZGESDD and ZGELSD.

- The function LENGTH() now checks that it is only applied to vector arguments. However, in packages length() should be used. (In R itself LENGTH() is a macro without the function overhead of length().)

- Calls to `SET_VECTOR_ELT()` and `SET_STRING_ELT()` are now checked for indices which are in-range: several packages were writing one element beyond the allocated length.
- `allocVector3` has been added which allows custom allocators to be used for individual vector allocations.

DEPRECATED AND DEFUNCT

- `chol(pivot = TRUE,LINPACK = TRUE)` is defunct.

  Arguments `EISPACK` for `eigen()` and `LINPACK` for `chol()`, `chol2inv()`, `solve()` and `svd()` are ignored: LAPACK is always used.
- `.find.package()` and `.path.package()` are defunct: only the versions without the initial dot introduced in R 2.13.0 have ever been in the API.
- Partial matching when using the `$` operator *on data frames* now throws a warning and may become defunct in the future. If partial matching is intended, replace `foo$bar` by `foo[["bar",exact = FALSE]]`.
- The long-deprecated use of `\synopsis` in the 'Usage' section of '.Rd' files has been removed: such sections are now ignored (with a warning).
- `package.skeleton()`'s deprecated argument `namespace` has been removed.
- Many methods are no longer exported by package **stats**. They are all registered on their generic, which should be called rather than calling a method directly.
- Functions `readNEWS()` and `checkNEWS()` in package **tools** are defunct.
- `download.file(method = "lynx")` is deprecated.
- `.C(DUP = FALSE)` and `.Fortran(DUP = FALSE)` are now deprecated, and may be disabled in future versions of R. As their help has long said, `.Call()` is much preferred.

  `R CMD check` notes such usages (by default).
- The workaround of setting `R_OSX_VALGRIND` has been removed: it is not needed in current valgrind.

BUG FIXES

- Calling `lm.wfit()` with no non-zero weights gave an array-overrun in the Fortran code and a not very sensible answer. It is now special-cased with a simpler answer (no `qr` component).
- Error messages involving non-syntactic names (e.g., as produced by `` `\r` `` when that object does not exist) now encode the control characters. (Reported by Hadley Wickham.)
- `getGraphicsEvent()` caused 100% usage of one CPU in Windows. (PR#15500)
- `nls()` with no `start` argument may now work inside another function (scoping issue).
- `pbeta()` and similar work better for very large (billions) `ncp`.
- Where time zones have changed abbreviations over the years, the software tries to more consistently use the abbreviation appropriate to the time or if that is unknown, the current abbreviation. On some platforms where the C function `localtime` changed the `tzname` variables the reported abbreviation could have been that of the last time converted.
- `all.equal(list(1),identity)` now works.

- Bug fix for pushing viewports in **grid** (reported by JJ Allaire and Kevin Ushey).

  NOTE for anyone poking around within the graphics engine display list (despite the warnings not to) that this changes what is recorded by **grid** on the graphics engine display list.

- Extra checks have been added for unit resolution and conversion in **grid**, to catch instances of division-by-zero. This may introduce error messages in existing code and/or produce a different result in existing code (but only where a non-finite location or dimension may now become zero).

- Some bugs in TRE have been corrected by updating from the `git` repository. This allows R to be installed on some platforms for which this was a blocker (PR#15087 suggests Linux on ARM and HP-UX).

- `?` applied to a call to an S4 generic failed in several cases. (PR#15680)

- The implicit S4 generics for primitives with `...` in their argument list were incorrect. (PR#15690)

- Bug fixes to `methods::callGeneric()`. (PR#15691)

- The bug fix to `aggregrate()` in PR#15004 introduced a new bug in the case of no grouping variables. (PR#15699)

- In rare cases printing deeply nested lists overran a buffer by one byte and on a few platforms segfaulted. (PR#15679)

- The dendrogram method of `as.dendrogram()` was hidden accidentally, (PR#15703), and `order.dendrogram(d)` gave too much for a leaf `d`. (PR#15702)

- R would try to kill processes on exit that have pids ever used by a child process spawned by `mcparallel` even though the current process with that pid was not actually its child.

- `cophenetic()` applied to a `"dendrogram"` object sometimes incorrectly returned a `"Labels"` attribute with dimensions. (PR#15706)

- `printCoefmat()` called from quite a few `print()` methods now obeys small `getOption("width")` settings, line wrapping the '`"signif. codes"`' legend appropriately. (PR#15708)

- `model.matrix()` assumed that the stored dimnames for a matrix was `NULL` or length 2, but length 1 occurred.

- The clipping region for a device was sometimes used in base graphics before it was set.

CHANGES IN R 3.0.3

NEW FEATURES

- On Windows there is support for making '.texi' manuals using `texinfo` 5.0 or later: the setting is in file 'src/gnuwin32/MkRules.dist'.

  A packaging of the Perl script and modules for `texinfo` 5.2 has been made available at `http://www.stats.ox.ac.uk/pub/Rtools/`.

- `write.table()` now handles matrices of 2^{31} or more elements, for those with large amounts of patience and disc space.

- There is a new function, `La_version()`, to report the version of LAPACK in use.

- The HTML version of 'An Introduction to R' now has links to PNG versions of the figures.

- There is some support to produce manuals in ebook formats. (See 'doc/manual/Makefile'. Suggested by Mauro Cavalcanti.)
- On a Unix-alike `Sys.timezone()` returns `NA` if the environment variable `TZ` is unset, to distinguish it from an empty string which on some OSes means the '`UTC`' time zone.
- The backtick may now be escaped in strings, to allow names containing them to be constructed, e.g. `` `\`` ``. (PR#15621)
- `read.table()`, `readLines()` and `scan()` now warn when an embedded nul is found in the input. (Related to PR#15625 which was puzzled by the behaviour in this unsupported case.)
- (Windows only.) `file.symlink()` works around the undocumented restriction of the Windows system call to backslashes. (Wish of PR#15631.)
- `KalmanForecast(fast = FALSE)` is now the default, and the help contains an example of how `fast = TRUE` can be used in this version. (The usage will change in 3.1.0.)
- `strptime()` now checks the locale only when locale-specific formats are used and caches the locale in use: this can halve the time taken on OSes with slow system functions (e.g., OS X).
- `strptime()` and the `format()` methods for classes `"POSIXct"`, `"POSIXlt"` and `"Date"` recognize strings with marked encodings: this allows, for example, UTF-8 French month names to be read on (French) Windows.
- `iconv(to = "utf8")` is now accepted on all platforms (some implementations did already, but GNU **libiconv** did not: however converted strings were not marked as being in UTF-8). The official name, `"UTF-8"` is still preferred.
- `available.packages()` is better protected against corrupt metadata files. (A recurring problem with Debian package **shogun-r**: PR#14713.)
- Finalizers are marked to be run at garbage collection, but run only at a somewhat safer later time (when interrupts are checked). This circumvents some problems with finalizers running arbitrary code during garbage collection (the known instances being running `options()` and (C-level) `path.expand()` re-entrantly).

INSTALLATION and INCLUDED SOFTWARE

- The included version of PCRE has been updated to 8.34. This fixes bugs and makes the behaviour closer to Perl 5.18. In particular, the concept of 'space' includes '`VT`' and hence agrees with POSIX's.

PACKAGE INSTALLATION

- The new field '`SysDataCompression`' in the 'DESCRIPTION' file allows user control over the compression used for 'sysdata.rda' objects in the lazy-load database.
- `install.packages(dependencies = value)` for `value = NA` (the default) or `value = TRUE` omits packages only in `LinkingTo` for binary package installs.

C-LEVEL FACILITIES

- The long undocumented remapping of `rround()` to `Rf_fround()` in header 'Rmath.h' is now formally deprecated: use `fround()` directly.
- Remapping of `prec()` and `trunc()` in the 'Rmath.h' header has been disabled in C++ code (it has caused breakage with `libc++` headers).

