

The Journal

Volume 9/1, June 2017

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

Contents

Editorial	4
Contributed Research Articles	
iotools : High-Performance I/O Tools for R	6
IsoGeneGUI : Multiple Approaches for Dose-Response Analysis of Microarray Data Using R	14
Network Visualization with ggplot2	27
OrthoPanels : An R Package for Estimating a Dynamic Panel Model with Fixed Effects Using the Orthogonal Reparameterization Approach	60
The mosaic Package: Helping Students to ‘Think with Data’ Using R	77
smoof : Single- and Multi-Objective Optimization Test Functions	103
minval : An R package for MINimal VALidation of Stoichiometric Reactions	114
Working with Daily Climate Model Output Data in R and the futureheatwaves Pack- age	124
alineR : an R Package for Optimizing Feature-Weighted Alignments and Linguistic Distances	138
Implementing a Metapopulation Bass Diffusion Model using the R Package deSolve	153
MDplot : Visualise Molecular Dynamics	164
On Some Extensions to GA Package: Hybrid Optimisation, Parallelisation and Islands Evolution	187
imputeTS : Time Series Missing Value Imputation in R	207
The NoiseFiltersR Package: Label Noise Preprocessing in R	219
coxphMIC : An R Package for Sparse Estimation of Cox Proportional Hazards Models via Approximated Information Criteria	229
Update of the nlme Package to Allow a Fixed Standard Deviation of the Residual Error	239
EMSaov : An R Package for the Analysis of Variance with the Expected Mean Squares and its Shiny Application	252
GsymPoint : An R Package to Estimate the Generalized Symmetry Point, an Optimal Cut-off Point for Binary Classification in Continuous Diagnostic Tests	262
autoimage : Multiple Heat Maps for Projected Coordinates	284
Market Area Analysis for Retail and Service Locations with MCI	298

PSF : Introduction to R Package for Pattern Sequence Based Forecasting Algorithm . . .	324
flan : An R Package for Inference on Mutation Models.	334
Multilabel Classification with R Package mlr	352
Counterfactual : An R Package for Counterfactual Analysis	370
Retrieval and Analysis of Eurostat Open Data with the eurostat Package	385
PGEE : An R Package for Analysis of Longitudinal Data with High-Dimensional Covariates.	393
BayesBinMix : an R Package for Model Based Clustering of Multivariate Binary Data .	403
pdp : An R Package for Constructing Partial Dependence Plots	421
checkmate : Fast Argument Checks for Defensive R Programming	437
milr : Multiple-Instance Logistic Regression with Lasso Penalty	446
spcadjust : An R Package for Adjusting for Estimation Error in Control Charts	458
Weighted Effect Coding for Observational Data with wec	477
Hosting Data Packages via drat : A Case Study with Hurricane Exposure Data	486
 News and Notes	
R Foundation News	498
Conference Report: European R Users Meeting 2016	501
Changes on CRAN	505
News from the Bioconductor Project	508
Changes in R	509

The R Journal is a peer-reviewed publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors. All articles are licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0, <http://creativecommons.org/licenses/by/4.0/>).

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

Editor-in-Chief:

Roger Bivand, Norwegian School of Economics, Bergen,
Norway

Executive editors:

Michael Lawrence, Genentech, Inc., San Francisco, USA
John Verzani, City University of New York, USA
Norman Matloff, University of California, Davis, USA

Email:

r-journal@R-project.org

R Journal Homepage:

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

Editorial advisory board:

Bettina Grün, Johannes Kepler Universität Linz, Austria
Deepayan Sarkar, Indian Statistical Institute, Delhi, India
Friedrich Leisch, University of Natural Resources and Life
Sciences, Vienna, Austria

Hadley Wickham, RStudio, Houston, Texas, USA
Heather Turner, University of Warwick, Coventry, UK
John Fox, McMaster University, Hamilton, Ontario, Canada
Kurt Hornik, WU Wirtschaftsuniversität Wien, Vienna,
Austria

Paul Murrell, University of Auckland, New Zealand
Peter Dalgaard, Copenhagen Business School, Denmark
Martyn Plummer, International Agency for Research on
Cancer, Lyon, France

Vincent Carey, Harvard Medical School, Boston, USA
Torsten Hothorn, University of Zurich, Switzerland

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

Editorial

by Roger Iovand

This new issue, Volume 9, Issue 1, of the R Journal contains 33 contributed research articles, like the second issue of 2016. Most of the articles present R packages, and cover a very wide range of uses of R. Our journal continues to be critically dependent on its readers, authors, reviewers and editors. Annual submission numbers have grown markedly, but the rate of growth is less than that of the number of CRAN packages. Table 1 shows the outcomes of submitted contributed articles by year of submission. The proportion of submissions reaching publication has been roughly half since 2012.

	2009	2010	2011	2012	2013	2014	2015	2016
Published	26	26	26	22	31	36	51	58
Rejected	11	14	11	24	29	32	53	64
Under review	0	0	0	0	0	0	0	19
Total	37	40	37	46	60	68	104	141

Table 1: Submission outcomes 2009–2016, by year of submission.

In order to try to restore some balance to the inflow of submissions, the kinds of articles solicited were clarified in January 2017. Articles introducing CRAN or Bioconductor packages — the most common kind of submission — should now provide broader context. We would like to encourage the submission of reviews and proposals, comparisons and the benchmarking of alternative implementations, and presentations of applications demonstrating how new or existing techniques can be applied in an area of current interest using R.

	2009	2010	2011	2012	2013	2014	2015	2016
Page count	109	123	123	136	362	358	479	895
Article count	18	18	20	18	35	33	36	62
Average length	6.1	6.8	6.2	7.6	10.3	10.8	13.3	14.4

Table 2: Published contributed articles 2009–2016, by year of publication.

Not only has the number of submissions increased, but the length of published articles has also increased (see Table 2). The apparent jump from 2012 to 2013 may be associated with the change from a two column to a single column format, but page counts have risen, increasing the workload of reviewers and editors. We only have consistent records of the time taken to process accepted contributed articles for the 2013–2016 period. Again, the excellent work done by our generous reviewers and my very hard-working predecessors and especially Michael Lawrence last year, is evident in holding median times from receipt to publication online to a little over 200 days, as Table 3 shows.

	2013	2014	2015	2016
Median	347.0	225.5	212.5	212.0

Table 3: Median day count from acknowledgement to acceptance and online publication 2013–2016, by year of publication.

Using `gender` (Blevins and Mullen, 2015; Mullen, 2016) and `genderizeR` (Wais, 2016a,b), it is also possible to use author given names¹ to try to monitor author diversity; affiliation

¹The articles describing the packages used here stress the uncertainty involved in binary assignment.

location has not yet been successfully examined. Table 4 shows that there remains plenty to do to reflect the strengths of our community adequately².

	2009	2010	2011	2012	2013	2014	2015	2016
Women	5	9	8	6	10	18	27	32
Men	32	30	33	27	62	55	55	121
Unknown	3	5	3	3	7	4	9	10

Table 4: Authors of published articles 2009–2016, by year of publication; women/men split based on author given names.

In addition to re-framing the description of the kinds of articles we invite authors to contribute to our journal, work has been done on our website. Its appearance has been brought into line with that of the main R project website, and articles are reached through “landing” pages containing the abstract and citation information as well as listings of CRAN and Bioconductor packages cited in the article. So far very few contributed articles associate themselves directly with CRAN Task Views, so these are inferred from cited CRAN packages and listed on the landing pages. Further progress in helping to make work published in our journal more accessible is planned.

I hope you continue to enjoy and benefit from reading work published in our journal.

Bibliography

- C. Blevins and L. Mullen. Jane, John . . . Leslie? a historical method for algorithmic gender prediction. *Digital Humanities Quarterly*, 9, 2015. URL <http://www.digitalhumanities.org/dhq/vol/9/3/000223/000223.html>. [p4]
- L. Mullen. *gender: Predict Gender from Names Using Historical Data*, 2016. URL <https://github.com/ropensci/gender>. R package version 0.5.1. [p4]
- K. Wais. Gender Prediction Methods Based on First Names with *genderizeR*. *The R Journal*, 8(1):17–37, 2016a. URL <https://journal.r-project.org/archive/2016/RJ-2016-002/index.html>. [p4]
- K. Wais. *genderizeR: Gender Prediction Based on First Names*, 2016b. URL <https://CRAN.R-project.org/package=genderizeR>. R package version 2.0.0. [p4]

Roger Bivand

Roger.Bivand@r-project.org

²Although relative binary proportions do not differ greatly from those shown by a recent survey of useR participants (<https://forwards.github.io/blog/2017/01/13/mapping-users/>), the Norwegian context of the editor suggests that complacency or change of focus are unhelpful.

iotools: High-Performance I/O Tools for R

by Taylor Arnold, Michael J. Kane, and Simon Urbanek

Abstract The **iotools** package provides a set of tools for input and output intensive data processing in R. The functions `chunk.apply` and `read.chunk` are supplied to allow for iteratively loading contiguous blocks of data into memory as raw vectors. These raw vectors can then be efficiently converted into matrices and data frames with the **iotools** functions `mstrsplit` and `dstrsplit`. These functions minimize copying of data and avoid the use of intermediate strings in order to drastically improve performance. Finally, we also provide `read.csv.raw` to allow users to read an entire dataset into memory with the same efficient parsing code. In this paper, we present these functions through a set of examples with an emphasis on the flexibility provided by chunk-wise operations. We provide benchmarks comparing the speed of `read.csv.raw` to data loading functions provided in base R and other contributed packages.

Introduction

When processing large datasets, specifically those too large to fit into memory, the performance bottleneck is often getting data from the hard-drive into the format required by the programming environment. The associated latency comes from a combination of two sources. First, there is hardware latency from moving data from the hard-drive to RAM. This is especially the case with “spinning” disk drives, which can have throughput speeds several orders of magnitude less than those of RAM. Hardware approaches for addressing latency have been an active area of research and development since hard-drives have existed. Solid state drives and redundant arrays of inexpensive disks (RAID) now provide throughput comparable to RAM. They are readily available on commodity systems and they continue to improve. The second source comes from the software latency associated with transforming data from its representation on the disk to the format required by the programming environment. This translation slows down performance for many R users, especially in the context of larger data sources.

We can compare the time needed to read, parse, and create a data frame with the time needed to just read data from disk. In order to do this, we will make use of the “Airline on-time performance” dataset, which was compiled for the 2009 American Statistical Association (ASA) Section on Statistical Computing and Statistical Graphics biannual data exposition from data released by the United States Department of Transportation (RITA, 2009). The dataset includes commercial flight arrival and departure information from October 1987 to April 2008 for those carriers with at least 1% of domestic U.S. flights in a given year. In total, there is information for over 120 million flights, with 29 variables related to flight time, delay time, departure airport, arrival airport, and so on. The uncompressed dataset is 12 gigabytes (GB) in size. While 12 GBs may not seem impressively “big” to many readers, it is still large enough to investigate the performance properties and, unlike most other large datasets, it is freely available. Supplemental materials, accessible at <https://github.com/statsmaths/iotools-supplement>, provide code for downloading the dataset and running the benchmarks included in this paper.

In a first step before measuring the time to load the dataset, column classes are defined so that this part of the data frame processing does not become part of our timings. These can be inferred by reviewing the dataset documentation and inspecting the first few rows of data. The column data types for this dataset are given by:

```
> col_types <- c(rep("integer", 8), "character", "integer", "character",  
+               rep("integer", 5), "character", "character",  
+               rep("integer", 4), "character", rep("integer", 6))
```

Now, we will read in the file `2008.csv`, which contains comma-separated values with 29 columns and 7,009,728 rows.

```
> system.time(read.csv("2008.csv", colClasses = col_types))["elapsed"]  
[1] 68.257
```

It takes just over 68 seconds to read the file into R and parse its contents into a data frame object. We can test how much of this time is due to just loading the results from the hard-drive into R’s memory. This is done using the `readBin` function to read the file as a raw string of bytes and `file.info` to infer the total size of the file; both functions are provided by the **base** package of R.

```
> system.time(readBin("2008.csv", "raw",  
+                   file.info("2008.csv")[["size"]]))["elapsed"]  
[1] 0.493
```

This takes less than one half of a second. It takes about 138 times longer to read and parse the data with `read.csv` than to just read the data in as raw bytes, indicating there may be room for improvement. This is not to say `read.csv` and its associated functions are poorly written. On the contrary, they are robust and do an excellent job inferring format and shape characteristics from data. They allow users to import and examine a dataset without knowing how many rows it has, how many columns it has, or its column types. Because of these functions, statisticians using R are able to focus on data exploration and modeling instead of file formats and schemas.

While existing functions are sufficient for processing relatively small datasets, larger ones require a different approach. For large files, data are often processed on a single machine by first being broken into a set of consecutive rows or “chunks.” Each chunk is loaded and processed individually, before retrieving the next chunk. The results from processing each chunk are then aggregated and returned. Small, manageable subsets are streamed from the disk to the processor with enough memory to represent a single chunk required by the system. This approach is common not only in single machine use but also in distributed environments with technologies such as Spark (Zaharia et al., 2010) and Hadoop MapReduce (Dean and Ghemawat, 2008). Clusters of commodity machines, such as those provided by Amazon Elastic Compute Cloud (EC2) and Digital Ocean, are able to process vast amounts of data one chunk at a time. Many statistical methods are compatible with this computational approach and are justified in a variety of contexts, including Hartigan (1975), Kleiner et al. (2014), Guha et al. (2012), and Matloff (2016).

However, base R together with the contributed packages currently does not provide convenient functionality to implement this common computing pattern. Packages such as `bigmemory` (Kane et al., 2013) and `ff` (Adler et al., 2014) provide data structures using their own binary formats over memory-mapped files stored on disk. The data structures they provide are not native R objects. They therefore do not exhibit properties such as copy-on-write behavior, which avoids making unnecessary copies of the dataset in memory (Rodeh, 2008), and, in general, they cannot be seamlessly integrated with R’s plethora of user contributed packages. The `readr` package (Wickham et al., 2016) provides fast importing of “`data.frame`” objects but it does not support chunk-wise operations for arbitrarily large files. The `foreach` package (Revolution Analytics and Weston, 2015a), and its associated `iterators` package (Revolution Analytics and Weston, 2015b), provide a general framework for chunked processing but does not provide the low-level connection-based utilities for transforming binary data stored on the disk to those native to R.