BUG FIXES

- `getParseData()` truncated the imaginary part of complex number constants. (Reported by Yihui Xie.)
- `dbeta(x,a,b)` with `a` or `b` within a factor of 2 of the largest representable number could infinite-loop. (Reported by Ioannis Kosmidis.)
- `provideDimnames()` failed for arrays with a 0 dimension. (PR#15465)
- `rbind()` and `cbind()` did not handle list objects correctly. (PR#15468)
- `replayPlot()` now checks if it is replaying a plot from the same session.
- `rasterImage()` and `grid.raster()` now give error on an empty (zero-length) raster. (Reported by Ben North.)
- `plot.lm()` would sometimes scramble the labels in plot type 5. (PR#15458 and PR#14837)
- `min()` did not handle `NA_character_` values properly. (Reported by Magnus Thor Torfason.)
- (Windows only.) `readRegistry()` would duplicate default values for keys. (PR#15455)
- `str(...,strict.width = "cut")` did not handle it properly when more than one line needed to be cut. (Reported by Gerrit Eichner.)
- Removing subclass back-references when S4 classes were removed or their namespace unloaded had several bugs (e.g., PR#15481).
- `aggregate()` could fail when there were too many levels present in the `by` argument. (PR#15004)
- `namespaceImportFrom()` needed to detect primitive functions when checking for duplicated imports (reported by Karl Forner).
- `getGraphicsEvent()` did not exit when a user closed the graphics window. (PR#15208)
- Errors in vignettes were not always captured and displayed properly. (PR#15495)
- `contour()` could fail when dealing with extremely small z values. (PR#15454)
- Several functions did not handle zero-length vectors properly, including `browseEnv()`, `format()`, `gl()`, `relist()` and `summary.data.frame()`. (E.g., PR#15499)
- `Sweave()` did not restore the R output to the console if it was interrupted by a user in the middle of evaluating a code chunk. (Reported by Michael Sumner.)
- Fake installs of packages with vignettes work again.
- Illegal characters in the input caused `parse()` (and thus `source()`) to segfault. (PR#15518)
- The nonsensical use of `nmax = 1` in `duplicated()` or `unique()` is now silently ignored.
- `qcauchy(p,*)` is now fully accurate even when p is very close to 1. (PR#15521)
- The `validmu()` and `valideta()` functions in the standard `glm()` families now also report non-finite values, rather than failing.
- Saved vignette results (in a '.Rout.save' file) were not being compared to the new ones during `R CMD check`.
- Double-clicking outside of the list box (e.g., on the scrollbar) of a Tk listbox widget generated by `tk_select.list()` no longer causes the window to close. (PR#15407)

- Improved handling of edge cases in `parallel::splitindices()`. (PR#15552)
- HTML display of results from `help.search()` and `??` sometimes contained badly constructed links.
- `c()` and related functions such as `unlist()` converted raw vectors to invalid logical vectors. (PR#15535)
- (Windows only) When a call to `system2()` specified one of `stdin`, `stdout` or `stderr` to be a file, but the command was not found (e.g., it contained its arguments, or the program was not on the `PATH`), it left the file open and unusable until R terminated. (Reported by Mathew McLean.)
- The `bmp()` device was not recording `res = NA` correctly: it is now recorded as 72 ppi.
- Several potential problems with compiler-specific behaviour have been identified using the 'Undefined Behaviour Sanitizer' in conjunction with the `clang` compiler.
- `hcl()` now honours `NA` inputs (previously they were mapped to black).
- Some translations in base packages were being looked up in the main catalog rather than that for the package.
- As a result of the 3.0.2 change about 'the last second before the epoch', most conversions which should have given `NA` returned that time. (The platforms affected include Linux and OS X, but not Windows nor Solaris.)
- `rowsum()` has more support for matrices and data frames with 2^{31} or more elements. (PR#15587)
- `predict(<lm object>,interval = "confidence",scale = <something>)` now works. (PR#15564)
- The bug fix in 3.0.2 for PR#15411 was too aggressive, and sometimes removed spaces that should not have been removed. (PR#15583)
- Running R code in a **tcltk** callback failed to set the busy flag, which will be needed to tell OS X not to 'App Nap'.
- The code for date-times before 1902 assumed that the offset from GMT in 1902 was a whole number of minutes: that was not true of Paris (as recorded on some platforms).
- Using `Sys.setlocale` to set `LC_NUMERIC` to `"C"` (to restore the sane behavior) no longer gives a warning.
- `deparse()` now deparses complex vectors in a way that re-parses to the original values. (PR#15534, patch based on code submitted by Alex Bertram.)
- In some extreme cases (more than 10^{15}) integer inputs to `dpqrxxx()` functions might have been rounded up by one (with a warning about being non-integer). (PR#15624)
- Plotting symbol `pch = 14` had the triangle upside down on some devices (typically screen devices). The triangle is supposed to be point up. (Reported by Bill Venables.)
- `getSrcref()` did not work on method definitions if `rematchDefinition()` had been used.
- `KalmanForecast(fast = FALSE)` reported a (harmless) stack imbalance.
- The count of observations used by `KalmanRun()` did not take missing values into account.

- In locales where the abbreviated name of one month is a partial match for the full name of a later one, the `%B` format in `strptime()` could fail. An example was French on OS X, where 'juin' is abbreviated to 'jui' and partially matches `juillet`. Similarly for weekday names.
- `pbeta(x,a,b,log.p = TRUE)` sometimes underflowed to zero for very small and very differently sized `a`, `b`. (PR#15641)
- `approx()` and `approxfun()` now handle infinite values with the `"constant"` method. (PR#15655)
- `stripchart()` again respects reversed limits in `xlim` and `ylim`. (PR#15664)

CHANGES IN R 3.0.2

NEW FEATURES

- The 'NEWS' files have been re-organized.

  This file contains news for R >= 3.0.0: news for the 0.x.y, 1.x.y and 2.x.y releases is in files 'NEWS.0', 'NEWS.1' and 'NEWS.2'. The latter files are now installed when R is installed. An HTML version of news from 2.10.0 to 2.15.3 is available as 'doc/html/NEWS.2.html'.
- `sum()` for integer arguments now uses an integer accumulator of at least 64 bits and so will be more accurate in the very rare case that a cumulative sum exceeds 2^{53} (necessarily summing more than 4 million elements).
- The `example()` and `tools::Rd2ex()` functions now have parameters to allow them to ignore `\dontrun` markup in examples. (Suggested by Peter Solymos.)
- `str(x)` is considerably faster for very large lists, or factors with 100,000 levels, the latter as in PR#15337.
- `col2rgb()` now converts factors to character strings not integer codes (suggested by Bryan Hanson).
- `tail(warnings())` now works, *via* the new `` `[` `` method.
- There is now support for the LaTeX style file 'zi4.sty' which has in some distributions replaced 'inconsolata.sty'.
- `unlist(x)` now typically returns all non-list `x`s unchanged, not just the "vector" ones. Consequently, `format(lst)` now also works when the list `lst` has non-vector elements.
- The `tools::getVignetteInfo()` function has been added to give information about installed vignettes.
- New `assertCondition()`, etc. utilities in **tools**, useful for testing.
- Profiling now records non-inlined calls from byte-compiled code to `BUILTIN` functions.
- Various functions in **stats** and elsewhere that use non-standard evaluation are now more careful to follow the namespace scoping rules. E.g., `stats::lm()` can now find `stats::model.frame()` even if **stats** is not on the search path or if some package defines a function of that name.
- If an invalid/corrupt `.Random.seed` object is encountered in the workspace it is ignored with a warning rather than giving an error. (This allows R itself to rely on a working RNG, e.g. to choose a random port.)
- `seq()` and `seq.int()` give more explicit error messages if called with invalid (e.g., `NaN`) inputs.

- When `parse()` finds a syntax error, it now makes partial parse information available up to the location of the error. (Request of Reijo Sund.)
- Methods invoked by `NextMethod()` had a different dynamic parent to the generic. This was causing trouble where S3 methods invoked *via* lazy evaluation could lose track of their generic. (PR#15267)
- Code for the negative binomial distribution now treats the case `size == 0` as a one-point distribution at zero.
- `abbreviate()` handles without warning non-ASCII input strings which require no abbreviation.
- `read.dcf()` no longer has a limit of 8191 bytes per line. (Wish of PR#15250.)
- `formatC(x)` no longer copies the class of `x` to the result, to avoid misuse creating invalid objects as in PR#15303. A warning is given if a class is discarded.
- Dataset `npk` has been copied from **MASS** to allow more tests to be run without recommended packages being installed.
- The initialization of the regression coefficients for non-degenerate differenced models in `arima()` has been changed and in some examples avoids a local maximum. (PR#15396)
- `termplot()` now has an argument `transform.x` to control the display of individual terms in the plot. (PR#15329)
- `format()` now supports `digits = 0`, to display `nsmall` decimal places.
- There is a new read-only `par()` parameter called `"page"`, which returns a logical value indicating whether the next `plot.new()` call will start a new page.
- Processing Sweave and Rd documents to PDF now renders backticks and single quotes better in several instances, including in '`\code`' and '`\samp`' expressions.
- `utils::modifyList()` gets a new argument `keep.null` allowing `NULL` components in the replacement to be retained, instead of causing corresponding components to be deleted.
- `tools::pkgVignettes()` gains argument `check`; if set to `TRUE`, it will warn when it appears a vignette requests a non-existent vignette engine.