The `iotools` package provides tools for data processing using any data source represented as a connection (Arnold and Urbanek, 2015). Users of the package can import text data into R and process large datasets iteratively over small chunks. The package’s functions can be several orders of magnitude faster than R’s native facilities. The package provides general tools for quickly processing large datasets in consecutive chunks and provides a basis for speeding up distributed computing frameworks including Hadoop Streaming (The Apache Software Foundation, 2013) and Spark.

The remainder of this paper introduces the use of the `iotools` package for quickly importing datasets and processing them in R. Examples center around the calculation of ordinary least squares (OLS) slope coefficients via the normal equations in a linear regression. This particular application was chosen because it balances read and write times with processing time.

Input methods and formatters

R’s file operations make use of Standard C input and output operations including `fread` and `fwrite`. Data are read in, elements are parsed, and parsed values populate data structures. The `iotools` package also uses the Standard C library but it makes use of “bulk” binary operations including `memchr` and `strchr`. These functions make use of hardware specific, single instruction, multiple data operations (SIMD) and tend to be faster than their Standard I/O counterparts. (See, for example, Zhou and Ross (2002) for a complete overview of the benefits and common implementations of SIMD instructions.) As a result, `iotools` is able to find and retrieve data at a higher rate. In addition, an entire dataset or chunk is buffered rather than scanned and transformed line-by-line as in the `read.table` function. Thus, by buffering chunks of data and making use of low-level, system functions `iotools` is able to provide faster data ingestion than what is available in base R.

Importing data with `dstrsplit` and `read.csv.raw`

A core function in the `iotools` package is `dstrsplit`. It takes either a raw or character vector and splits it into a data frame according to a specified separator. Each column may be parsed into a logical, integer, numeric, character, raw, complex or POSIXct vector. Columns of type factor are not supported as a method of input, though columns may be converted to a factor once the dataset is

platform	method	integer	logical	num	char	num & char	complex	raw
Ubuntu 16.04	readRDS	0.1	0.1	0.2	7.0	7.2	2.0	0.1
Ubuntu 16.04	dstrsplit	0.8	1.0	2.7	2.6	5.2	5.1	0.6
Ubuntu 16.04	read_csv	1.5	1.7	5.9	2.5	9.7	.	.
Ubuntu 16.04	read.csv	11.0	15.0	50.2	9.0	59.4	96.2	8.4
macOS Sierra	readRDS	0.2	0.2	0.3	5.4	6.5	1.9	0.1
macOS Sierra	dstrsplit	0.9	1.0	2.8	2.4	5.2	5.3	0.6
macOS Sierra	read_csv	1.4	1.5	6.2	2.0	8.0	.	.
macOS Sierra	read.csv	8.6	11.1	39.3	6.7	46.6	70.1	6.2
Windows 7	readRDS	0.1	0.1	0.3	5.6	6.1	2.3	0.1
Windows 7	dstrsplit	1.5	1.3	4.4	2.7	5.6	8.8	0.7
Windows 7	read_csv	1.3	1.9	8.9	1.7	7.6	.	.
Windows 7	read.csv	6.3	7.5	25.7	4.7	29.7	48.1	3.8

Table 1: Time in seconds (average over 10 replications) to import a data frame by element type. Each data frame has 1 million rows and 25 columns of the specified data, except for the “num & char” column which has 25 columns of character values interleaved with 25 columns of numeric columns. Note that `read_csv`, from the `readr` package, does not support complex and raw types. Linux benchmarks used a server with a 3.7 GHz Intel Xeon E5 and 32 GB of memory. Mac benchmarks used a mid-2015 MacBook Pro with 2.5 GHz Intel Core i7 and 16 GB of memory. Windows benchmarks used a desktop machine with a 3.2 GHz Intel Core i5 CPU and 8 GB of memory.

loaded. It will be shown later that `dstrsplit` can be used in a streaming context and in this case data are read sequentially. As a result, the set of factor levels cannot be deduced until the entire sequence is read. However, in most cases, a caller knows the schema and is willing to specify factor levels after loading the data or is willing to use a single pass to find all of the factor levels.

The tools in `iotools` were primarily developed to support the chunk-wise processing of large datasets that are too large to be read entirely into memory. As an additional benefit it was observed that these functions are also significantly faster when compared to the `read.table` family of functions when importing a large plain-text character separated dataset into R. The `readAsRaw` function takes either a connection or a file name and returns the contents as a raw type. Combining this with `dstrsplit`, we can load the `2008.csv` file significantly faster:

```
> system.time(dstrsplit(readAsRaw("2008.csv"), sep = ",",
+ col_types = col_types))["elapsed"]
[1] 14.087
```

This takes about 14 seconds, which is roughly a five-fold decrease when compared to the `read.csv` function. In order to simplify its usage, the function `read.csv.raw` was written as a wrapper around `dstrsplit` for users who want to use `iotools` to import data in a manner similar to `read.table`:

```
> system.time(read.csv.raw("2008.csv", colClasses = col_types))["elapsed"]
[1] 14.251
```

The performance is very similar to the `dstrsplit` example.

Table 1 shows the time needed to import a data file with 1,000,000 rows and 25 columns using `readRDS`, `dstrsplit`, `read_csv` (from the `readr` package), and `read.table`. Imports were performed for each of R’s native types to see how their different size requirements affect performance. The return value of `read_csv` includes additional metadata because a ‘tbl_df’ object is returned, which is a subtype of R’s native “data.frame” class. Otherwise all functions return the exact same data. The benchmarks show that, except for character vectors, `readRDS` is fastest. This is unsurprising since `readRDS` stores the binary representation of an R object and importing consists of copying the file to memory and registering the object in R. `read_csv`’s performance is reasonably close to those of `iotools` across all three platforms and data types. The `iotools` functions are generally faster on integer and numeric types, whereas the `readr` functions are slightly faster on character types. In both cases, no performance is lost when dealing with data sets that have mixed data types and results are consistent across operating systems.

Processing the model matrix

In this and the following section, we will show how to estimate, based on the entire Airline on-time performance dataset, the slope coefficients for the following linear regression model using OLS:

$$\text{ArrDelay} \sim \text{DayOfWeek} + \text{DepTime} + \text{Month} + \text{DepDelay}. \quad (1)$$

The OLS slope estimates can be calculated by creating the model matrix and applying the normal equations to derive the coefficients. Given the size of the dataset in this example, it will not be possible to calculate the normal equation matrices directly in memory. Instead, the model matrix will be created sequentially over blocks of rows. As this dataset is further split with each year of data being stored in a separate file, we will also need an outer loop over the available yearly files (1988 to 2008).

We first construct the model matrix and save it in a file on disk; the slope coefficients will be calculated in a second step. Separate processing and model fitting in this case are mostly for the sake of breaking down the example into digestible bits. In many real-world data challenges it may still be a good idea, as it provides an intermediate way of checking whether problems arise while fitting the model. Regardless if problems occur either due to a bug in the code or an interruption in computing services, the model matrix does not need to be recalculated.

In order to construct a large model matrix file, we cycle over the individual data files and work on each separately. Each file is loaded using the `read.csv.raw` function, the variables `DayOfWeek` and `Month` are converted into factors. The departure time variable is given in a 24-hour format, in local time, and with the colon removed from a standard representation of time. For example, 4:30pm is given as "1630" and 5:23am is "523". Our code extracts the hour and minute and converts the time into minutes since midnight; less than 0.5% of the flights depart between midnight and 3:59am, so ignoring the circular nature of time is reasonable in this simple application. Finally, the entire output is stored as a comma separated file with the first column representing the response.

```
> out_file <- file("airline_mm.csv", "wb")
> for (data_file in sprintf("%04d.csv", 1988:2008)) {
+   df <- read.csv.raw(data_file, col_types = col_types)
+   df$DayOfWeek <- factor(df$DayOfWeek, levels = 1:7)
+   df$Month <- factor(df$Month, levels = 1:12)
+   df$DepTime <- sprintf("%04d", df$DepTime)
+   df$DepTime <- as.numeric(substr(df$DepTime, 1, 2)) * 60 +
+                 as.numeric(substr(df$DepTime, 3, 4))
+
+   mf <- model.frame(ArrDelay ~ DayOfWeek + DepTime + DepDelay + Month, df)
+   mm <- cbind(model.response(mf), model.matrix(mf, df))
+   rownames(mm) <- NULL
+   writeBin(as.output(mm, sep = ","), out_file)
+ }
> mm_names <- colnames(mm)
> close(out_file)
```

The output connection is recycled in each iteration of the loop thereby appending each year's data; the names of the model matrix are stored in memory for the next step. In the end we have one large file that contains the entire model matrix. The output is representative of the kinds of large datasets often encountered in industry applications.

Fitting the model with `mstrsplit` and `chunk.apply`

With the model matrices created, the next step is to estimate the slope coefficients β in the model

$$Y = X\beta + \varepsilon, \quad (2)$$

where $Y, \varepsilon \in \mathbb{R}^n$, and $\beta \in \mathbb{R}^d$, $n \geq d$; each element of ε is an i.i.d. random variable with mean zero; and X is a matrix in $\mathbb{R}^{n \times d}$ with full column rank. The analytic solution for estimating the OLS slope coefficients, β , is

$$\hat{\beta} = (X^T X)^{-1} X^T Y. \quad (3)$$

platform	method	integer	logical	num	char	complex	raw
Ubuntu 16.04	readRDS	0.2	0.2	0.2	0.2	0.2	0.2
Ubuntu 16.04	mstrsplit	0.5	0.7	2.4	1.2	4.6	0.4
Ubuntu 16.04	as.matrix	11.2	15.2	50.6	16.0	98.3	12.0
macOS Sierra	readRDS	0.2	0.2	0.2	0.2	0.2	0.2
macOS Sierra	mstrsplit	0.6	0.8	2.7	1.3	4.8	0.4
macOS Sierra	as.matrix	8.8	11.3	39.7	12.4	72.1	9.3
Windows 7	readRDS	0.1	0.2	0.1	0.1	0.3	0.1
Windows 7	mstrsplit	0.7	0.9	3.0	0.8	5.7	0.5
Windows 7	as.matrix	6.5	7.7	26.0	10.7	50.5	8.8

Table 2: Time in seconds to import a matrix by element type. Each matrix has 1 million rows and 25 columns. Linux benchmarks used a server with a 3.7 GHz Intel Xeon E5 and 32 GB of memory. Mac benchmarks used a mid-2015 MacBook Pro with 2.5 GHz Intel Core i7 and 16 GB of memory. Windows benchmarks used a desktop machine with a 3.2 GHz Intel Core i5 CPU and 8 GB of memory.

Consider the row-wise partitioning (or chunking) of Equation 2:

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_r \end{pmatrix} = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_r \end{pmatrix} \beta + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_r \end{pmatrix},$$

where Y_1, Y_2, \dots, Y_r ; X_1, X_2, \dots, X_r ; and $\varepsilon_1, \varepsilon_2, \dots, \varepsilon_r$ are data partitions where each chunk is composed of subsets of rows of the model matrix. Equation 3 may then be expressed as (Friedman et al., 2001),

$$\hat{\beta} = \left(\sum_{i=1}^r X_i^T X_i \right)^{-1} \sum_{i=1}^r X_i^T Y_i. \quad (4)$$

The matrices $X_i^T X_i$ and $X_i^T Y_i$ can be calculated on each chunk and then summed to calculate the slope coefficients.

In the previous step, the data were read into a data frame, but we now need to read data into a numeric matrix. Interestingly enough, this functionality is not provided in base R or the **Matrix** (Bates and Maechler, 2017) package. Users who wanted to read data from a file into a matrix must read it in as a data frame and then convert it using the `as.matrix` function. The **iotools** package fills this gap by providing the function `mstrsplit`, a matrix import function similar to function `dstrsplit`. Table 2 compares the performance of `mstrsplit` with `read.table` followed by a call to `as.matrix` along binary importing using `load`. As with `dstrsplit`, `mstrsplit` outperforms the base R's `read.table` benchmarks by an order of magnitude.

In order to fit the model we will need to read from `airline_mm.csv` in chunks. The function `chunk.apply`, provided in **iotools**, allows us to do this easily over an open connection. By default the data is read in as 32MB chunks, though this can be changed by the `CH.MAX.SIZE` parameter to `chunk.apply`. The parameter `CH.MERGE` describes how the outputs of all the chunks are combined. Common options include `list` or, when the result is a single vector, `c`. Here, we use `chunk.apply` to calculate the matrices $X^T y$ and $X^T X$ over chunks of the data:

```
> ne_chunks <- chunk.apply("airline_mm.csv",
+   function(x) {
+     mm <- mstrsplit(x, sep = ",", type= "numeric")
+     colnames(mm) <- mm_names
+     list(xtx = crossprod(mm[, -1]),
+         xty = crossprod(mm[, -1], mm[, 1, drop = FALSE]))
+   }, CH.MERGE = list)
```

Notice that we do not need to manually specify the chunks; this detail is abstracted away by the `chunk.apply` function, which simply selects contiguous sets of rows for our function to evaluate. The output of the chunk-wise operation can be combined using the Reduce function:

```
> xtx <- Reduce("+", Map(function(x) x$xtx, ne_chunks))
> xty <- Reduce("+", Map(function(x) x$xty, ne_chunks))
```

With these results, the regression function can be solved with the normal equations where d is small.