UTILITIES

- `R CMD check --as-cran` checks the line widths in usage and examples sections of the package Rd files.
- `R CMD check --as-cran` now implies '`--timings`'.
- `R CMD check` looks for command `gfile` if a suitable `file` is not found. (Although `file` is not from GNU, OpenCSW on Solaris installs it as `gfile`.)
- `R CMD build` (with the internal `tar`) checks the permissions of 'configure' and 'cleanup' files and adds execute permission to the recorded permissions for these files if needed, with a warning. This is useful on OSes and file systems which do not support execute permissions (notably, on Windows).
- `R CMD build` now weaves and tangles all vignettes, so suggested packages are not required during package installation if the source tarball was prepared with current `R CMD build`.

- `checkFF()` (used by `R CMD check`) does a better job of detecting calls from other packages, including not reporting those where a function has been copied from another namespace (e.g., as a default method). It now reports calls where `.NAME` is a symbol registered in another package.
- On Unix-alike systems, `R CMD INSTALL` now installs packages group writably whenever the library (`lib.loc`) is group writable. Hence, `update.packages()` works for other group members (suggested originally and from a patch by Dirk Eddelbuettel).
- `R CMD javareconf` now supports the use of symbolic links for `JAVA_HOME` on platforms which have `realpath`. So it is now possible to use

  ```
  R CMD javareconf JAVA_HOME=/usr/lib/jvm/java-1.7.0
  ```

  on a Linux system and record that value rather than the frequently-changing full path such as '/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.25.x86_64'.
- (Windows only.) `Rscript -e` requires a non-empty argument for consistency with Unix versions of R. (Also `Rterm -e` and `R -e`.)
- `R CMD check` does more thorough checking of declared packages and namespaces. It reports
  - packages declared in more than one of the 'Depends', 'Imports', 'Suggests' and 'Enhances' fields of the 'DESCRIPTION' file.
  - namespaces declared in 'Imports' but not imported from, neither in the 'NAMESPACE' file nor using the `::` nor `:::` operators.
  - packages which are used in `library()` or `requires()` calls in the R code but were already put on the search path *via* 'Depends'.
  - packages declared in 'Depends' not imported *via* the 'NAMESPACE' file (except the standard packages). Objects used from 'Depends' packages should be imported to avoid conflicts and to allow correct operation when the namespace is loaded but not attached.
  - objects imported *via* `:::` calls where `::` would do.
  - objects imported by `::` which are not exported.
  - objects imported by `:::` calls which do not exist.

  See 'Writing R Extensions' for good practice.
- `R CMD check` optionally checks for non-standard top-level files and directories (which are often mistakes): this is enabled for '`--as-cran`'.
- LaTeX style file `upquote.sty` is no longer included (the version was several years old): it is no longer used in R. A much later version is commonly included in LaTeX distributions but does not play well with the `ae` fonts which are the default for Sweave vignettes.
- `R CMD build` makes more use of the 'build' sub-directory of package sources, for example to record information about the vignettes.
- `R CMD check` analyses `:::` calls.

INSTALLATION and INCLUDED SOFTWARE

- The macros used for the texinfo manuals have been changed to work better with the incompatible changes made in `texinfo 5.x`.
- The minimum version for a system `xz` library is now 5.0.3 (was 4.999). This is in part to avoid 5.0.2, which can compress in ways other versions cannot decompress.

- The included version of PCRE has been updated to 8.33.
- The included version of `zlib` has been updated to 1.2.8, a bug-fix release.
- The included version of xz utils's `liblzma` has been updated to 5.0.5.
- Since `javareconf` (see above) is used when R is installed, a stable link for `JAVA_HOME` can be supplied then.
- Configuring with '`--disable-byte-compilation`' will override the 'DESCRIPTION' files of recommended packages, which typically require byte-compilation.
- More of the installation and checking process will work even when `TMPDIR` is set to a path containing spaces, but this is not recommended and external software (such as `texi2dvi`) may fail.

PACKAGE INSTALLATION

- Installation is aborted immediately if a `LinkingTo` package is not installed.
- `R CMD INSTALL` has a new option `--no-byte-compile` which will override a '`ByteCompile`' field in the package's 'DESCRIPTION' file.
- License '`BSD`' is deprecated: use '`BSD_3_clause`' or '`BSD_2_clause`' instead.

  License '`X11`' is deprecated: use '`MIT`' or '`BSD_2_clause`' instead.
- Version requirements for `LinkingTo` packages are now recognized: they are checked at installation. (Fields with version requirements were previously silently ignored.)
- The limit of 500 `S3method` entries in a `NAMESPACE` file has been removed.
- The default 'version' of Bioconductor for its packages has been changed to the upcoming '`2.13`', but this can be set by the environment variable `R_BIOC_VERSION` when R is installed.

C-LEVEL FACILITIES

- 'Rdefines.h' has been tweaked so it can be included in C++ code after 'R_ext/Boolean.h' (which is included by 'R.h').

  Note that 'Rdefines.h' is not kept up-to-date, and 'Rinternals.h' is preferred for new code.
- `eval` and `applyClosure` are now protected against package code supplying an invalid `rho`.

DEPRECATED AND DEFUNCT

- The unused `namespace` argument to `package.skeleton()` is now formally deprecated and will be removed in R 3.1.0.
- `plclust()` is deprecated: use the `plot()` method for class `"hclust"` instead.
- Functions `readNEWS()` and `checkNEWS()` in package **tools** are deprecated (and they have not worked with current 'NEWS' files for a long time).

DOCUMENTATION

- 'An Introduction to R' has a new chapter on using R as a scripting language including interacting with the OS.

BUG FIXES

- `help.request()` could not determine the current version of R on CRAN. (PR#15241)
- On Windows, `file.info()` failed on root directories unless the path was terminated with an explicit `"."`. (PR#15302)
- The `regmatches<-()` replacement function mishandled results coming from `regexpr()`. (PR#15311)
- The help for `setClass()` and `representation()` still suggested the deprecated argument `representation=`. (PR#15312)
- `R CMD config` failed in an installed build of R 3.0.1 (only) when a sub-architecture was used. (Reported by Berwin Turlach.)
- On Windows, the installer modified the 'etc/Rconsole' and 'etc/Rprofile.site' files even when default options were chosen, so the MD5 sums did not refer to the installed versions. (Reported by Tal Galili.)
- `plot(hclust(),cex =)` respects `cex` again (and possibly others similarly). (Reported by Peter Langfelder.)
- If multiple packages were checked by `R CMD check`, and one was written for a different OS, it would set `--no-install` for all following packages as well as itself.
- `qr.coef()` and related functions did not properly coerce real vectors to complex when necessary. (PR#15332)
- `ftable(a)` now fixes up empty `dimnames` such that the result is printable.
- `package.skeleton()` was not starting its search for function objects in the correct place if `environment` was supplied. (Reported by Karl Forner.)
- Parsing code was changing the length field of vectors and confusing the memory manager. (PR#15345)
- The Fortran routine `ZHER2K` in the reference BLAS had a comment-out bug in two places. This caused trouble with `eigen()` for Hermitian matrices. (PR#15345 and report from Robin Hankin)
- `vignette()` and `browseVignettes()` did not display non-Sweave vignettes properly.
- Two warning/error messages have been corrected: the (optional) warning produced by a partial name match with a pairlist, the error message from a zero-length argument to the `:` operator. (Found by Radford Neal; PR#15358, PR#15356)
- `svd()` returned `NULL` rather than omitting components as documented. (Found by Radford Neal; PR#15360)
- `mclapply()` and `mcparallel()` with `silent = TRUE` could break a process that uses `stdout` output unguarded against broken pipes (e.g., `zip` will fail silently). To work around such issues, they now replace `stdout` with a descriptor pointed to '/dev/null' instead. For this purpose, internal `closeStdout` and `closeStderr` functions have gained the `to.null` flag.
- `log()`, `signif()` and `round()` now raise an error if a single named argument is not named `x`. (PR#15361)
- `deparse()` now deparses raw vectors in a form that is syntactically correct. (PR#15369)
- The `jpeg` driver in Sweave created a JPEG file, but gave it a '.png' extension. (PR#15370)
- Deparsing of infix operators with named arguments is improved. (PR#15350)

- `mget()`, `seq.int()` and `numericDeriv()` did not duplicate arguments properly. (PR#15352, PR#15353, PR#15354)
- `kmeans(algorithm = "Hartigan-Wong")` now always stops iterating in the QTran stage. (PR#15364).
- `read.dcf()` re-allocated incorrectly and so could segfault when called on a file with lines of more than 100 bytes.
- On systems where `mktime()` does not set `errno`, the last second before the epoch could not be converted from `POSIXlt` to `POSIXct`. (Reported by Bill Dunlap.)
- `add1.glm()` miscalculated F-statistics when df > 1. (Bill Dunlap, PR#15386).
- `stem()` now discards infinite inputs rather than hanging. (PR#15376)
- The parser now enforces C99 syntax for floating point hexadecimal constants (e.g., `0x1.1p0`), rather than returning unintended values for malformed constants. (PR#15234)
- `model.matrix()` now works with very long LHS names (more than 500 bytes). (PR#15377)
- `integrate()` reverts to the pre-2.12.0 behaviour: from 2.12.0 to 3.0.1 it sometimes failed to achieve the requested tolerance and reported error estimates that were exceeded. (PR#15219)
- `strptime()` now handles '`%W`' fields with value 0. (PR#15915)
- R is now better protected against people trying to interact with the console in startup code. (PR#15325)
- Subsetting 1D arrays often lost dimnames (PR#15301).
- Unary `+` on a logical vector did not coerce to integer, although unary `-` did.
- `na.omit()` and `na.exclude()` added a row to a zero-row data frame. (PR#15399)
- All the (where necessary cut-down) vignettes are installed if R was configured with '`--without-recommended-packages`'.
- `source()` did not display filenames when reporting syntax errors.
- Syntax error reports misplaced the caret pointing out the bad token.
- (Windows only) Starting R with `R` (instead of `Rterm` or `Rgui`) would lose any zero-length strings from the command line arguments. (PR#15406)
- Errors in the encoding specified on the command line *via* `--encoding=foo` were not handled properly. (PR#15405)
- If `x` is a symbol, `is.vector(x,"name")` now returns `TRUE`, since `"name"` and `"symbol"` should be synonyms. (Reported by Hervé Pagès.)
- `R CMD rtags` works on platforms (such as OS X) with a XSI-conformant shell command `echo`. (PR#15231)
- `is.unsorted(NA)` returns false as documented (rather than `NA`).
- `R CMD LINK` did not know about sub-architectures.
- `system()` and `system2()` are better protected against users who misguidedly have spaces in the temporary directory path.
- `file.show()` and `edit()` are now more likely to work on file paths containing spaces. (Where external utilities are used, not the norm on Windows nor in `R.app` which should previously have worked.)

- Packages using the **methods** package are more likely to work when they import it but it is not attached. (Several parts of its C code were looking for its R functions on the search path rather than in its namespace.)
- `lgamma(-x)` is no longer `NaN` for very small x.
- (Windows) `system2()` now respects specifying `stdout` and `stderr` as files if called from `Rgui`. (PR#15393)
- Closing an `x11()` device whilst `locator()` or `identify()` is in progress no longer hangs R. (PR#15253)
- `list.dirs(full.names = FALSE)` was not implemented. (PR#15170)
- `format()` sometimes added unnecessary spaces. (PR#15411)
- `all.equal(check.names = FALSE)` would ignore the request to ignore the names and would check them as attributes.
- The symbol set by `tools::Rd2txt_options(itemBullet=)` was not respected in some locales. (PR#15435)
- `mcMap()` was not exported by package **parallel**. (PR#15439)
- `plot()` for `TukeyHSD` objects did not balance `dev.hold()` and `dev.flush()` calls on multi-page plots. (PR#15449)

CHANGES IN R 3.0.1

NEW FEATURES

- `chooseCRANmirror()` and `chooseBioCmirror()` gain an `ind` argument (like `setRepositories()`).
- `mcparallel` has a new argument `mc.interactive` which can modify the interactive flag in the child process. The new default is `FALSE` which makes child processes non-interactive by default (this prevents lock-ups due to children waiting for interactive input).
- `scan()` now warns when end-of-file occurs within a quoted string.
- `count.fields()` is now consistent with `scan()` in its handling of newlines in quoted strings. Instead of triggering an error, this results in the current line receiving `NA` as the field count, with the next line getting the total count of the two lines.
- The default method of `image()` will plot axes of the class of `xlim` and `ylim` (and hence of `x` and `y` if there is a suitable `range()` method). Based on a suggestion of Michael Sumner.
- `load()` now has a `verbose` argument for debugging support, to print the names of objects just before loading them.
- When loading a serialized object encounters a reference to a namespace which cannot be loaded, this is replaced by a reference to the global environment, with a warning.
- `pairs()` gains a `line.main` option for title placement.
- The remaining instances in which serialization to a raw vector was limited to 2GB have been unlimited on a 64-bit platform, and in most cases serialization to a vector of more than 1GB will be substantially faster.

UTILITIES

- `R CMD config` now make use of personal 'Makevars' files under '~/.R' and a site file 'Makevars.site', in the same way as `R CMD SHLIB` and `R CMD INSTALL`. This makes the utility more useful in package `configure` scripts.

  On Windows finding the personal files may require the environment variable `HOME` set.

  The old behaviour can be obtained with the new options '`--no-user-files`' and '`--no-site-files`'.

PACKAGE INSTALLATION

- Alternatives to the site and user customization files 'Makevars.site' and '~/.R/Makevars' can be specified *via* the environment variables `R_MAKEVARS_SITE` and `R_MAKEVARS_USER` respectively. These can be used to suppress the use of the default files by setting an empty value (where possible) or a non-existent path.

BUG FIXES

- `sys.source()` did not report error locations when `keep.source = TRUE`.
- `as.POSIXct.numeric` was coercing `origin` using the `tz` argument and not `"GMT"` as documented (PR#14973).
- The active binding to assign fields in reference classes has been cleaned up to reduce dependence on the class' package environment, also fixing bug in initializing read-only fields (inspired by a report from Hadley Wickham).
- `str(d)` no longer gives an error when `names(d)` contain illegal multibyte strings (PR#15247).
- Profiling of built-in functions with `line.profiling= TRUE` did not record the line from which they were called.
- `citation(pkg)` dropped the header and footer specified in the 'CITATION' file (PR#15257).
- Quotes were handled differently when reading the first line and reading the rest, so `read.table()` misread some files that contained quote characters (PR#15245).
- `cat()` with `sep` a character vector of length greater than one and more than one argument was using separators inconsistently (PR#15261).
- On Windows in R 3.0.0, `savePlot()` failed because of an incorrect check on the argument count.
- `unzip(list = TRUE)` returned `Names` as a factor and not a character vector (as documented) for the internal method. (Noticed by Sean O'Riordain.)
- `contourLines()` now checks more comprehensively for conformance of its `x`, `y` and `z` arguments (it was used incorrectly in package **R2G2**).
- Saved graphics display lists are R version-specific. Attempting to load workspaces containing them (or some other version-specific objects) aborted the load in R 3.0.0 and earlier; now it does a partial load and generates a warning instead.
- In R 3.0.0, `identify()` and `locator()` did not record information correctly, so replaying a graph (e.g., by copying it to another device) would fail. (PR#15271)
- Calling `file.copy()` or `dirname()` with the invalid input `""` (which was being used in packages, despite not being a file path) could have caused a segfault.

  `dirname("")` is now `""` rather than `"."` (unless it segfaulted).