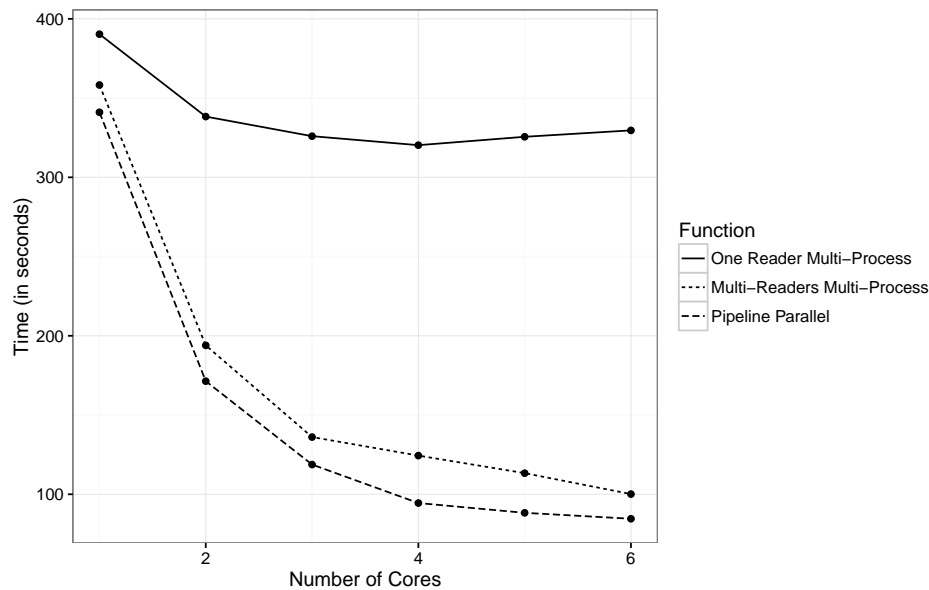


Figure 1: Average time over ten iterations to fit a linear model over the Airline on-time performance dataset as a function of the number of cores used and the number of parallel tasks used.

```
> qr.solve(xtx, xty)
      [,1]
(Intercept) 0.5564085990
DayOfWeek2  0.5720431343
DayOfWeek3  0.8480978666
DayOfWeek4  1.2436976583
DayOfWeek5  1.0805744488
DayOfWeek6 -1.2235684080
DayOfWeek7 -0.9883340887
DepTime     0.0003022008
DepDelay    0.9329374752
Month2      0.2880436452
Month3     -0.2198123852
...
```

The technique used by the `lm` function is similar to our approach, except that the QR-decomposition there is done directly on X itself rather than on $X^T X$. The difference is rarely an issue unless the problem is particularly ill-conditioned. In these cases, a small ridge regression penalty can be added to stabilize the solution (Friedman et al., 2001).

Parallel processing of chunks

In the example above the `xtx` and `xty` chunks are calculated serially. The `chunk.apply` function includes a parameter, `parallel`, allowing the user to specify the number of parallel processes, taking advantage of the embarrassingly parallel nature of these calculations. However, it is worth noting that parallelism in the `chunk.apply` function is slightly different from other functions such as `mclapply`.

Most parallel functions in R work by having worker processes receive data and an expression to compute. The master process initiates the computations and waits for them to complete. For I/O-intensive computations this in general means that either the master loads data before initiating the computation or the worker processes load the data. The former case is supported in `iotools` through iterator functions (`idstrsplit` and `imstrsplit`), which are compatible with the `foreach` package. However, in this case, new tasks cannot be started until data has been loaded for each of the workers. Loading the data on the master process may become a bottleneck and it may require much more time to load the data than to process it. The latter approach is also supported in `iotools` and ensures the master process is not a bottleneck. If, however, multiple worker processes on a single machine load a large amount of data from the same disk, resource contention at the system level may also cause excessive delays. The operating system has to service multiple requests for data from the same disk having limited I/O capability.

A third option, implemented in `chunk.apply`, provides *pipeline parallelism* where the master process sequentially loads data and then calls `mcparrallel` to initiate the parallel computation. When the maximum number of worker processes has been reached the master process *pre-fetches* the next chunk and then blocks on the result of the running worker processes. When the result is returned a newly created worker begins processing the pre-fetched data. In this way the master does not wait idly for worker processing and there is no resource contention since only the master is retrieving data. Pipeline parallelism increases execution throughput when the computation time is around the same order as the load time. When the load time exceeds the execution time, the overhead involved in initiating worker processes and getting their results will yield less desirable performance gains from parallelization. In the case of particularly long load times, the overhead will overwhelm the process and the parallel execution may be slower than a single serial calculation.

Figure 1 shows the times required to calculate $X^T X$ and $X^T Y$ for the normal equations in the regression described above using the three approaches described: all workers read, only the master reads, and pipeline parallelism. Pipeline parallelism performs the best overall, with all workers reading following close behind. However, all workers reading will only be able to keep pace with pipeline parallelism as long as there is sufficient hard-drive bandwidth and little contention from multiple reads. As a result, the pipeline parallel approach is likely a more general and therefore preferred strategy.

Conclusion

This paper presents the **iotools** package for the processing of data much larger than memory. Tools are included to efficiently load medium-sized files into memory and to parse raw vectors into matrices and data frames. The chunk-wise functionality is used as a building block to enable the processing of terabyte- and even petabyte-scale data. The examples emphasize computing on a single machine, however **iotools** is by no means limited to this configuration. The “chunk” functions are compatible with any object derived from a connection and can therefore be used with compressed files or even pipes and sockets. Our current work, in fact, uses **iotools** as a building block for more tightly integrating R into the Hadoop Streaming and Spark frameworks.

Bibliography

- D. Adler, C. Gläser, O. Nenadic, J. Oehlschlägel, and W. Zucchini. *ff: Memory-Efficient Storage of Large Data on Disk and Fast Access Functions*, 2014. URL <https://CRAN.R-project.org/package=ff>. R package version 2.2-13. [p7]
- T. Arnold and S. Urbanek. *iotools: I/O Tools for Streaming*, 2015. URL <https://CRAN.R-project.org/package=iotools>. R package version 0.1-12. [p7]
- 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. [p10]
- J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. URL <https://doi.org/10.1145/1327452.1327492>. [p7]
- J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001. URL <https://doi.org/10.1007/978-0-387-21606-5>. [p10, 11]
- S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W. S. Cleveland. Large complex data: divide and recombine (D&R) with RHIFE. *Stat*, 1(1):53–67, 2012. URL <https://doi.org/10.1002/sta4.7>. [p7]
- J. A. Hartigan. Necessary and sufficient conditions for asymptotic joint normality of a statistic and its subsample values. *The Annals of Statistics*, 3(3):573–580, 1975. [p7]
- M. J. Kane, J. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. URL <https://doi.org/10.18637/jss.v055.i14>. [p7]
- A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society B*, 76(4):795–816, 2014. URL <https://doi.org/10.1111/rssb.12050>. [p7]
- N. Matloff. Software alchemy: Turning complex statistical computations into embarrassingly-parallel ones. *Journal of Statistical Software, Articles*, 71(4):1–15, 2016. URL <https://doi.org/10.18637/jss.v071.i04>. [p7]

- Revolution Analytics and S. Weston. *foreach: Provides Foreach Looping Construct for R*, 2015a. URL <https://CRAN.R-project.org/package=foreach>. R package version 1.4.3. [p7]
- Revolution Analytics and S. Weston. *iterators: Provides Iterator Construct for R*, 2015b. URL <https://CRAN.R-project.org/package=iterators>. R package version 1.0.8. [p7]
- RITA. The Airline on-time performance data set website, 2009. URL <http://stat-computing.org/dataexpo/2009/>. Research and Innovation Technology Administration, Bureau of Transportation Statistics. [p6]
- O. Rodeh. B-trees, shadowing, and clones. *ACM Transactions on Storage (TOS)*, 3(4):Article No. 2, 2008. URL <https://doi.org/10.1145/1326542.1326544>. [p7]
- The Apache Software Foundation, 2013. Apache Hadoop Streaming, available at <http://hadoop.apache.org>. [p7]
- H. Wickham, J. Hester, and R. Francois. *readr: Read Tabular Data*, 2016. URL <https://CRAN.R-project.org/package=readr>. R package version 1.0.0. [p7]
- M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10:10–10, 2010. URL <http://dl.acm.org/citation.cfm?id=1863103.1863113>. [p7]
- J. Zhou and K. A. Ross. Implementing database operations using SIMD instructions. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 145–156. ACM, 2002. [p7]

Taylor Arnold
University of Richmond
28 Westhampton Way, Richmond, VA
USA
tarnold2@richmond.edu

Michael J. Kane
Yale University
300 George Street, New Haven, CT
USA
michael.kane@yale.edu

Simon Urbanek
AT&T Labs – Statistics Research
1 AT&T Way, Bedminster, NJ
USA
urbanek@research.att.com

IsoGeneGUI: Multiple Approaches for Dose-Response Analysis of Microarray Data Using R

by Martin Otava, Rudradev Sengupta, Ziv Shkedy, Dan Lin, Setia Pramana, Tobias Verbeke, Philippe Haldermans, Ludwig A. Hothorn, Daniel Gerhard, Rebecca M. Kuiper, Florian Klinglmueller and Adetayo Kasim

Abstract The analysis of transcriptomic experiments with ordered covariates, such as dose-response data, has become a central topic in bioinformatics, in particular in omics studies. Consequently, multiple R packages on CRAN and Bioconductor are designed to analyse microarray data from various perspectives under the assumption of order restriction. We introduce the new R package IsoGene Graphical User Interface (**IsoGeneGUI**), an extension of the original **IsoGene** package that includes methods from most of available R packages designed for the analysis of order restricted microarray data, namely **orQA**, **ORIClust**, **goric** and **ORCME**. The methods included in the new **IsoGeneGUI** range from inference and estimation to model selection and clustering tools. The **IsoGeneGUI** is not only the most complete tool for the analysis of order restricted microarray experiments available in R but also it can be used to analyse other types of dose-response data. The package provides all the methods in a user friendly fashion, so analyses can be implemented by users with limited knowledge of R programming.

Introduction

Modelling the dose-response relationship plays an important role in the drug discovery process in the pharmaceutical industry. Typical responses are efficacy or toxicity measures that are modelled with the aim of identifying the dose that is simultaneously efficacious and safe (Pinheiro et al., 2006). The recent development of microarray technology introduced gene expression level as an additional important outcome related to dose. Genes, for which the expression level changes over the dose of the experimental drug, are of interest, since they provide insight into efficacy, toxicity and many other phenotypes. Order restriction is often assumed in the dose-response modelling, usually in terms of monotone trend (Lin et al., 2012b). The restriction is a consequence of the assumption that higher dose levels induce stronger effects in the response (either increasing or decreasing). However, order restriction can also be related to umbrella profiles. In such a case, monotonicity is assumed up to a certain dose level and the direction of the dose-response relationship changes thereafter (Bretz and Hothorn, 2003).

Order restricted analysis received a lot of attention in previous years and several R packages were developed for this purpose. Specifically, the R packages **IsoGene** (Lin et al., 2013 and Pramana et al., 2010) and **orQA** (Klinglmueller, 2010) were developed for inference, **goric** (Gerhard and Kuiper, 2012 and Kuiper and Hoijtink, 2013) for model selection, and **ORCME** (Kasim et al., 2014) and **ORIClust** (Liu et al., 2012) were developed for order restricted clustering of genes.

Inference consists of testing a null hypothesis of a no dose-response relationship, against an ordered alternative. Multiplicity correction needs to be applied due to the large number of tests. The model selection framework quantifies the expected relative distance of a given model to the true underlying model in order to select the best model among a set of candidate models. The model selection approach is basis for the identification of the minimal effective dose or lowest-observed-adverse-effect level (Kuiper et al., 2014). Order restricted clustering is a data analysis approach which aims to form subsets of genes with similar expression profiles. It can be very useful when reference genes are available and the aim of the analysis is to identify genes that behave in a similar way to the reference genes. The clusters can be formed in unsupervised way and the genes that share cluster with reference genes can be identified. Additionally, the resulting clusters can be used to establish potential pathways and gene sets that react to the exposure in close agreement.

All the different methods mentioned above were scattered across multiple specialized packages. The **IsoGeneGUI** package is an envelope package in which all the methods are available together in user friendly framework, allowing to explore the gene expression data set with collection of state-of-the-art tools. The overview of the package structure is schematically shown in Figure 1.

Not all scientists performing microarray experiment analysis are necessarily educated in using R. Hence, the package **IsoGeneGUI** (Pramana et al., 2012) was originally created as a graphical user interface extension of the **IsoGene** package. The large number of **IsoGeneGUI** package downloads

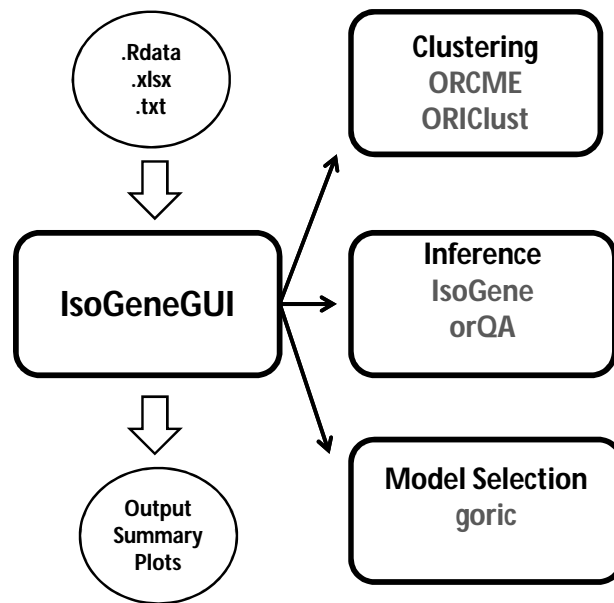


Figure 1: The general structure of the **IsoGeneGUI** package.

from the BioConductor (Gentleman et al., 2004) repository suggests that there is a demand for GUI data analysis tools for inference, model selection, estimation and order restricted clustering. Therefore, the **IsoGeneGUI** package was extended to embrace all currently available tools in one package. In addition to the data analysis tools for estimation, inference, model selection and clustering, the package contains many tools for exporting results, their visualization and easy handling of produced figures. Therefore, **IsoGeneGUI** provides the most complete and simultaneously user friendly data analysis tool, dealing with order restricted microarray experiments and other dose-response studies, that is currently available in R.

The aim of this manuscript is to provide a brief introduction to the package, both the underlying methodological aspects and its particular implementation are discussed. Methods for inference, estimation, clustering and model selection available in **IsoGeneGUI** package are introduced in following section. The structure of the package is described and details about implementation of the methods are given.

Modelling order-restricted dose-response data

Estimation under order restriction

The methodology described in this section has its roots in the maximum likelihood estimate (MLE) under the order constraints. The MLE is obtained by least squares minimization, with restriction on monotonicity of dose-specific means. The estimation procedure under such condition is called 'isotonic regression' (Barlow et al., 1972). It can be shown that the MLE can be obtained using the 'pool adjacent violators algorithm' (PAVA). The algorithm first computes the dose-specific means. If there is any violation of the monotonicity between any two estimates of means, it assigns to both of them their weighted average as new estimate for both means. The weights are proportional to the number of observations for particular dose. The procedure is repeated, until all the estimates comply with the monotonicity assumption.