- `supsmu()` could read/write outside its input vectors for very short inputs (seen in package **rms** for `n = 4`).
- `as.dendrogram()`'s `hclust` method uses less memory and hence gets considerably faster for large (n ~ 1000) clusterings, thanks to Daniel Müllner. (PR#15174)
- The return value when all workers failed from `parallel::mclapply(mc.preschedule = TRUE)` was a list of strings and not of error objects. (Spotted by Karl Forner and Bernd Bischl.)
- In R 3.0.0, when `help()` found multiple pages with the same alias, the HTML display of all the selections was not produced. (PR#15282)
- `splinefun(method="monoH.FC")` now produces a function with first argument named `x` and allows `deriv=3`, as documented. (PR#15273)
- `summaryRprof()` would only read the first `chunksize` lines of an `Rprof` file produced with `line.profiling=TRUE`. By default, this is the first 100 seconds. (PR#15288)
- `lsfit()` produced an incorrect error message when argument `x` had more columns than rows or `x` had a different number of rows than `y`. (Spotted by Renaud Gaujoux.)
- Binary operations on equal length vectors copied the class name from the second operand when the first had no class name, but did not set the object bit. (PR#15299)
- The `trace()` method for reference generator objects failed after those objects became function definitions.
- `write.table()` did not check that factors were constructed correctly, and so caused a segment fault when writing bad ones. (PR#15300)
- The internal HTTP server no longer chokes on POST requests without body. It will also pass-through other request types for custom handlers (with the method stored in Request-Method header) instead of failing.

CHANGES IN R 3.0.0

SIGNIFICANT USER-VISIBLE CHANGES

- Packages need to be (re-)installed under this version (3.0.0) of R.
- There is a subtle change in behaviour for numeric index values 2^{31} and larger. These never used to be legitimate and so were treated as `NA`, sometimes with a warning. They are now legal for long vectors so there is no longer a warning, and `x[2^31] <-y` will now extend the vector on a 64-bit platform and give an error on a 32-bit one.
- It is now possible for 64-bit builds to allocate amounts of memory limited only by the OS. It may be wise to use OS facilities (e.g., `ulimit` in a `bash` shell, `limit` in `csh`), to set limits on overall memory consumption of an R process, particularly in a multi-user environment. A number of packages need a limit of at least 4GB of virtual memory to load.

  64-bit Windows builds of R are by default limited in memory usage to the amount of RAM installed: this limit can be changed by command-line option '`--max-mem-size`' or setting environment variable `R_MAX_MEM_SIZE`.
- Negative numbers for colours are consistently an error: previously they were sometimes taken as transparent, sometimes mapped into the current palette and sometimes an error.

NEW FEATURES

- `identical()` has a new argument, `ignore.environment`, used when comparing functions (with default `FALSE` as before).
- There is a new option, `options(CBoundsCheck=)`, which controls how `.C()` and `.Fortran()` pass arguments to compiled code. If true (which can be enabled by setting the environment variable `R_C_BOUNDS_CHECK` to 'yes'), raw, integer, double and complex arguments are always copied, and checked for writing off either end of the array on return from the compiled code (when a second copy is made). This also checks individual elements of character vectors passed to `.C()`.

  This is not intended for routine use, but can be very helpful in finding segfaults in package code.
- In `layout()`, the limits on the grid size have been raised (again).
- New simple `provideDimnames()` utility function.
- Where methods for `length()` return a double value which is representable as an integer (as often happens for package **Matrix**), this is converted to an integer.
- Matrix indexing of data frames by two-column numeric indices is now supported for replacement as well as extraction.
- `setNames()` now has a default for its `object` argument, useful for a character result.
- `StructTS()` has a revised additive constant in the `loglik` component of the result: the previous definition is returned as the `loglik0` component. However, the help page has always warned of a lack of comparability of log-likelihoods for non-stationary models. (Suggested by Jouni Helske.)
- The logic in `aggregate.formula()` has been revised. It is now possible to use a formula stored in a variable; previously, it had to be given explicitly in the function call.
- `install.packages()` has a new argument `quiet` to reduce the amount of output shown.
- Setting an element of the graphics argument `lwd` to a negative or infinite value is now an error. Lines corresponding to elements with values `NA` or `NaN` are silently omitted.

  Previously the behaviour was device-dependent.
- Setting graphical parameters `cex`, `col`, `lty`, `lwd` and `pch` in `par()` now requires a length-one argument. Previously some silently took the first element of a longer vector, but not always when documented to do so.
- `Sys.which()` when used with inputs which would be unsafe in a shell (e.g., absolute paths containing spaces) now uses appropriate quoting.
- `as.tclObj()` has been extended to handle raw vectors. Previously, it only worked in the other direction. (Contributed by Charlie Friedemann, PR#14939.)
- New functions `cite()` and `citeNatbib()` have been added, to allow generation of in-text citations from `"bibentry"` objects. A `cite()` function may be added to `bibstyle()` environments.
- A `sort()` method has been added for `"bibentry"` objects.
- The `bibstyle()` function now defaults to setting the default bibliography style. The `getBibstyle()` function has been added to report the name of the current default style.
- `scatter.smooth()` now has an argument `lpars` to pass arguments to `lines()`.

- `pairs()` has a new `log` argument, to allow some or all variables to be plotted on logarithmic scale. (In part, wish of PR#14919.)
- `split()` gains a `sep` argument.
- `termplot()` does a better job when given a model with interactions (and no longer attempts to plot interaction terms).
- The parser now incorporates code from Romain Francois' **parser** package, to support more detailed computation on the code, such as syntax highlighting, comment-based documentation, etc. Functions `getParseData()` and `getParseText()` access the data.
- There is a new function `rep_len()` analogous to `rep.int()` for when speed is required (and names are not).
- The undocumented use `rep(NULL,length.out = n)` for `n >0` (which returns `NULL`) now gives a warning.
- `demo()` gains an `encoding` argument for those packages with non-ASCII demos: it defaults to the package encoding where there is one.
- `strwrap()` converts inputs with a marked encoding to the current locale: previously it made some attempt to pass through as bytes inputs invalid in the current locale.
- Specifying both `rate` and `scale` to `[dpqr]gamma` is a warning (if they are essentially the same value) or an error.
- `merge()` works in more cases where the data frames include matrices. (Wish of PR#14974.)
- `optimize()` and `uniroot()` no longer use a shared parameter object across calls. (`nlm()`, `nlminb()` and `optim()` with numerical derivatives still do, as documented.)
- The `all.equal()` method for date-times is now documented: times are regarded as equal (by default) if they differ by up to 1 msec.
- `duplicated()` and `unique()` gain a `nmax` argument which can be used to make them much more efficient when it is known that there are only a small number of unique entries. This is done automatically for factors.
- Functions `rbinom()`, `rgeom()`, `rhyper()`, `rpois()`, `rnbinom()`, `rsignrank()` and `rwilcox()` now return integer (not double) vectors. This halves the storage requirements for large simulations.
- `sort()`, `sort.int()` and `sort.list()` now use radix sorting for factors of less than 100,000 levels when `method` is not supplied. So does `order()` if called with a single factor, unless `na.last = NA`.
- `diag()` as used to generate a diagonal matrix has been re-written in C for speed and less memory usage. It now forces the result to be numeric in the case `diag(x)` since it is said to have 'zero off-diagonal entries'.
- `backsolve()` (and `forwardsolve()`) are now internal functions, for speed and support for large matrices.
- More matrix algebra functions (e.g., `chol()` and `solve()`) accept logical matrices (and coerce to numeric).
- `sample.int()` has some support for $n \geq 2^{31}$: see its help for the limitations.

  A different algorithm is used for `(n,size,replace = FALSE,prob = NULL)` for `n >1e7` and `size <= n/2`. This is much faster and uses less memory, but does give different results.

- `approxfun()` and `splinefun()` now return a wrapper to an internal function in the **stats** namespace rather than a `.C()` or `.Call()` call. This is more likely to work if the function is saved and used in a different session.
- The functions `.C()`, `.Call()`, `.External()` and `.Fortran()` now give an error (rather than a warning) if called with a named first argument.
- `Sweave()` by default now reports the locations in the source file(s) of each chunk.
- `clearPushBack()` is now a documented interface to a long-existing internal call.
- `aspell()` gains filters for R code, Debian Control Format and message catalog files, and support for R level dictionaries. In addition, package **utils** now provides functions `aspell_package_R_files()` and `aspell_package_C_files()` for spell checking R and C level message strings in packages.
- `bibentry()` gains some support for "incomplete" entries with a '`crossref`' field.
- `gray()` and `gray.colors()` finally allow `alpha` to be specified.
- `monthplot()` gains parameters to control the look of the reference lines. (Suggestion of Ian McLeod.)
- Added support for new `%~%` relation ("is distributed as") in plotmath.
- `domain = NA` is accepted by `gettext()` and `ngettext()`, analogously to `stop()` etc.
- `termplot()` gains a new argument `plot = FALSE` which returns information to allow the plots to be modified for use as part of other plots, but does not plot them. (Contributed by Terry Therneau, PR#15076.)
- `quartz.save()`, formerly an undocumented part of `R.app`, is now available to copy a device to a `quartz()` device. `dev.copy2pdf()` optionally does this for PDF output: `quartz.save()` defaults to PNG.
- The default method of `pairs()` now allows `text.panel = NULL` and the use of `<foo>.panel = NULL` is now documented.
- `setRefClass()` and `getRefClass()` now return class generator functions, similar to `setClass()`, but still with the reference fields and methods as before (suggestion of Romain Francois).
- New functions `bitwNot()`, `bitwAnd()`, `bitwOr()` and `bitwXor()`, using the internal interfaces previously used for classes `"octmode"` and `"hexmode"`.

  Also `bitwShiftL()` and `bitwShiftR()` for shifting bits in elements of integer vectors.
- New option `"deparse.cutoff"` to control the deparsing of language objects such as calls and formulae when printing. (Suggested by a comment of Sarah Goslee.)
- `colors()` gains an argument `distinct`.
- New `demo(colors)` and `demo(hclColors)`, with utility functions.
- `list.files()` (aka `dir()`) gains a new optional argument `no..` which allows to exclude `"."` and `".."` from listings.
- Multiple time series are also of class `"matrix"`; consequently, `head()`, e.g., is more useful.
- `encodeString()` preserves UTF-8 marked encodings. Thus if factor levels are marked as UTF-8 an attempt is made to print them in UTF-8 in `RGui` on Windows.

- `readLines()` and `scan()` (and hence `read.table()`) in a UTF-8 locale now discard a UTF-8 byte-order-mark (BOM). Such BOMs are allowed but not recommended by the Unicode Standard: however Microsoft applications can produce them and so they are sometimes found on websites.

  The encoding name `"UTF-8-BOM"` for a connection will ensure that a UTF-8 BOM is discarded.

- `mapply(FUN,a1,..)` now also works when `a1` (or a further such argument) needs a `length()` method (which the documented arguments never do). (Requested by Hervé Pagès; with a patch.)

- `.onDetach()` is supported as an alternative to `.Last.lib`. Unlike `.Last.lib`, this does not need to be exported from the package's namespace.

- The `srcfile` argument to `parse()` may now be a character string, to be used in error messages.

- The `format()` method for `ftable` objects gains a `method` argument, propagated to `write.ftable()` and `print()`, allowing more compact output, notably for LaTeX formatting, thanks to Marius Hofert.

- The `utils::process.events()` function has been added to trigger immediate event handling.

- `Sys.which()` now returns `NA` (not `""`) for `NA` inputs (related to PR#15147).

- The `print()` method for class `"htest"` gives fewer trailing spaces (wish of PR#15124).

  Also print output from `HoltWinters()`, `nls()` and others.

- `loadNamespace()` allows a version specification to be given, and this is used to check version specifications given in the '`Imports`' field when a namespace is loaded.

- `setClass()` has a new argument, `slots`, clearer and less ambiguous than `representation`. It is recommended for future code, but should be back-compatible. At the same time, the allowed slot specification is slightly more general. See the documentation for details.

- `mget()` now has a default for `envir` (the frame from which it is called), for consistency with `get()` and `assign()`.

- `close()` now returns an integer status where available, invisibly. (Wish of PR#15088.)

- The internal method of `tar()` can now store paths too long for the '`ustar`' format, using the (widely supported) GNU extension. It can also store long link names, but these are much less widely supported. There is support for larger files, up to the '`ustar`' limit of 8GB.

- Local reference classes have been added to package **methods**. These are a technique for avoiding unneeded copying of large components of objects while retaining standard R functional behavior. See `?LocalReferenceClasses`.

- `untar()` has a new argument `restore_times` which if false (not the default) discards the times in the tarball. This is useful if they are incorrect (some tarballs submitted to CRAN have times in a local time zone or many years in the past even though the standard required them to be in UTC).

- `replayplot()` cannot (and will not attempt to) replay plots recorded under R < 3.0.0. It may crash the R session if an attempt is made to replay plots created in a different build of R >= 3.0.0.

- Palette changes get recorded on the display list, so replaying plots (including when resizing screen devices and using `dev.copy()`) will work better when the palette is changed during a plot.
- `chol(pivot = TRUE)` now defaults to LAPACK, not LINPACK.
- The `parse()` function has a new parameter `keep.source`, which defaults to `options("keep.source")`.
- Profiling *via* `Rprof()` now optionally records information at the statement level, not just the function level.
- The `Rprof()` function now quotes function names in in its output file on Windows, to be consistent with the quoting in Unix.
- Profiling *via* `Rprof()` now optionally records information about time spent in GC.
- The HTML help page for a package now displays non-vignette documentation files in a more accessible format.
- To support `options(stringsAsFactors = FALSE)`, `model.frame()`, `model.matrix()` and `replications()` now automatically convert character vectors to factors without a warning.
- The `print` method for objects of class `"table"` now detects tables with 0-extents and prints the results as, e.g., '`<table of extent 0 x 1 x 2 >`'. (Wish of PR#15198.)
- Deparsing involving calls to anonymous functions has been made closer to reversible by the addition of extra parentheses.
- The function `utils::packageName()` has been added as a lightweight version of `methods::getPackageName()`.
- `find.package(lib.loc = NULL)` now treats loaded namespaces preferentially in the same way as attached packages have been for a long time.
- In Windows, the Change Directory dialog now defaults to the current working directory, rather than to the last directory chosen in that dialog.
- `available.packages()` gains a `"license/restricts_use"` filter which retains only packages for which installation can proceed solely based on packages which are guaranteed not to restrict use.
- New `check_packages_in_dir()` function in package **tools** for conveniently checking source packages along with their reverse dependencies.
- R's completion mechanism has been improved to handle help requests (starting with a question mark). In particular, help prefixes are now supported, as well as quoted help topics. To support this, completion inside quotes are now handled by R by default on all platforms.
- The memory manager now allows the strategy used to balance garbage collection and memory growth to be controlled by setting the environment variable `R_GC_MEM_GROW`. See `?Memory` for more details.
- ('For experts only', as the introductory manual says.) The use of environment variables `R_NSIZE` and `R_VSIZE` to control the initial (= minimum) garbage collection trigger for number of cons cels and size of heap has been restored: they can be overridden by the command-line options `--min-nsize` and `--min-vsize`; see `?Memory`.
- On Windows, the device name for bitmap devices as reported by `.Device` and `.Devices` no longer includes the file name. This is for consistency with other platforms and was requested by the **lattice** maintainer.

  `win.metafile()` still uses the file name: the exact form is used by package **tkrplot**.

- `set.seed(NULL)` re-initializes `.Random.seed` as done at the beginning of the session if not already set. (Suggestion of Bill Dunlap.)
- The `breaks` argument in `hist.default()` can now be a function that returns the breakpoints to be used (previously it could only return the suggested number of breakpoints).
- File 'share/licenses/licenses.db' has some clarifications, especially as to which variants of 'BSD' and 'MIT' is intended and how to apply them to packages. The problematic licence 'Artistic-1.0' has been removed.

LONG VECTORS

This section applies only to 64-bit platforms.

- There is support for vectors longer than $2^{31} - 1$ elements. This applies to raw, logical, integer, double, complex and character vectors, as well as lists. (Elements of character vectors remain limited to $2^{31} - 1$ bytes.)
- Most operations which can sensibly be done with long vectors work: others may return the error 'long vectors not supported yet'. Most of these are because they explicitly work with integer indices (e.g., `anyDuplicated()` and `match()`) or because other limits (e.g., of character strings or matrix dimensions) would be exceeded or the operations would be extremely slow.
- `length()` returns a double for long vectors, and lengths can be set to 2^{31} or more by the replacement function with a double value.
- Most aspects of indexing are available. Generally double-valued indices can be used to access elements beyond $2^{31} - 1$.
- There is some support for matrices and arrays with each dimension less than 2^{31} but total number of elements more than that. Only some aspects of matrix algebra work for such matrices, often taking a very long time. In other cases the underlying Fortran code has an unstated restriction (as was found for complex `svd()`).
- `dist()` can produce dissimilarity objects for more than 65536 rows (but for example `hclust()` cannot process such objects).
- `serialize()` to a raw vector is unlimited in size (except by resources).
- The C-level function `R_alloc` can now allocate 2^{35} or more bytes.
- `agrep()` and `grep()` will return double vectors of indices for long vector inputs.
- Many calls to `.C()` have been replaced by `.Call()` to allow long vectors to be supported (now or in the future). Regrettably several packages had copied the non-API `.C()` calls and so failed.
- `.C()` and `.Fortran()` do not accept long vector inputs. This is a precaution as it is very unlikely that existing code will have been written to handle long vectors (and the R wrappers often assume that `length(x)` is an integer).
- Most of the methods for `sort()` work for long vectors.