Specifically, assume non-decreasing profile and denote μ_0, \dots, μ_{K-1} the dose-specific means and $n_i, i = 0, \dots, K-1$ number of observations per dose level i . The unrestricted MLE are equal to the dose-specific sample means denoted by $\hat{\mu}_0, \dots, \hat{\mu}_{K-1}$. Isotonic means $\hat{\mu}_0^*, \dots, \hat{\mu}_{K-1}^*$, i.e. the means under the assumption of monotonicity, are computed as $\hat{\mu}_j^* = \hat{\mu}_j, j = 0, \dots, K-1$, if the means fulfill $\hat{\mu}_j \leq \hat{\mu}_{j+1}$. If any $\hat{\mu}_j > \hat{\mu}_{j+1}$, then $\hat{\mu}_j^* = \hat{\mu}_{j+1}^* = (n_j \hat{\mu}_j + n_{j+1} \hat{\mu}_{j+1}) / (n_j + n_{j+1})$. The procedure is repeated iteratively, until it holds that $\hat{\mu}_0^* \leq \dots \leq \hat{\mu}_{K-1}^*$.

Inference

The main goal of the inference framework is to test the relationship between the dose level and the response of interest; gene expression in our case. The primary interest is to test the null hypothesis of no dose effect on the response, given by

$$H_0 : \mu_0 = \mu_1 = \mu_2 = \dots = \mu_{K-1}, \quad (1)$$

against an ordered (monotonic) alternative

$$H^{up} : \mu_0 \leq \mu_1 \leq \mu_2 \leq \dots \leq \mu_{K-1}, \text{ or } H^{dn} : \mu_0 \geq \mu_1 \geq \mu_2 \geq \dots \geq \mu_{K-1}, \quad (2)$$

with at least one strict inequality. Several test statistics for order restricted problems were developed over the last few decades. In the package, the following methods are available: likelihood-ratio test (LRT, [Barlow et al., 1972](#)), Williams' test statistic ([Williams, 1971](#)), Marcus' statistic ([Marcus, 1976](#)), M statistic ([Hu et al., 2005](#)) and modified M statistic ([Lin et al., 2007](#)). The different methods are sensitive to different possible underlying profiles, so there is no overall best method. The choice strongly depends on the context of interest. The LRT test is based on the ratio of a residual sum of squares under monotonicity over a residual sum of squares under the null hypothesis. Williams' test statistic is based on difference $\hat{\mu}_{K-1}^* - \hat{\mu}_0$, i.e. fold change between isotonic mean of last dose and sample mean under first dose. Marcus' statistics is a modification of Williams' that compares isotonic means $\hat{\mu}_{K-1}^* - \hat{\mu}_0^*$. The M and modified M test statistics are based on the same difference as Marcus' test, but they differ in a way how the estimation of standard error is approached. Detailed discussion about the methods, their usage and advantages and disadvantages can be found in [Lin et al. \(2007\)](#). The distribution of some of the test statistics cannot be derived analytically. Therefore, resampling based inference is implemented to approximate distribution of test statistics under the null model ([Westfall and Young, 1993](#) and [Ge et al., 2003](#)).

When the tests are performed for a large number of genes, the multiplicity adjustment is necessary. Otherwise, the significance level control would be compromised and large number of false positives is expected. In general, there are two approaches for multiplicity corrections. Either by controlling the probability of at least one false positive among the findings (Family Wise Error Rate, FWER) or alternatively by controlling the proportion of false positives among the findings (False Discovery Rate, FDR). The FWER can be controlled by Bonferroni ([Bonferroni, 1936](#)), Holm ([Holm, 1979](#)), Hochberg ([Hochberg and Benjamini, 1990](#)) or Šidák single-step and step-down ([Šidák, 1971](#)) procedures. The method of Bonferroni and Šidák's are conservative methods due to the assumption of independence among tests, especially in case of large number of tests performed simultaneously, which is often the case in microarray setting. Hochberg's method is more powerful, but it only provides control of FWER under assumption that there is non-negative dependence among the tests ([Hochberg and Benjamini, 1990](#)). Therefore, among the FWER methods, we suggest to use Holm's procedure, unless there is strong motivation otherwise. The procedure is uniformly more powerful than Bonferroni's method, but does not need positive dependence assumption to control FWER, as in case of Hochberg's method. The use of FDR instead of FWER is common in microarray studies. It translates into relaxing the control of false positives, while decreasing false negatives. It is suitable to use, if few false positives among findings are not a practical problem and we are mainly interested in identification of as many true positives as possible. The FDR can be controlled using the Benjamini-Hochberg (BH, [Benjamini and Hochberg, 1995](#)) or Benjamini-Yekutieli (BY, [Benjamini and Yekutieli, 2001](#)) procedures. Similarly to FWER, the BY-FDR method is valid only under positive dependence among the tests, so the BH-FDR should be used unless there is a strong motivation otherwise.

A common issue in gene expression inference is the presence of genes with relatively low variance that induce large values of the test statistics under consideration, although the magnitude of the effect is negligible. Formally, the genes are declared statistically significant, but from a biological point of view, these genes will not be further investigated due to small fold change. Significance Analysis of Microarrays (SAM, [Tusher et al., 2001](#)) was proposed as a solution for this issue by inflating the standard error.

Clustering

The **IsoGeneGUI** package provides two clustering approaches based on algorithms that incorporate order restrictions. The **ORCME** package implements the δ -clustering algorithm ([Kasim et al., 2012](#)) which is based on the δ -biclustering algorithm proposed by [Cheng and Church \(2000\)](#). The clustering is not applied on the data themselves, but on the isotonic means. Hence, it ignores the within dose variability and uncertainty about the mean estimation. Therefore, it is advised that the algorithm is applied either to a filtered data set (i.e. genes with fold change higher than given threshold) or on the genes showing significant dose-response profile (i.e. after the inference step). The method is sensitive

both to differences in shape and to the different magnitude of the effect (even if shape is similar). The clustering criterion in [Kasim et al. \(2012\)](#) is based on error sum of squares. The more robust version of the algorithm is achieved by using the median polish algorithm to compute the residuals ([Mosteller and Tukey, 1977](#)) and by replacing the squared error by an absolute value of error. This implies that less weight is put on outlying residuals and the clusters are allowed for greater deviations under the same degree of homogeneity.

The **ORIClust** package implements the one or two-stage Order Restricted Information Criterion Clustering algorithm (ORICC, [Liu et al., 2009](#), [Lin et al., 2012a](#)) which is based on an information criterion that takes into account order restrictions. The filtering step can be addressed within the algorithm itself. The ORICC algorithm considers different type of dose-response profiles, such as monotone profiles and umbrella profiles, that can be used for clustering. Umbrella profiles assume that the monotonicity holds up to a certain dose and then the trend changes the direction. Practical example, when such profiles are suitable, is overdosing with the drug, changing beneficial effect to the harmful one. In contrast to the clustering approach implemented in the δ -clustering method, the ORICC algorithm pulls together all monotone profiles. Hence, it is not suitable for the separation of non-decreasing monotone profiles with a true zero effect at some dose levels (i.e. some dose-specific means are equal) from strictly increasing profiles. This is the main difference between these two clustering algorithms, ORICC and δ -clustering, proposed by [Liu et al. \(2009\)](#) and [Lin et al. \(2012a\)](#), respectively. For that reason, they are both needed to provide a complete toolbox for an order restricted analysis of microarray data.

Model selection

The task of model selection procedures is to select the 'best' model out of the given set of possible models. The 'best' is translated into a combination of the likelihood, i.e. how well the model fits the data, and a penalty on the number of parameters (i.e. model complexity). The form of penalty distinguishes various methods developed over past decades. A model selection based method is implemented in the package **goric** using Generalized Order Restricted Information Criterion (GORIC, [Kuiper et al., 2011](#)). The GORIC method incorporates the information about the order constraints when calculating the information criteria. It extends the ORIC ([Anraku, 1999](#)) algorithm designed for simple monotone order restriction by allowing more complicated structure of constraints. The set of possible models is given and the output of the GORIC provides weights for each of these models ([Kuiper et al., 2014](#)). The weights can be interpreted as posterior model probabilities ([Lin et al., 2012a](#)). It is often the case that a null model is not considered in this step, if the method is applied conditionally on the inference step that selected genes with significant dose-response relationship. Based on the model weights, the best model or set of models can be selected. The main motivation for selecting one or more models is to estimate a quantity of interest that characterize dose-response relationship. For example, the determination of the minimum effective dose (MED). The MED is defined as the first dose that exhibits some effect of dose on the response. It can be estimated either based on the best model or as weighted average of several models, with weights proportional to the model weights ([Kuiper et al., 2014](#)).

The structure of the package

The package **IsoGeneGUI** encompasses all the methods mentioned in previous section. The summary is given in Table 1. The GUI was build using Tcl/Tk environment.

Package	Analysis type	Reference
IsoGene	Inference	Lin et al. (2012b)
orQA	Inference	Klingmueller (2010)
ORCME	Clustering	Kasim et al. (2014)
ORIClust	Clustering	Liu et al. (2012)
goric	Model selection	Gerhard and Kuiper (2012)

Table 1: Packages for the analysis of order-restricted dose-response gene expression data available on CRAN.

The **IsoGeneGUI** is freely available from R-Forge repository <https://r-forge.r-project.org/projects/isogenegui/>. It can be downloaded and run from R with commands:

```
install.packages("IsoGeneGUI", repos = "http://r-forge.r-project.org")
```

```
library(IsoGeneGUI)
IsoGeneGUI()
```

The main window of the package is shown in Figure 2. The top tab lists several submenus. First the submenu 'File' (A in Figure 2) allows to load the data set and to display the data values as table. The data compatible with package can be provided either as plain text file, Microsoft Excel spreadsheet or the .RData file. The submenu 'Analysis(HD)' (B) comprises the methods for inference, estimation and model selection, i.e. it contains the packages **IsoGene**, **orQA** and **goric**. The submenu 'Analysis(SD)' (C) is to be used when there is only one sample available. The clustering of the genes based on their profiles can be performed in a separate submenu (D), using the methods implemented in **ORCME** and **ORIClust**. Some of the plots can be obtained from the analysis windows, but more general plots are listed in the visualization techniques submenu (E). The graphical techniques listed in submenu D typically use outputs of the methods implemented in other submenus. The plots can be saved in multiple file types. The last submenu 'Help' (F) contains the help files for the **IsoGene** package, the **IsoGeneGUI** package and the vignette for **IsoGeneGUI**. The box in the center of the main window (G) gathers the results of the analyses and displays summary statistics of the results. Additionally, it serves as indicator of which outputs are currently active (if analysis was run multiple times) and will be plotted by visualization tools.

An example of the package interface is fully shown in Figure 3. We can see the main window again (A), now with the box showing the properties of active data set (A1) and a summary of results of a clustering procedure (A2). The window that was used for clustering with δ -clustering method is displayed on the left side of the Figure 3 (B) and the results are displayed in the table (C). One of the clusters was plotted using one of the visualization options (D). Further examples are shown in following section.

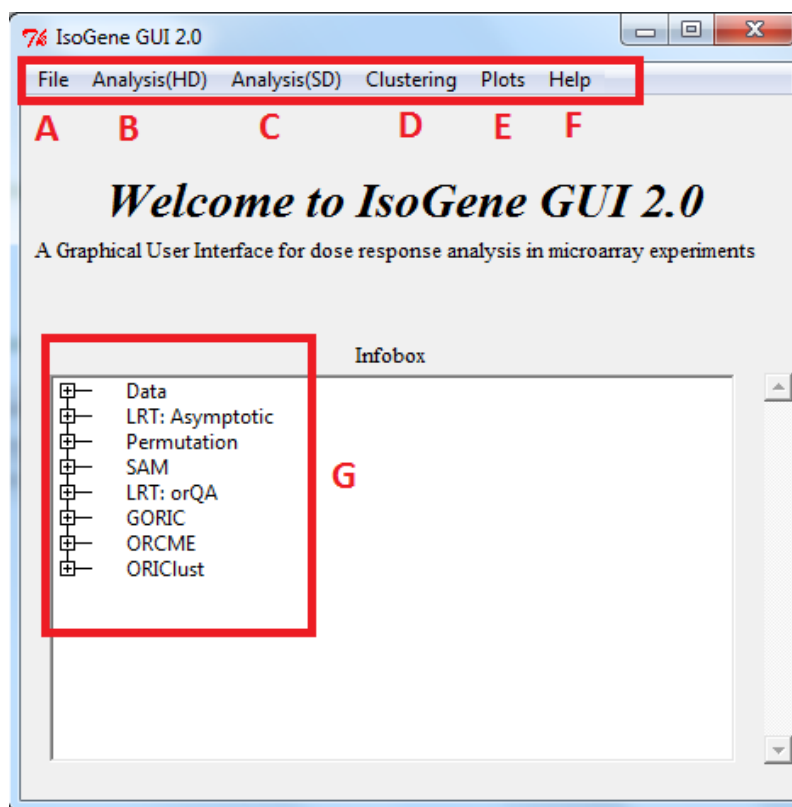


Figure 2: The **IsoGeneGUI** package main menu with highlighted submenus.

Applications

The **IsoGeneGUI** implementation of the available methods is less flexible than in original packages. That is natural trade-off between clarity and accessibility of options in GUI compared to plain R packages that are more flexible but also more difficult to operate without proficient knowledge of R. This section describes the implementation of the methods for inference, clustering and model selection. The examples shown in Figure 4 to Figure 6 were obtained using the example data set dopamine that is

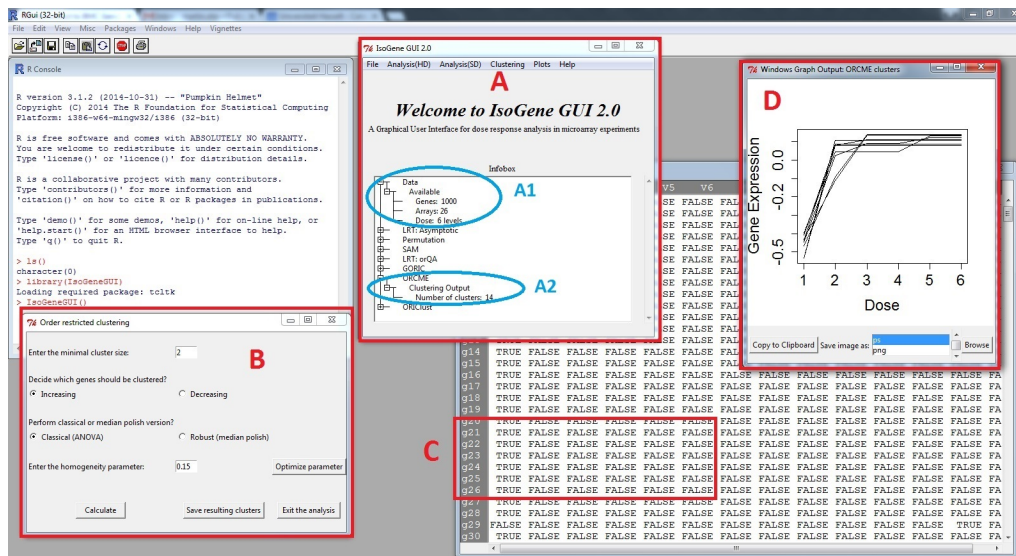


Figure 3: R with opened IsoGeneGUI package.

part of the **IsoGeneGUI** package. The dopamine data set is subset of a larger dose-response experiment (Lin et al., 2012b) and consist of 1000 genes with four doses and 3 arrays per dose. In each figure, one method is presented, accompanied with one of available graphical displays.

Inference

The permutation test is implemented for all five test statistics discussed above, using the functions from **IsoGene** package. For the LRT, a much faster implementation of the permutation test is available from **orQA**. Both methods produce the same result (within the sampling error), so the slower version should be used only in case that additional test statistics are of interest. Additionally, there is an asymptotic solution available for the LRT as well. Note that it is advised to avoid this option in case of small sample sizes.

The window that facilitates permutation test based on the **IsoGene** package is shown in Figure 4. The left panel shows the window itself. The top part allows to select the genes for which the raw p-values based on permutation test will be obtained. The middle part of window offers seven multiplicity adjustment methods and computation of significant genes based on any of the five test statistics. The last part produces three types of plots. The right panel of Figure 4 shows an example of one of the plots: the adjustment of p-value while controlling FDR. In this case, both BH and BY methods agreed on same set of genes, but that is not necessarily case in general. For FDR equal to 5%, we expect three false discoveries among the 62 null hypotheses that were rejected. The left panel of Figure 5 shows the window for the LRT using the **orQA** package, providing nearly same options as permutation method. The right panel of Figure 5 shows example of so called 'volcano plot' that compares the $-\log(p\text{-value})$ and fold change. Note that the high value for $-\log(p\text{-value})$ of genes with fold change around zero is often caused by a small variance among the observations of these genes. This is an indication that the SAM method should be applied (Lin et al., 2012b).

Clustering

Order restricted clustering is addressed by two algorithms, the δ -clustering from **ORCME** and the ORICC from **ORIClust**. As mentioned above, the package contains two versions of the δ -clustering method: clustering based on the least squares and a robust clustering based on least of absolute residuals. The ORCME window and output is shown in Figure 3. The window implementing ORICC is shown in left panel of Figure 6. All monotone and umbrella profiles are automatically considered and the user cannot influence this setting. However, this setting provides the flexible framework for clustering. The complete profile can be included to the set as well. One or two-stage type of ORICC can be run and output is automatically saved in both text and visual form. The clustering results are shown in right panel of Figure 6 for case in which the top 30 genes are kept for final clustering step.

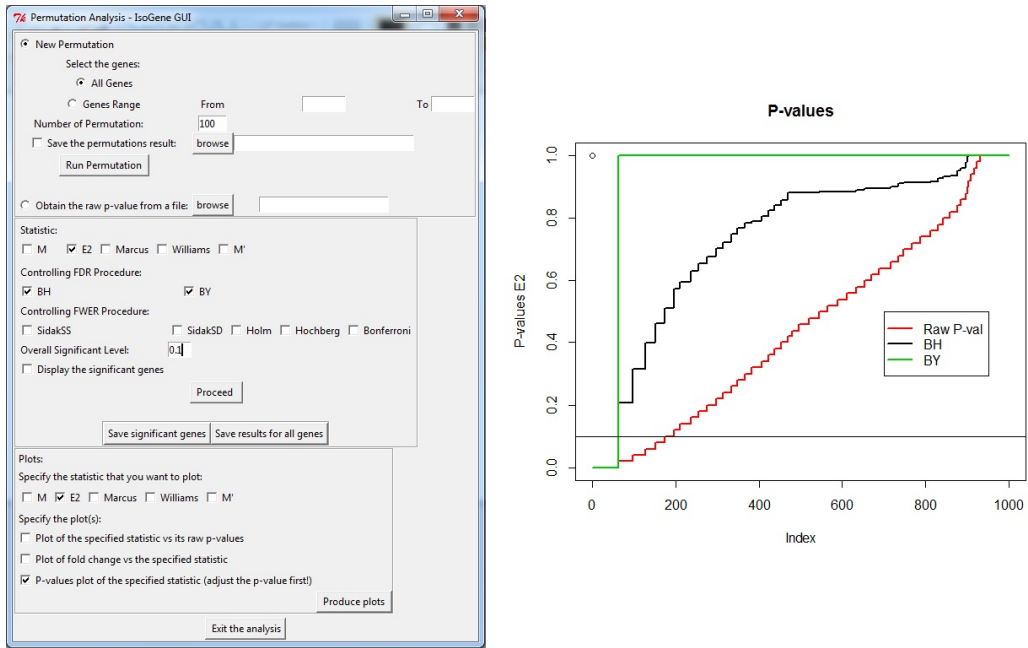


Figure 4: Resampling based inference. Left panel: The window for performing permutation test. Right panel: Plot of an effect of multiplicity adjustment.

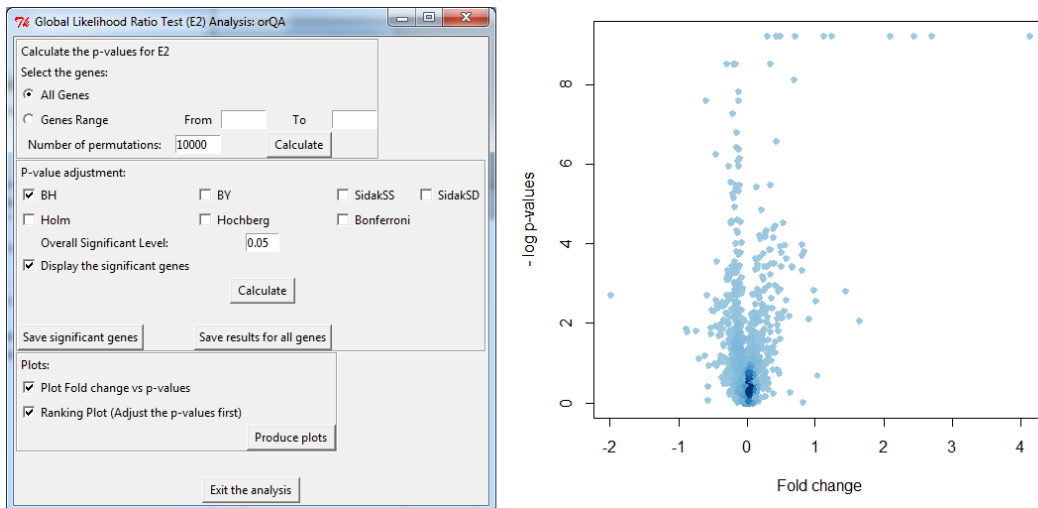


Figure 5: Inference with orQA. Left panel: The window for performing LRT. Right panel: Volcano plot.

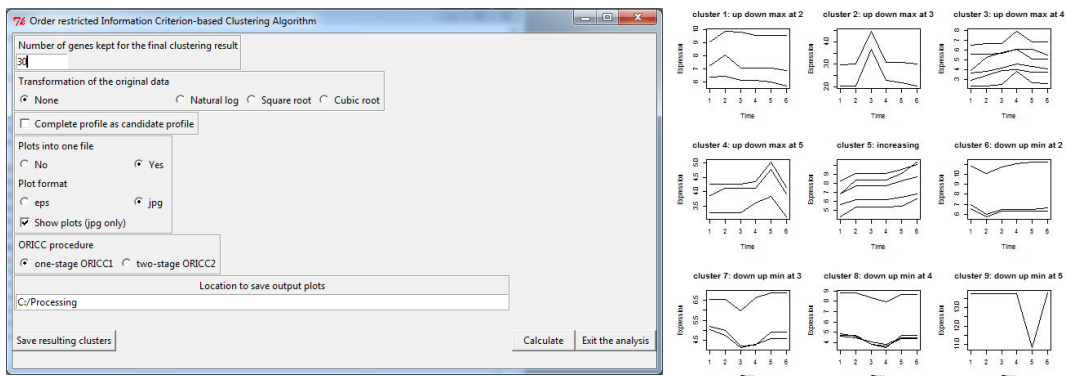


Figure 6: Order restricted clustering using ORIClust. Left panel: The window for clustering. Right panel: Plot of all the resulting clusters.

Model selection

The current implementation in **IsoGeneGUI** runs automatically GORIC for all possible models for a given direction (upward or downward trends). Therefore, for an experiment with control and $K - 1$ dose levels, 2^{K-1} models are considered, including the null model of no dose effect. In case that some of these models are a priori not considered for the analysis, the weights can be easily normalized for the smaller set of models. Only one gene at the time can be analyzed using the GORIC procedure, due to computational intensity of the derivation of the model weights. For the dopamine data, there are six dose levels and therefore, for an upward trend that are 32 possible monotone non-decreasing models (including the null model).

Input and output

The **IsoGeneGUI** package accepts the data sets to be provided either as plain text file, Microsoft Excel spreadsheet or the .RData file. The structure of the data set can vary slightly in case of two former file types, but in general, the structure need to be followed where genes are listed in rows and columns represent the different conditions (doses). In case of .RData, both microarray results and dose data can be passed at once, as it is done in case of the example data set available in the package.

Similarly, the output of the procedures can be saved either as Excel spreadsheet or as .RData file. In general, all the intermediate results can be retrieved, if needed, rather than final result of each procedure only. In addition, the visualization tools mentioned above provides an option to save figures in various formats as well as copy them directly to the clipboard. The goal is to provide the user with option to retrieve most of the relevant results, while emphasizes is put on final results of the procedures that are easily visualized and typically shown on the screen automatically, once the method is finished.

Although the **IsoGeneGUI** package was developed for the analysis of microarray dose-response experiments, it can be used for an analysis of any high dimensional data in a dose-response setting in which the basic data structure is a matrix, given in (3), with n variables (or features, the rows of the matrix) and m observations (or conditions, samples, the columns of the matrix).

$$\mathbf{X} = \begin{pmatrix} X_{11} & X_{12} & \dots & X_{1m} \\ X_{21} & X_{22} & \dots & X_{2m} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ X_{n1} & X_{n2} & \dots & X_{nm} \end{pmatrix}. \quad (3)$$

In addition, the labeling vector \mathbf{R} which links each j th sample to a dose related to the sample is given by

$$\mathbf{R} = (d_1, d_1, \dots, d_1, \quad d_2, d_2, \dots, d_2, \quad \dots \quad d_K, d_K, \dots, d_K). \quad (4)$$

In the manuscript, \mathbf{X} refers to a gene expression data (as in Section L1000 Dataset) but the package can be used to analyze any high dimensional experiment for which \mathbf{X} and \mathbf{R} are available. For example, the package can be used for RNA-seq data analysis, after normalization and transformation proposed by Law et al. (2014) for an analysis of RNA-seq using **limma**. Furthermore, the case that $n = 1$, i.e.

$$\mathbf{X} = (X_{11} \quad X_{12} \quad \dots \quad X_{1m}), \quad (5)$$

corresponds to a setting in which a dose-response experiment was conducted for one variable of interest. Such an experiment can be analyzed using the **IsoGeneGUI** package as well. An example of such an analysis is given in Section Angina Dataset.

L1000 dataset

The L1000 database (<http://www.lincsproject.org/>) is one of the very new microarray datasets, which is of interest to many researchers in this field, nowadays. After analyzing several sources of gene expression data, it was noticed that 1000 carefully selected landmark genes can explain approximately 80% of the information and the dataset produced using this set of genes is known as L1000 dataset. It is essentially a high-throughput gene expression microarray dataset in which cultured cells are treated with various chemical and genetic perturbations and the corresponding transcriptional responses are measured at different concentrations. A computational pipeline is used for data-processing, where raw fluorescence intensity is converted into differential gene expression signatures and the data at each stage of this pipeline are available. The **IsoGeneGUI** package was used for analysis of the gene expression data from level 2 in order to identify statistically significant genes with respect to

dose-response profile. Figure 7 shows an example in which the **IsoGeneGUI** package is used to analyse L1000 gene data for a specific compound (BRD-A19037878). All the options available under the tab 'Analysis(HD)' can be used for this dataset as well.

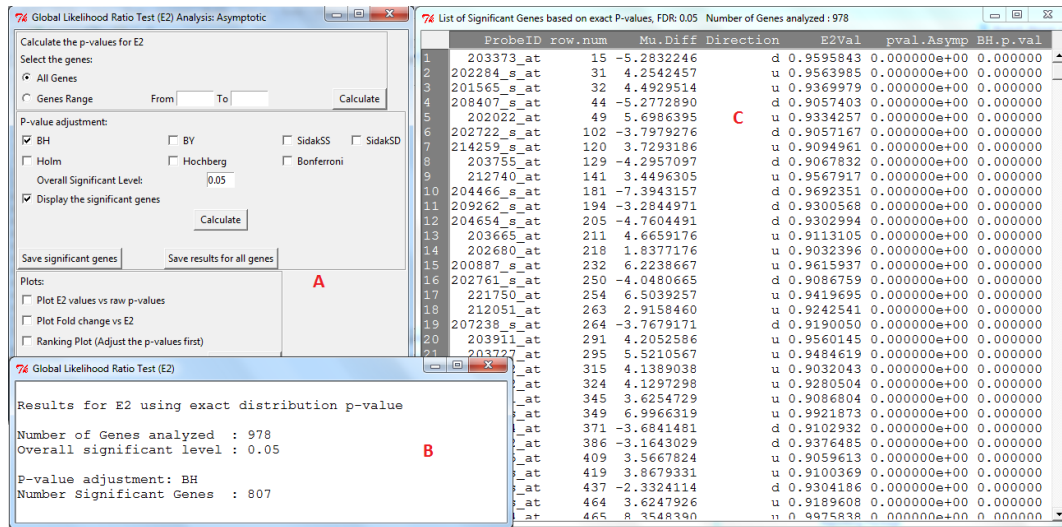


Figure 7: L1000 data. Panel A shows the GUI window for Likelihood Ratio Test. Panel B gives summary of the test results and panel C displays the test results in details.