  `rank()`, `sort.list()` and `order()` support long vectors (slowly except for radix sorting).
- `sample()` can do uniform sampling from a long vector.

PERFORMANCE IMPROVEMENTS

- More use has been made of R objects representing registered entry points, which is more efficient as the address is provided by the loader once only when the package is loaded.

  This has been done for packages `base`, `methods`, `splines` and `tcltk`: it was already in place for the other standard packages.

  Since these entry points are always accessed by the R entry points they do not need to be in the load table which can be substantially smaller and hence searched faster. This does mean that `.C` / `.Fortran` / `.Call` calls copied from earlier versions of R may no longer work – but they were never part of the API.

- Many `.Call()` calls in package **base** have been migrated to `.Internal()` calls.

- `solve()` makes fewer copies, especially when `b` is a vector rather than a matrix.

- `eigen()` makes fewer copies if the input has dimnames.

- Most of the linear algebra functions make fewer copies when the input(s) are not double (e.g., integer or logical).

- A foreign function call (`.C()` etc) in a package without a `PACKAGE` argument will only look in the first DLL specified in the 'NAMESPACE' file of the package rather than searching all loaded DLLs. A few packages needed `PACKAGE` arguments added.

- The `@<-` operator is now implemented as a primitive, which should reduce some copying of objects when used. Note that the operator object must now be in package **base**: do not try to import it explicitly from package **methods**.

PACKAGE INSTALLATION

- The transitional support for installing packages without namespaces (required since R 2.14.0) has been removed. `R CMD build` will still add a namespace, but a `.First.lib()` function will need to be converted.

  `R CMD INSTALL` no longer adds a namespace (so installation will fail), and a `.First.lib()` function in a package will be ignored (with an installation warning for now).

  As an exception, packages without a 'R' directory and no 'NAMESPACE' file can still be installed.

- Packages can specify in their 'DESCRIPTION file' a line like

  ```
  Biarch: yes
  ```

  to be installed on Windows with '`--force-biarch`'.

- Package vignettes can now be processed by other engines besides `Sweave`; see 'Writing R Extensions' and the `tools::vignetteEngine` help topic for details.

- The '*.R' tangled source code for vignettes is now included in tarballs when `R CMD build` is used to produce them. In R 3.0.0, '*.R' files not in the sources will be produced at install time, but eventually this will be dropped.

- The package type "`mac.binary`" now looks in a path in the repository without any Mac subtype (which used to be '`universal`' or '`leopard`'): it looks in 'bin/macosx/contrib/3.0' rather than 'bin/macosx/leopard/contrib/2.15'). This is the type used for the CRAN binary distribution for OS X as from R 3.0.0.

- File 'etc/Makeconf' makes more use of the macros `$(CC)`, `$(CXX)`, `$(F77)` and `$(FC)`, so the compiler in use can be changed by setting just these (and if necessary the corresponding flags and `FLIBS`) in file '~/.R/Makevars'.

  This is convenient for those working with binary distributions of R, e.g. on OS X.

UTILITIES

- `R CMD check` now gives a warning rather than a note if it finds calls to `abort`, `assert` or `exit` in compiled code, and has been able to find the '.o' file in which the calls occur.

  Such calls can terminate the R process which loads the package.

- The location of the build and check environment files can now be specified by the environment variables `R_BUILD_ENVIRON` and `R_CHECK_ENVIRON`, respectively.

- `R CMD Sweave` gains a '`--compact`' option to control possibly reducing the size of the PDF file it creates when '`--pdf`' is given.

- `R CMD build` now omits Eclipse's '.metadata' directories, and `R CMD check` warns if it finds them.

- `R CMD check` now does some checks on functions defined within reference classes, including of `.Call()` etc calls.

- `R CMD check --as-cran` notes assignments to the global environment, calls to `data()` which load into the global environment, and calls to `attach()`.

- `R CMD build` by default uses the internal method of `tar()` to prepare the tarball. This is more likely to produce a tarball compatible with `R CMD INSTALL` and `R CMD check`: an external `tar` program, including options, can be specified *via* the environment variable `R_BUILD_TAR`.

- `tools::massageExamples()` is better protected against packages which re-define base functions such as `cat()` and `get()` and so can cause `R CMD check` to fail when checking examples.

- `R CMD javareconf` has been enhanced to be more similar to the code used by `configure`.

  There is now a test that a JNI program can be compiled (like `configure` did) and only working settings are used.

  It makes use of custom settings from configuration recorded in 'etc/javaconf'.

- The '`--no-vignettes`' argument of `R CMD build` has been renamed to the more accurate '`--no-build-vignettes`': its action has always been to (re)build vignettes and never omitted them.

  `R CMD check` accepts '`--no-build-vignettes`' as a preferred synonym for '`--no-rebuild-vignettes`'.

DEPRECATED AND DEFUNCT

- The `ENCODING` argument to `.C()` is defunct. Use `iconv()` instead.

- The `.Internal(eval.with.vis)` non-API function has been removed.

- Support for the converters for use with `.C()` has been removed, including the oft misused non-API header 'R_ext/RConverters.h'.

- The previously deprecated uses of `array()` with a 0-length `dim` argument and `tapply()` with a 0-length `INDEX` list are now errors.

- '`Translation`' packages are defunct.

- Calling `rep()` or `rep.int()` on a pairlist or other non-vector object is now an error.

- Several non-API entry points have been transferred to packages (e.g., `R_zeroin2`) or replaced by different non-API entry points (e.g., `R_tabulate`).

- The 'internal' graphics device invoked by `.Call("R_GD_nullDevice", package = "grDevices")` has been removed: use `pdf(file = NULL)` instead.

- The `.Fortran()` entry point `"dqrls"` which has not been used by R since version 2.15.1 is no longer available.
- Functions `traceOn()` and `traceOff()` in package **methods** are now defunct.
- Function `CRAN.packages()` is finally defunct.
- Use of `col2rgb(0)` is defunct: use `par("bg")` or `NA` instead.
- The long-defunct functions `Rd_parse()`, `anovalist.lm()`, `categpry()`, `clearNames()`, `gammaCody()`, `glm.fit.null()`, `lm.fit.null()`, `lm.wfit.null()`, `manglePackageNames()`, `mauchley.test()`, `package.contents()`, `print.coefmat()`, `reshapeLong()`, `reshapeWide()`, `tkclose()`, `tkcmd()`, `tkfile.dir()`, `tkfile.tail()`, `tkopen()`, `tkputs()`, `tkread()`, `trySilent()` and `zip.file.extract()` have been removed entirely (but are still documented in the help system).
- The unused `dataPath` argument to `attachNamespace()` has been removed.
- `grid.prompt()` has been removed: use `devAskNewPage()` instead.
- The long-deprecated `intensities` component is no longer returned by `hist()`.
- `mean()` for data frames and `sd()` for data frames and matrices are defunct.
- `chol(pivot = FALSE,LINPACK = TRUE)`, `ch2inv(LINPACK = TRUE)`, `eigen(EISPACK = TRUE)`, `solve(LINPACK = TRUE)` and `svd(LINPACK = TRUE)` are defunct: LAPACK will be used, with a warning.
- The `keep.source` argument to `library()` and `require()` is defunct. This option needs to be set at install time.
- Documentation for `real()`, `as.real()` and `is.real()` has been moved to 'defunct' and the functions removed.
- The `maxRasters` argument of `pdf()` (unused since R 2.14.0) has been removed.
- The unused `fontsmooth` argument has been removed from the `quartz()` device.
- All the (non-API) EISPACK entry points in R have been removed.
- `chol(pivot = TRUE,LINPACK = TRUE)` is deprecated.
- The long-deprecated use of `\synopsis` in the 'Usage' section of '.Rd' files will be removed in R 3.1.0.
- `.find.package()` and `.path.package()` are deprecated: only the public versions without the dot have ever been in the API.
- In a package's 'DESCRIPTION' file,

  ```
  License: X11
  ```

  is deprecated, since it includes 'Copyright (C) 1996 X Consortium' which cannot be appropriate for a current R package. Use 'MIT' or 'BSD_2_clause' instead.