Angina dataset

This example demonstrates usability of the package on data from a dose-response experiment with a single response variable. The Angina data set (Westfall et al., 1999, p. 164) represents dose-response study of a drug to treat angina pectoris. The response is the duration (in minutes) of pain-free walking after treatment relative to the values before treatment. Four active doses were used together with a control dose with placebo only. Ten patients per dose were examined. Large values indicate positive effects on patients. The data were used in Kuiper et al. (2014) and are available under the name *angina* in the package *mratios* (Djira et al., 2012) of the R software. Figure 8 displays the results of the GORIC analysis when this dataset is used. The GORIC window is shown in left panel of Figure 8. The middle plot of Figure 8 displays the data and the model with highest weights, M_{15} , increasing in all dose levels. The right panel shows the weights for all the fitted models.

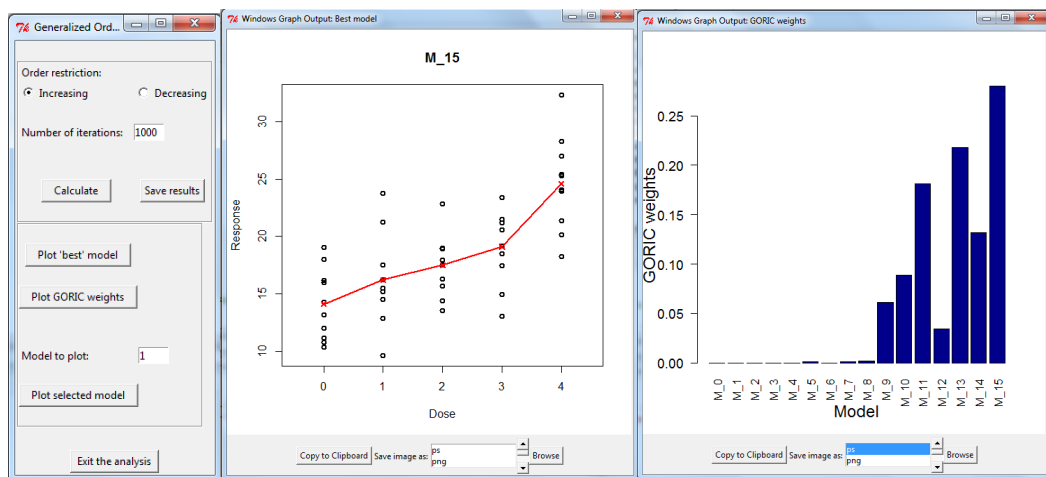


Figure 8: Angina Dataset. The GORIC method when only one sample is available (Angina data). Left panel: The window for providing the inputs required to perform the analysis. Middle panel: Dose-response relationship under model M15. Right panel: GORIC weights for all the models fitted.

Summary

The analysis of dose-response relationship for order restricted experiments is highly relevant in the drug discovery process. Multiple R packages offer methodology within this framework. The new version of the **IsoGeneGUI** package encompasses a wide range of these packages in a unified way. The package contains data analysis tools for estimation, inference, model selection and clustering. To our knowledge, it is the only software package providing such a wide range of tools simultaneously. Additionally, the GUI implementation of the package allows non-statisticians to conduct the analysis with only minimal knowledge of R.

Although the **IsoGeneGUI** package was developed for the analysis of microarray dose-response experiments, it can be used for an analysis of any high dimensional data in a dose-response setting. For example, for analysis of RNA-seq data, the package can be used after normalization and transformation in same way that it is done by Law et al. (2014). Furthermore, it can be used for the case of single dose-response experiment.

In summary, the **IsoGeneGUI** package is a state-of-the-art collection of methodologies covering a wide range of analyses that are meaningful for order restricted microarray experiments as well as in more general setting of dose-response experiments. Moreover, the package can be used in a straightforward way by the general scientific community.

Acknowledgement

This work was supported by the Interuniversity Attraction Poles Research Network P7/06 of the Belgian State (Belgian Science Policy) and the Research Project of Hasselt University (BOF11DOC09; to MO).

Bibliography

- K. Anraku. An information criterion for parameters under a simple order restriction. *Biometrika*, 86: 141–152, 1999. URL <https://doi.org/10.1093/biomet/86.1.141>. [p17]
- R. E. Barlow, D. J. Bartholomew, M. J. Bremner, and H. D. Brunk. *Statistical Inference under Order Restriction*. John Wiley & Sons, 1972. [p15, 16]
- Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society B*, 57:289–300, 1995. [p16]
- Y. Benjamini and D. Yekutieli. The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics*, 29(4):1165–1188, 2001. URL <https://doi.org/10.1214/aos/1013699998>. [p16]
- C. E. Bonferroni. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936. [p16]
- F. Bretz and L. A. Hothorn. Statistical analysis of monotone or non-monotone dose-response data from *in Vitro* toxicological assays. *Alternatives to Lab Animals*, 31(Supplement 1):81–96, 2003. [p14]
- Y. Cheng and G. M. Church. Biclustering of expression data. *Proceedings of the Conference on Intelligent Systems for Molecular Biology*, 55:93–104, 2000. [p16]
- G. D. Djira, M. Hasler, D. Gerhard, and F. Schaarschmidt. *Mratios: Inferences for Ratios of Coefficients in the General Linear Model*, 2012. URL <http://CRAN.R-project.org/package=mratio>. [p22]
- Y. Ge, S. Dudoit, and T. P. Speed. Resampling-based multiple testing for microarray data analysis. *Test*, 12(1):1–77, 2003. URL <https://doi.org/10.1007/bf02595811>. [p16]
- R. C. Gentleman, V. J. Carey, D. M. Bates, and et al. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004. URL <https://doi.org/10.1186/gb-2004-5-10-r80>. [p15]
- D. Gerhard and R. M. Kuiper. *goric: Generalized Order-Restricted Information Criterion*, 2012. URL <http://CRAN.R-project.org/package=goric>. R package version 0.0-7. [p14, 17]
- Y. Hochberg and Y. Benjamini. More powerful procedures for multiple significance testing. *Statistics in Medicine*, 9:811–818, 1990. URL <https://doi.org/10.1002/sim.4780090710>. [p16]

- S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6: 65–70, 1979. [p16]
- J. Hu, M. Kapoor, W. Zhang, S. R. Hamilton, and K. R. Coombes. Analysis of dose response effects on gene expression data with comparison of two microarray platforms. *Bioinformatics*, 21(17):3524–3529, 2005. URL <https://doi.org/10.1093/bioinformatics/bti592>. [p16]
- A. Kasim, S. Van Sanden, M. Otava, S. Hochreiter, D. Clevert, W. Talloen, and D. Lin. δ -clustering of monotone profiles. In D. Lin, Z. Shkedy, D. Yekutieli, D. Amaratunga, and L. Bijmens, editors, *Modeling Dose-Response Microarray Data in Early Drug Development Experiments Using R*, pages 135–149. Springer-Verlag, 2012. URL <https://doi.org/10.1007/978-3-642-24007-2>. [p16, 17]
- A. Kasim, M. Otava, and T. Verbeke. *ORCME: Order Restricted Clustering for Microarray Experiments*, 2014. URL <http://CRAN.R-project.org/package=ORCME>. R package version 2.0. [p14, 17]
- F. Klinglmueller. *orQA: Order Restricted Assessment of Microarray Titration Experiments*, 2010. URL <http://CRAN.R-project.org/package=orQA>. R package version 0.2.1. [p14, 17]
- R. M. Kuiper and H. Hoijtink. A Fortran 90 program for the generalization of the order-restricted information criterion. *Journal of Statistical Software*, 54:1–19, 2013. URL <https://doi.org/10.18637/jss.v054.i08>. [p14]
- R. M. Kuiper, H. Hoijtink, and M. J. Silvapulle. An Akaike-type information criterion for model selection under inequality constraints. *Biometrika*, 98:495–501, 2011. [p17]
- R. M. Kuiper, D. Gerhard, and L. A. Hothorn. Identification of the minimum effective dose for normally distributed endpoints using a model selection approach. *Statistics in Biopharmaceutical Research*, 6(1): 55–66, 2014. URL <https://doi.org/10.1080/19466315.2013.847384>. [p14, 17, 22]
- C. W. Law, Y. Chen, W. Shi, and G. K. Smyth. Voom: Precision weights unlock linear model analysis tools for rna-seq read counts. *Genome Biology*, 2014. URL <https://doi.org/10.1186/gb-2014-15-2-r29>. [p21, 23]
- D. Lin, Z. Shkedy, D. Yekutieli, T. Burzykowki, H. W. H. Göhlmann, A. De Bondt, T. Perera, T. Geerts, and L. Bijmens. Testing for trend in dose-response microarray experiments: Comparison of several testing procedures, multiplicity, and resampling-based inference. *Statistical Application in Genetics and Molecular Biology*, 6(1):Article26, 2007. URL <https://doi.org/10.2202/1544-6115.1283>. [p16]
- D. Lin, Z. Shkedy, and M. Aerts. Classification of monotone gene profiles using information theory selection methods. In D. Lin, Z. Shkedy, D. Yekutieli, D. Amaratunga, and L. Bijmens, editors, *Modeling Dose-Response Microarray Data in Early Drug Development Experiments Using R*, pages 151–164. Springer-Verlag, 2012a. URL <https://doi.org/10.1007/978-3-642-24007-2>. [p17]
- D. Lin, Z. Shkedy, D. Yekutieli, D. Amaratunga, and L. Bijmens, editors. *Modeling Dose-Response Microarray Data in Early Drug Development Experiments Using R - Order Restricted Analysis of Microarray Data*. Springer-Verlag, 2012b. URL <https://doi.org/10.1007/978-3-642-24007-2>. ISBN 978-3-642-24006-5. [p14, 17, 19]
- D. Lin, S. Pramana, T. Verbeke, and M. Otava. *IsoGene: Testing for Monotonic Relationship between Gene Expression and Doses in a Microarray Experiment*, 2013. URL <http://CRAN.R-project.org/package=IsoGene>. R package version 1.0-22. [p14]
- T. Liu, N. Lin, S. Ningzhong, and B. Zhang. Information criterion-based clustering with order-restricted candidate profiles in short time-course microarray experiments. *BMC Bioinformatics*, 10:146, 2009. URL <https://doi.org/10.1186/1471-2105-10-146>. [p17]
- T. Liu, N. Lin, N. Shi, and B. Zhang. *ORIClust: Order-Restricted Information Criterion-Based Clustering Algorithm*, 2012. URL <http://CRAN.R-project.org/package=ORIClust>. R package version 1.0-1. [p14, 17]
- R. Marcus. The powers of some tests of the equality of normal means against an ordered alternative. *Biometrika*, 63:177–183, 1976. URL <https://doi.org/10.2307/2335100>. [p16]
- F. Mosteller and J. W. Tukey. *Data Analysis and Regression*. Addison-Wesley, 1977. [p17]
- J. C. Pinheiro, F. Bretz, and M. Branson. Analysis of dose response studies: Modeling approaches. In N. Ting, editor, *Dose Finding in Drug Development*, pages 146–171. Springer-Verlag, 2006. URL <https://doi.org/10.1007/0-387-33706-7>. [p14]

- S. Pramana, D. Lin, P. Haldermans, Z. Shkedy, T. Verbeke, H. W. H. Göhlmann, A. De Bondt, W. Talloen, and L. Bijmens. IsoGene: An R package for analyzing dose-response studies in microarray experiments. *The R Journal*, 2(1):5–12, 2010. [p14]
- S. Pramana, D. Lin, P. Haldermans, and T. Verbeke. *IsoGeneGUI: A Graphical User Interface to Conduct a Dose-Response Analysis of Microarray Data*, 2012. URL <http://www.ibiostat.be/software/IsoGeneGUI/index.html>. R package version 1.20.0. [p14]
- V. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences*, 98:5116–5121, 2001. URL <https://doi.org/10.1073/pnas.091062498>. [p16]
- Z. Šidák. On probabilities of rectangles in multivariate student distributions: Their dependence on correlations. *The Annals of Mathematical Statistics*, 42:169–175, 1971. URL <https://doi.org/10.1214/aoms/1177693504>. [p16]
- P. H. Westfall and S. S. Young. *Resampling-Based Multiple Testing: Examples and Methods for p-Value Adjustment*. John Wiley & Sons, 1993. [p16]
- P. H. Westfall, R. D. Tobias, D. Rom, R. D. Wolfinger, and Y. Hochberg. *Multiple Comparisons and Multiple Tests Using the SAS System*. SAS Institute Inc., 1999. [p22]
- D. A. Williams. A test for differences between treatment means when several dose levels are compared with a zero dose control. *Biometrics*, 27:103–117, 1971. URL <https://doi.org/10.2307/2528930>. [p16]

Martin Otava

*Interuniversity Institute for Biostatistics and Statistical Bioinformatics
Universiteit Hasselt, Martelarenlaan 32, B-3500 Hasselt
Belgium
martin.otava@uhasselt.be*

Rudradev Sengupta

*Interuniversity Institute for Biostatistics and Statistical Bioinformatics
Universiteit Hasselt, Martelarenlaan 32, B-3500 Hasselt
Belgium
rudradev.sengupta@uhasselt.be*

Ziv Shkedy

*Interuniversity Institute for Biostatistics and Statistical Bioinformatics
Universiteit Hasselt, Martelarenlaan 32, B-3500 Hasselt
Belgium
ziv.shkedy@uhasselt.be*

Dan Lin

*Interuniversity Institute for Biostatistics and Statistical Bioinformatics
Universiteit Hasselt, Martelarenlaan 32, B-3500 Hasselt
Belgium
and
GlaxoSmithKline
Rue de l'institut 89, 1330 Rixensart
Belgium
Dan.8.lin@gsk.com*

Setia Pramana

*Sekolah Tinggi Ilmu Statistik/Institute of Statistics
Jl. Otto Iskandardinata No. 64C, Jakarta 13330
Indonesia
and
Department of Medical Epidemiology and Biostatistics, Karolinska Institutet
PO Box 281, SE-171 77 Stockholm
Sweden
setia.pramana@ki.se*

Tobias Verbeke
Open Analytics NV
Jupiterstraat 20, B-2600 Antwerp
Belgium
tobias.verbeke@openanalytics.eu

Philippe Haldermans
PXL-IT, PXL University College
Elfdé-Liniestraat 24, B-3500 Hasselt
Belgium
Philippe.Haldermans@pxl.be

Ludwig A. Hothorn
Institute of Biostatistics, Leibniz University Hannover
Herrenhaeuserstr. 2, D-30419 Hannover
Germany
hothorn@biostat.uni-hannover.de

Daniel Gerhard
School of Mathematics and Statistics, University of Canterbury
Private Bag 4800, 8140 Christchurch
New Zealand
daniel.gerhard@canterbury.ac.nz

Rebecca M. Kuiper
Department of Methodology & Statistics, Utrecht University
Padualaan 14, 3584 CH Utrecht
The Netherlands
R.M.Kuiper@uu.nl

Florian Klinglmueller
Section for Medical Statistics, Center for Medical Statistics and Informatics, Medical University of Vienna
Spitalgasse 23, A-1090 Wien
Austria
float@elefant.net

Adetayo Kasim
Wolfson Research Institute for Health and Wellbeing, Durham University
Queen's Campus, University Boulevard, Stockton-on-Tees, TS17 6BH
United Kingdom
a.s.kasim@durham.ac.uk

Network Visualization with `ggplot2`

by Sam Tyner, François Briatte and Heike Hofmann

Abstract This paper explores three different approaches to visualize networks by building on the grammar of graphics framework implemented in the `ggplot2` package. The goal of each approach is to provide the user with the ability to apply the flexibility of `ggplot2` to the visualization of network data, including through the mapping of network attributes to specific plot aesthetics. By incorporating networks in the `ggplot2` framework, these approaches (1) allow users to enhance networks with additional information on edges and nodes, (2) give access to the strengths of `ggplot2`, such as layers and facets, and (3) convert network data objects to the more familiar data frames.

Introduction

There are many kinds of networks, and networks are extensively studied across many disciplines (Watts, 2004). For instance, social network analysis is a longstanding and prominent sub-field of sociology, and the study of biological networks, such as protein-protein interaction networks or metabolic networks, is a notable sub-field of biology (Prell, 2011; Junker and Schreiber, 2008). In addition, the ubiquity of social media platforms, like Facebook, Twitter, and LinkedIn, has brought the concepts of networks out of academia and into the mainstream. Though these disciplines and the many others that study networks are themselves very different and specialized, they can all benefit from good network visualization tools.

Many R packages already exist to manipulate network objects, such as `igraph` by Csardi and Nepusz (2006), `sna` by Butts (2014), and `network` by Butts et al. (2014) (Butts, 2008, see also). Each one of these packages were developed with a focus of analyzing network data and not necessarily for rendering visualizations of networks. Though these packages do have network visualization capabilities, visualization was not intended as their primary purpose. This is by no means a critique or an inherently negative aspect of these packages: they are all hugely important tools for network analysis that we have relied on heavily in our own work. We have found, however, that visualizing network data in these packages requires a lot of extra work if one is accustomed to working with more common data structures such as vectors, data frames, or arrays. The visualization tools in these packages require detailed knowledge of each one of them and their syntax in order to build meaningful network visualizations with them. This is obviously not a problem if the user is very familiar with network structures and has already spent time working with network data. If, however, the user is new to network data or is more comfortable working with the aforementioned common data structures, they could find the learning curve for these packages burdensome.

The packages described in this paper have, by contrast, have one primary purpose: to create beautiful network visualizations by providing a wrapper of existing network layout capabilities (see for example the `statnet` suite of packages by Handcock et al. (2008)) to the popular `ggplot2` package (Wickham, 2016). And so, our focus here is not on adding to the analysis of network data or to the field of graph drawing, (cf. Tamassia, 2013) but rather it is on implementing existing graph drawing capabilities in the `ggplot2` framework, using the common data frame structure. The `ggplot2` package is hugely popular, and many other packages and tools interface with it in order to better visualize a wide variety of data types. By creating a `ggplot2` implementation, we hope to place network visualization within a large, active community of data visualization enthusiasts, bringing new eyes and potentially new innovations to the field of network visualization. With our approaches, we have two primary audiences in mind. The first audience is made up of frequent users of network structures and those who are fluent in the language of packages such as `network` or `igraph`. This audience will find that two of our three approaches (`ggnet2` and `ggnetwork`) directly incorporate the network structures and functions with which they are familiar with into the less familiar visualization paradigm of `ggplot2` (Briatte, 2016). The second audience, targeted by `geomnet`, consists of those users who are not familiar with network structures, but are familiar with data manipulation and tidying, and who happen to find themselves examining some data that can be expressed as a network (Tyner and Hofmann, 2016a). For this audience, we do the heavy network lifting internally, while also relying on their familiarity with `ggplot2` externally.

The `ggplot2` package was designed as an implementation of the ‘grammar of graphics’ proposed by Wilkinson (1999), and it has become extremely popular among R users.¹

¹In order to give an indication of how large the user base of `ggplot2` is, we looked at its usage statistics from January 1, 2016 to December 31, 2016 (see <http://cran-logs.rstudio.com/>). Over this period, the `ggplot2` package was downloaded over 3.2 million times from CRAN, which amounts to almost 9,000 downloads per day. Almost 800 R packages import or depend on `ggplot2`.

Because the syntax implemented in the **ggplot2** package is extendable to different kinds of visualizations, many packages have built additional functionality on top of the **ggplot2** framework. Examples include the **ggmap** package by Kahle and Wickham (2013) for spatial visualization, the **ggfortify** package for visualizing statistical models (see Horikoshi and Tang (2016), Tang et al. (2016)), the package **GGally** by Schloerke et al. (2016), which encompasses various complementary visualization techniques to **ggplot2**, and the **ggbio** and **ggtree** Bioconductor packages by Yin et al. (2012) and Yu et al. (2017), which both provide visualizations for biological data. These packages have expanded the utility of **ggplot2**, likely resulting in an increase of its user base. We hope to appeal to this user base and potentially add to it by applying the benefits of the grammar of graphics implemented in **ggplot2** to network visualization.

Our efforts rely upon recent changes to **ggplot2**, which allow users to more easily extend the package through additional geometries or ‘geoms’.²

In the remainder of this paper, we present three different approaches to network visualization through **ggplot2** wrappers. The first is a function, `ggnet2` from the **GGally** package, that acts as a wrapper around a network object to create a **ggplot2** graph. The second is a package, **geomnet**, that combines all network pieces (nodes, edges, and labels) into a single geom and is intended to look the most like other **ggplot2** geoms in use. The final is another package, **ggnetwork**, that performs some data manipulation and aliases other geoms in order to layer the different network aspects one on top of the other. The section [Brief introduction to networks](#) introduces the basic terminology of networks and illustrates their ubiquity in natural and social life. The next section [Three implementations of network visualizations](#) then discusses the structure and capabilities of each of the three approaches that we offer. The section [Examples](#) extends that discussion through several examples ranging from simple to complex networks, for which we provide the code corresponding to each approach alongside its graphical result. We follow with some considerations of runtime behavior in plotting networks in the section [Some considerations of speed](#) before closing with a discussion.

Brief introduction to networks

In its essence, a *network* is simply a set of vertices connected in pairs by a set of edges (Newman, 2010). Throughout this paper, we also use the term *node* to refer to vertices, as well as the terms *ties* or *relationships* to refer to edges, depending on context. The two sets of graphical objects that make up a network visualization, points and segments between them, have been used to examine a huge variety and quantity of information across many different fields of study. For instance, networks of scientific collaboration, a food web of marine animals, and American college football games are all covered in a paper on community detection in networks by Girvan and Newman (2002). Additionally, Buldyrev et al. (2010) study node failure in interdependent networks like power grids. Social networks such as links between television and film actors found on <http://www.imdb.com/> and neural networks, like the completely mapped neural network of the *C. elegans* worm are also extensively studied (Watts and Strogatz, 1998).

These examples show that networks can vary widely in scope and complexity: the smallest connected network is simply one edge between two vertices, while one of the most commonly used and most complex networks, the world wide web, has billions of vertices (Web pages) and billions of edges (hyperlinks) connecting them. Additionally, the edges in a network can be directed or undirected: *directed* edges represent an ordering of vertices, like a relationship extending from one vertex to another, where switching the direction would change the structure of the network. The World Wide Web is an example of a directed network because hyperlinks connect one Web page to another, but not necessarily the other way around. *Undirected* edges are simply connections between vertices where order does not matter. Co-authorship networks are examples of undirected networks, where nodes are authors and they are connected by an edge if they have written an academic publication together.

As a reference example, we turn to a specific instance of a social network. A *social network* is a network that everyone is a part of in one way or another, whether through friends, family, or other human interactions. We do not necessarily refer here to social media like Facebook or LinkedIn, but rather to the connections we form with other people. To demonstrate the functionality of our tools for plotting networks, we have chosen an example of a social network from the popular television show *Mad Men*. This network, which was compiled by Chang (2013) and made available in **gcookbook** (Chang, 2012), consists of 52 vertices and 87 edges. Each vertex represents a character on the show, and there is an edge between every two characters who have had a romantic relationship.

²Version 2.1.0, released 1 March 2016. See <https://cran.r-project.org/web/packages/ggplot2/news.html> for the full list of changes in **ggplot2** 2.1.0, as well as the new package vignette, “Extending ggplot2”, which explains how the internal `ggproto` system of object-oriented programming can be used to create new geoms.

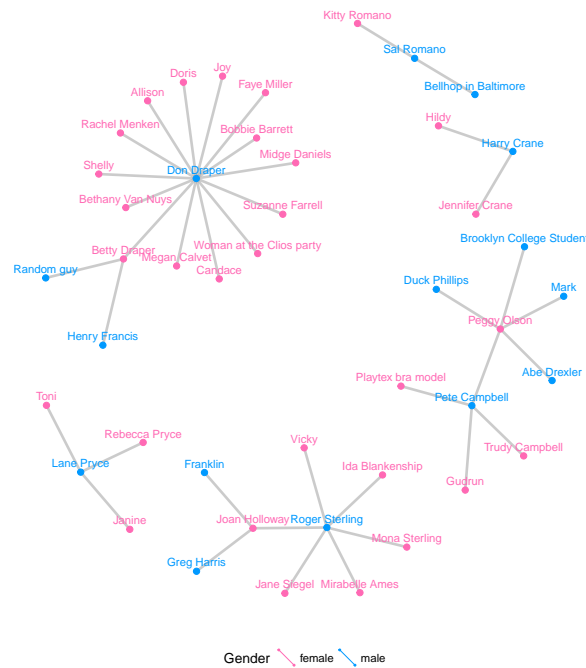


Figure 1: Graph of the characters in the show Mad Men who are linked by a romantic relationship.

Figure 1 is a visualization of this network. In the plot, we can see one central character who has many more relationships than any other character. This vertex represents the main character of the show, Don Draper, who is quite the “ladies’ man.” Networks like this one, no matter how simple or complex, are everywhere, and we hope to provide the curious reader with a straightforward way to visualize any network they choose.

Coloring the vertices or edges in a graph is a quick way to visualize grouping and helps with pattern or cluster detection. The vertices in a network and the edges between them compose the structure of a network, and being able to visually discover patterns among them is a key part of network analysis. Viewing multiple layouts of the same network can also help reveal patterns or clusters that would not be discovered when only viewing one layout or analyzing only its underlying adjacency matrix.

Three implementations of network visualizations

We present two basic approaches to using the **ggplot2** framework for network visualization. First, we implement network visualizations by providing a wrapper function, **ggnet2** for the user to visualize a network using **ggplot2** elements (Schloerke et al., 2016). Second, we implement network visualizations using layering in **ggplot2**. For the second approach, we have two ways of creating a network visualization. The first, **geomnet**, wraps all network structures, including vertices, edges, and vertex labels into a single geom. The second, **ggnetwork**, implements each of these structural components in an independent geom and layers them to create the visualization (Briatte, 2016). In each package, our goal is to provide users with a way to map network properties to aesthetic properties of graphs that is familiar to them and straightforward to implement. Each package has a slightly different approach to accomplish this goal, and we will discuss all of these approaches in this section. For each implementation, we also provide the code necessary to create Figure 1, and describe the arguments used. We conclude the section with a side-by-side comparison of the features available in all three implementations in Table 1.

ggnet2

The **ggnet2** function is a part of the **GGally** package, a suite of functions developed to extend the plotting capabilities of **ggplot2** (Schloerke et al., 2016). A detailed description of the **ggnet2** function is available from within the package as a vignette. Some example code to recreate Figure 1 using **ggnet2** is presented below.

```

library(GGally)
library(network)
# make the data available
data(madmen, package = 'geomnet')
# data step for both ggnet2 and ggnetwork
# create undirected network
mm.net <- network(madmen$edges[, 1:2], directed = FALSE)
mm.net # glance at network object
## Network attributes:
##   vertices = 45
##   directed = FALSE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 39
##   missing edges= 0
##   non-missing edges= 39
##
## Vertex attribute names:
##   vertex.names
##
## No edge attributes
# create node attribute (gender)
rownames(madmen$vertices) <- madmen$vertices$label
mm.net %v% "gender" <- as.character(
  madmen$vertices[ network.vertex.names(mm.net), "Gender"]
)
# gender color palette
mm.col <- c("female" = "#ff69b4", "male" = "#0099ff")
# create plot for ggnet2
set.seed(10052016)
ggnet2(mm.net, color = mm.col[ mm.net %v% "gender" ],
        labelon = TRUE, label.color = mm.col[ mm.net %v% "gender" ],
        size = 2, vjust = -0.6, mode = "kamadakawai", label.size = 3)

```

The `ggnet2` function offers a large range of network visualization functionality in a single function call. Although its result is a **ggplot2** object that can be further styled with **ggplot2** scales and themes, the syntax of the `ggnet2` function is designed to be easily understood by the users, who may not be familiar with **ggplot2** objects. The aesthetics relating to the nodes are controlled by arguments such as `node.alpha` or `node.color`, while those relating to the edges are controlled by arguments starting with 'edge'. Additionally, as seen in the code above, the usual **ggplot2** arguments like `color` can be used without the prefix to map node attributes to aesthetic values. The arguments with the `node.` prefix are aliased versions for readability of the code. Thus, while `ggnet2` applies the grammar of graphics to network objects, the function itself still works very much like the plotting functions of the **igraph** and **network** packages: a long series of arguments is used to control every possible aspect of how the network should be visualized.