CODE MIGRATION

- The C code underlying base graphics has been migrated to the **graphics** package (and hence no longer uses `.Internal()` calls).
- Most of the `.Internal()` calls used in the **stats** package have been migrated to C code in that package.

  This means that a number of `.Internal()` calls which have been used by packages no longer exist, including `.Internal(cor)` `.Internal(cov)`, `.Internal(optimhess)` and `.Internal(update.formula)`.

- Some `.External()` calls to the base package (really to the R executable or shared library) have been moved to more appropriate packages. Packages should not have been using such calls, but some did (mainly those used by `integrate()`).

PACKAGE parallel

- There is a new function `mcaffinity()` which allows getting or setting the CPU affinity mask for the current R process on systems that supports this (currently only Linux has been tested successfully). It has no effect on systems which do not support process affinity. Users are not expected to use this function directly (with the exception of fixing libraries that break affinity settings like OpenBLAS) – the function is rather intended to support affinity control in high-level parallel functions. In the future, R may supplement lack of affinity control in the OS by its own bookkeeping *via* `mcaffinity()` related to processes and threads it spawns.
- `mcparallel()` has a new argument `mc.affinity` which attempts to set the affinity of the child process according to the specification contained therein.
- The port used by socket clusters is chosen randomly: this should help to avoid clashes observed when two users of a multi-user machine try to create a cluster at the same time. To reproduce the previous behaviour set environment variable `R_PARALLEL_PORT` to `10187`.

C-LEVEL FACILITIES

- There has been some minor re-organization of the non-API header files. In particular, 'Rinternals.h' no longer includes the non-API header 'R_exts/PrtUtil.h', and that no longer includes 'R_exts/Print.h'.
- Passing `NULL` to `.C()` is now an error.
- `.C()` and `.Fortran()` now warn if `"single"` arguments are used with `DUP = FALSE`, as changes to such arguments are not returned to the caller.
- C entry points `R_qsort` and `R_qsort_I` now have `start` and `end` as `size_t` to allow them to work with longer vectors on 64-bit platforms. Code using them should be recompiled.
- A few recently added C entry points were missing the remapping to `Rf_`, notably `[dpq]nbinom_mu`.
- Some of the interface pointers formerly available only to `R.app` are now available to front-ends on all Unix-alikes: one has been added for the interface to `View()`.
- `PACKAGE = ""` is now an error in `.C()` etc calls: it was always contrary to the documentation.
- Entry point `rcont2` has been migrated to package **stats** and so is no longer available.
- `R_SVN_REVISION` in 'Rversion.h' is now an integer (rather than a string) and hence usable as e.g. `#if R_SVN_REVISION <70000`.
- The entry points `rgb2hsv` and `hsv2rgb` have been migrated to package **grDevices** and so are no longer available.
- `R_GE_version` has been increased to `10` and `name2col` removed (use `R_GE_str2col` instead). R internal colour codes are now defined using the typedef `rcolor`.
- The `REPROTECT` macro now checks that the protect index is valid.

- Several non-API entry points no longer used by R have been removed, including the Fortran entry points `chol`, `chol2inv`, `cg`, `ch` and `rg`, and the C entry points `Brent_fmin`, `fft_factor` and `fft_work`.
- If a `.External` call is registered with a number of arguments (other than `-1`), the number of arguments passed is checked for each call (as for other foreign function calls).
- It is now possible to write custom connection implementations outside core R using 'R_ext/Connections.h'. Please note that the implementation of connections is still considered internal and may change in the future (see the above file for details).

INTERNATIONALIZATION

- The management of translations has been converted to R code: see `?tools::update_pkg_po`.
- The translations for the R interpreter and `RGui.exe` are now part of the **base** package (rather than having sources in directory 'po' and being installed to 'share/locale'). Thus the **base** package supports three translation domains, `R-base`, `R` and `RGui`.
- The compiled translations which ship with R are all installed to the new package **translations** for easier updating. The first package of that name found on `.libPaths()` at the start of the R session will be used. (It is possible messages will be used before `.libPaths()` is set up in which case the default translations will be used: set environment variable `R_TRANSLATIONS` to point to the location of the intended **translations** package to use this right from the start.)
- The translations form a separate group in the Windows installer, so can be omitted if desired.
- The markup for many messages has been changed to make them easier to translate, incorporating suggestions from Łukasz Daniel.

INSTALLATION

- There is again support for building without using the C 'long double' type. This is required by C99, but system implementations can be slow or flawed. Use `configure` option '`--disable-long-double`'.
- `make pdf` and `make install-pdf` now make and install the full reference index (including all base and recommended packages).
- The 'reference manual' on the Windows GUI menu and included in the installer is now the full reference index, including all base and recommended packages.
- R help pages and manuals have no ISBNs because ISBN rules no longer allow constantly changing content to be assigned an ISBN.
- The Windows installer no longer installs a Start Menu link to the static help pages; as most pages are generated dynamically, this led to a lot of broken links.
- Any custom settings for Java configuration are recorded in file 'etc/javaconf' for subsequent use by `R CMD javareconf`.
- There is now support for `makeinfo` version 5.0 (which requires a slightly different '.texi' syntax).
- The minimum versions for '`--use-system-zlib`' and `--use-system-pcre` are now tested as 1.2.5 and 8.10 respectively.
- On Windows, the stack size is reduced to 16MB on 32-bit systems: misguided users were launching many threads without controlling the stack size.
- `configure` no longer looks for file '~/.Rconfig': '~/.R/config' has long been preferred.

BUG FIXES

- When `R CMD build` is run in an encoding other than the one specified in the package's 'DESCRIPTION' file it tries harder to expand the `authors@R` field in the specified encoding. (PR#14958)
- If `R CMD INSTALL` is required to expand the `authors@R` field of the 'DESCRIPTION' file, it tries harder to do so in the encoding specified for the package (rather than using ASCII escapes).
- Fix in package **grid** for pushing a viewport into a layout cell, where the layout is within a viewport that has zero physical width OR where the layout has zero total relative width (likewise for height). The layout column widths (or row heights) in this case were being calculated with non-finite values. (Reported by Winston Chang.)
- `solve(A,b)` for a vector `b` gave the answer names from `colnames(A)` for `LINPACK = TRUE` but not in the default case.
- `La.svd()` accepts logical matrices (as documented, and as `svd()` did).
- `legend()` now accepts negative `pch` values, in the same way `points()` long has.
- Parse errors when installing files now correctly display the name of the file containing the bad code.
- In Windows, tcltk windows were not always properly constructed. (PR#15150)
- The internal functions implementing `parse()`, `tools::parseLatex()` and `tools::parse_Rd()` were not reentrant, leading to errors in rare circumstances such as a garbage collection triggering a recursive call.
- Field assignments in reference class objects *via* `$<-` were not being checked because the magic incantation to turn methods on for that primitive operator had been inadvertently omitted.
- `setHook(hookname,value,action="replace")` set the hook to be the value, rather than a list containing the value as documented. (PR#15167)
- If a package used a 'NEWS.Rd' file, the main HTML package index page did not link to it. (Reported by Dirk Eddelbuettel.)
- The primitive implementation of `@<-` was not checking the class of the replacement. It now does a check, quicker but less general than `slot<-`. See the help.
- `split(x,f)` now recycles classed objects `x` in the same way as vectors. (Reported by Martin Morgan.)
- `pbeta(.28,1/2,2200,lower.tail=FALSE,log.p=TRUE)` is no longer `-Inf`; ditto for corresponding `pt()` and `pf()` calls, such as `pt(45,df=5000,lower.tail=FALSE,log.p=TRUE)`. (PR#15162)
- The Windows graphics device would crash R if a user attempted to load the graphics history from a variable that was not a saved history. (PR#15230)
- The workspace size for the `predict()` method for `loess()` could exceed the maximum integer size. (Reported by Hiroyuki Kawakatsu.)
- `ftable(x,row.vars,col.vars)` now also works when the `*.vars` arguments are (integer or character vectors) of length zero.
- Calling `cat()` on a malformed UTF-8 string could cause the Windows GUI to lock up. (PR#15227)
- `removeClass(cc)` gave "node stack overflow" for some class definitions containing `"array"` or `"matrix"`.

CHANGES in previous versions

- Older news can be found in text format in files 'NEWS.0', 'NEWS.1' and 'NEWS.2' in the 'doc' directory. News in HTML format for R versions from 2.10.0 to 2.15.3 is in 'doc/html/NEWS.2.html'.