The `ggnet2` function takes a single network object as input. This initial object might be an object of class "network" from the **network** package (with the exception of hypergraphs or multiplex graphs), or any data structure that can be coerced to an object of that class via functions in the **network** package, such as an incidence matrix, an adjacency matrix, or an edge list. Additionally, if the **intergraph** package (Bojanowski, 2015) is installed, the function also accepts a network object of class "igraph". Internally, the function converts the network object to two data frames: one for edges and another one for nodes. It then passes them to **ggplot2**. Each of the two data frames contain the information required by **ggplot2** to plot segments and points respectively, such as a shape for the points (nodes) and a line type for the segments (edges). The final result returned to the user is a plot with a minimum of two layers, or more if there are edge and/or node labels.

The `mode` argument of `ggnet2` controls how the nodes of the network are to be positioned in the plot returned by the function. This argument can take any of the layout values supported by the `gplot.layout` function of the **sna** package, and defaults to 'fruchtermanreingold', which places the nodes through the force-directed layout algorithm by Fruchterman and Reingold (1991). In the example presented above, the Kamada-Kawai layout is used by adding `'mode = "kamadakawai"'` to the

function call. Many other possible layouts and their parameters can also be passed to `ggnet2` through the `layout.par` argument. For a list of possible layouts and their arguments, see `?sna::gplot.layout`.

Other arguments passed to the `ggnet2` function offer extensive control over the aesthetics of the plot that it returns, including the addition of edge and/or node labels and their respective aesthetics. Arguments such as `node.shape` or `edge.lty`, which control the shape of the nodes and the line type of the edges, respectively, can take a single global value, a vector of global values, or the name of an edge or vertex attribute to be used as an aesthetic mapping. This feature is used to change the size of the nodes and the node labels by including `'size = 2'` and `'label.size = 3'` in the function call.

This last functionality builds on one of the strengths of the "network" class, which can store information on network edges and nodes as attributes that are then accessible to the user through the `%e%` and `%v%` operators respectively.³ Usage examples of these operators can be seen above. The attribute of `gender` is assigned to nodes, which in turn is accessed to color the nodes and node labels by gender. If the `ggnet2` function is given the `node.alpha = "importance"` argument, it will interpret it as an attempt to map the vertex attribute called 'importance' to the transparency level of the nodes. This works exactly like the command `net %v% "importance"`, which returns the vertex attribute 'importance' of the "network" object `net`. This functionality allows the `ggnet2` function to work in a similar fashion to `ggplot2` mappings of aesthetics within the `aes` operator.

The `ggnet2` function also provides a few network-specific options, such as sizing the nodes as a function of their unweighted degree, or using the primary and secondary modes of a bipartite network as an aesthetic mapping for the nodes.

All in all, the `ggnet2` function combines two different kinds of processes: it translates a network object into a data frame suitable for plotting with `ggplot2`, and it applies network-related aesthetic operations to that data frame, such as coloring the edges in function of the color of the nodes that they connect.

geomnet

```
# also loads ggplot2
library(geomnet)

# data step: join the edge and node data with a fortify call
MMnet <- fortify(as.edgedf(madmen$edges), madmen$vertices)
# create plot
set.seed(10052016)
ggplot(data = MMnet, aes(from_id = from_id, to_id = to_id)) +
  geom_net(aes(colour = Gender), layout.alg = "kamadakawai",
           size = 2, labelon = TRUE, vjust = -0.6, ecolour = "grey60",
           directed = FALSE, fontsize = 3, ealpha = 0.5) +
  scale_colour_manual(values = c("#FF69B4", "#0099ff")) +
  xlim(c(-0.05, 1.05)) +
  theme_net() +
  theme(legend.position = "bottom")
```

Data structure

The package `geomnet` implements network visualization in a single `ggplot2` layer. A stable version is available on CRAN, with a development version available at <https://github.com/sctyner/geomnet>. The package has two main functions: `stat_net`, which performs all of the calculations, and `geom_net`, which renders the plot. It also contains the secondary functions `geom_circle` and `theme_net`, which assist, respectively, in drawing self-referencing edges and removing axes and other background elements from the plots. The approach in `geomnet` is similar to the implementation of other, native `ggplot2` geoms, such as `geom_smooth`. When using `geom_smooth`, the user does not need to know about any of the internals of the `loess` function, and similarly, when using `geomnet`, the user is not expected to know about the internals of the layout algorithm, just the name of the algorithm they'd like to use. On the other hand, if users are comfortable with network analysis, the entire body of layout methods provided by the `sna` package is available to them through the parameters `layout.alg` and `layout.par`.

In network analysis there are usually two sources of information: one data set consisting of a description of the nodes, represented as the vertices in the network and vertex attributes, and another data set detailing the relationship between these nodes, i.e. it consists of the edge list and any additional edge attributes. The minimum amount of information needed is a vector of all vertex labels and

³See Butts et al. (2014, p. 22-24). The equivalent operators in the `igraph` package are called `E` and `V`.

a two column data frame that encodes the edge list of the network. In order for this geometry to work, these two data sets need to be combined into a single data frame. For this, we implemented several new fortify methods for producing the correct data structure from different S3 objects that encode network information. Supported classes are "network" from the **sna** and **network** packages, "igraph" from the **igraph** package, "adjmat", and "edgedf". The last two are new classes introduced in **geomnet** that are identical to the "matrix" and "data.frame" classes, respectively. We created these new classes and the functions `as.adjmat()` and `as.edgedf()` so that network data in adjacency matrix and edgelist (data frame) formats can have their own fortify functions, separate from the very generic "matrix" and "data.frame" classes. These fortify functions combine the edge and the node information using a full join. A full join is used because generally, there will be some vertices that are sinks in the network because they only show up in the 'to' column, and so we accommodate for these by adding artificial edges in the data set that have missing information for the 'to' column. The user may also pass two data frames to the function, e.g. 'data = edge_data' and 'vertices = vertex_data', but we recommend using the fortify methods whenever possible.

A usage example of the `fortify.edgedf` method is presented in the code above with the creation of the `MMnet` data set. Two dataframes, `madmen$edges` and `madmen$vertices` are joined to create the required data. The first few rows of these data sets and their merged result are below.

```
head(as.edgedf(madmen$edges), 3)
##      from_id      to_id
## 1 Betty Draper Henry Francis
## 2 Betty Draper   Random guy
## 3  Don Draper    Allison
head(madmen$vertices, 3)
##      label Gender
## Betty Draper Betty Draper female
## Don Draper   Don Draper   male
## Harry Crane  Harry Crane  male
head(fortify(as.edgedf(madmen$edges), madmen$vertices), 3)
##      from_id      to_id Gender
## 1 Betty Draper Henry Francis female
## 2 Betty Draper   Random guy female
## 3  Don Draper    Allison   male
```

The formal requirements of `stat_net` are two columns, called `from_id` and `to_id`. During this routine, columns `x`, `y` and `xend`, `yend` are calculated and used as a required input for `geom_net`.

Other variables may also be included for each edge, such as the edge weight, in-degree, out-degree or grouping variable.

Parameters and aesthetics

Parameters that are currently implemented in `geom_net` are:

- **layout:** the `layout.alg` parameter takes a character value corresponding to the possible network layouts in the **sna** package that are available within the `gplot.layout.*()` family of functions. The default layout algorithm used is the Kamada-Kawai layout, a force-directed layout for undirected networks (Kamada and Kawai, 1989).
In **sna**, for each layout there is a corresponding set of possible layout parameters, `layout.par`, which can be passed as a list to `geom_net`. If the user wishes to create small multiples using **ggplot2** facets, they can use `fiteach`, a logical value specifying whether the same layout should be used for all panels (default) or each panel's data should be fit separately. Finally, the `singletons` parameter is a logical value that dictates whether or not to include nodes with zero indegree and zero outdegree in the visualization. The default is set to `TRUE`, and if set to `FALSE` nodes will only appear in panels where they have indegree or outdegree of at least one.
- **vertices:** any of **ggplot2**'s aesthetics relating to points: `colour`, `size`, `shape`, `alpha`, `x`, and `y` are available and used for specifying the appearance of nodes in the network. For example `'aes(colour = Gender)'` is used above to color the nodes and node labels according to the gender of each character.
- **edges:** for edges we distinguish between two different sets of aesthetics: aesthetics that only relate to line attributes, such as `linewidth` and `linetype`, and aesthetics that are also used by the point `geom`. The former can be used in the same way as they are used in `geom_segment`, while

the latter, like `alpha` or `colour`, for instance, are used for vertices unless separately specified. Instead, use the parameters `ecolour` or `ealpha`, which are only applied to the edges. If the group variable is specified, a new variable, called `samegroup` is added during the layout process. This variable is `TRUE`, if an edge is between two vertices of the same group, and `FALSE` otherwise. If `samegroup` is `TRUE`, the corresponding edge will be colored using the same color as the vertices it connects. If the edge is between vertices of a different group, the default grey shade is used for the edge.

The parameter `curvature` is set to zero by default, but if specified, leads to curved edges using the newly implemented **ggplot2** geom `geom_curve` instead of the regular `geom_segment`. Note that the edge specific aesthetics that overwrite node aesthetics are currently considered as 'as.is' values: they do not get a legend and are not scaled within the **ggplot2** framework. This is done to avoid any clashes between node and edge scales.

self-referencing vertices: some networks contain self references, i.e. an edge has the same vertex id in its `from` and `to` columns. If the parameter `selfloops` is set to `TRUE`, a circle is drawn using the new `geom_circle` next to the vertex to represent this self reference.

- **arrow:** whenever the parameter `directed` is set from its default state to `TRUE`, arrows are drawn from the 'from' to the 'to' node, with tips pointing towards the 'to' node. By default, arrows have an absolute size of 10 points. The entire structure of the arrow can be changed by passing an arrow object from the **grid** package to the `arrow` argument. If the user doesn't wish to change the whole arrow object, the parameters `arrowsize` and `arrowgap` are also available. The `arrowsize` argument is of a positive numeric value that is used as a multiple of the original arrow size, i.e. `arrowsize = 2` shows arrow tips at twice their original size. The parameter `arrowgap` can be used to avoid overplotting of the arrow tips by the nodes, `arrowgap` specifies a proportion by which the edge should be shrunk with default of 0.05. A value of 0.5 will result in edges drawn only half way from the 'from' node to the 'to' node.
- **labels:** the `labelon` argument is a logical parameter, which when set to `TRUE` will label the nodes with their IDs, as is in Figure 1. The aes option `label` can also be used to label nodes, in which case the nodes are labeled with the value corresponding to their respective values of the provided variable. If `colour` is specified for the nodes, the same values are used for the labels, unless `labelcolour` is specified. If `fontsize` is specified, it changes the label size to that value in points. Other parameter values, such as `vjust` and `hjust` help in adjusting labels relative to the nodes. The parameters work in the same fashion as in native **ggplot2** geoms. Additionally, the label can be drawn by using `geom_text` (the default) or using the new `geom_label` in **ggplot2** by adding `'labelgeom = "label"'` to the arguments in `geom_net`. Finally, with the help of the package **ggrepel** by [Slowikowski \(2016\)](#) we have implemented the logical `repel` argument, which when true, uses `geom_text_repel` or `geom_label_repel` to plot the labels instead of `geom_text` or `geom_label`, respectively. Using `repel` can be extremely useful when the networks are dense or the labels are long, as in Figure 1, helping to solve a common problem with many network visualizations.

ggnetwork

ggnetwork is a small R package that mimics the behavior of `geomnet` by defining several geoms to achieve similar results.

```
# create plot for ggnetwork. uses same data created for ggnetwork2 function
library(ggnetwork)
set.seed(10052016)
ggplot(data = ggnetwork(mm.net, layout = "kamadakawai"),
       aes(x, y, xend = xend, yend = yend)) +
  geom_edges(color = "grey50") + # draw edge layer
  geom_nodes(aes(colour = gender), size = 2) + # draw node layer
  geom_nodetext(aes(colour = gender, label = vertex.names),
               size = 3, vjust = -0.6) + # draw node label layer
  scale_colour_manual(values = mm.col) +
  xlim(c(-0.05, 1.05)) +
  theme_blank() +
  theme(legend.position = "bottom")
```

The approach taken by the **ggnetwork** package is to alias some of the native geoms of the **ggplot2** package. An aliased geom is simply a variant of an already existing one. The **ggplot2** package contains

several examples of aliased geoms, such as `geom_histogram`, which is a variant of `geom_bar` see (see Wickham, 2016, p. 67, Table 4.6).

Following that logic, the **ggnetwork** package adds four aliased geometries to **ggplot2**:

- `geom_nodes`, an alias to `geom_point`;
- `geom_edges`, an alias to either `geom_segment` or `geom_curve`;
- `geom_nodetext`, an alias to `geom_text`; and
- `geom_edgetext`, an alias to `geom_label`.

The four geoms are used to plot nodes, edges, node labels and edge labels, respectively. Two of the geoms that they alias, `geom_curve` and `geom_label`, are part of the new geometries introduced in **ggplot2** version 2.1.0. All four geoms behave exactly like those that they alias, and take exactly the same arguments. The only exception to that rule is the special case of `geom_edges`, which accepts both the arguments of `geom_segment` and those of `geom_curve`; if its `curvature` argument is set to anything but 0 (the default), then `geom_edges` behaves exactly like `geom_curve`; otherwise, it behaves exactly like `geom_segment`. Three of the four available aliased geoms are used above to create the visualization of the Mad Men relationship network.

Just like the `ggnet2` function, the **ggnetwork** package takes a single network object as input. This can be an object of class "network", some data structure coercible to that class, or an object of class "igraph" when the **intergraph** package is installed. This object is passed to the 'workhorse' function of the package, which is also called `ggnetwork` to create a data frame, and then to the data argument of `ggplot()`.

Internally, the `ggnetwork` function starts by computing the x and y coordinates of all nodes in the network with respect to its layout argument, which defaults to the Fruchterman-Reingold layout algorithm (Fruchterman and Reingold, 1991). It then extracts the edge list of the network, to which it adds the coordinates of the sender and receiver nodes as well as all edge-level attributes. The result is a data frame with as many rows as there are edges in the network, and where the x, y, xend and yend hold the coordinates of the network edges.

At that stage, the `ggnetwork` function, like the **geomnet** package, performs a left-join of that augmented edge list with the vertex-level attributes of the 'from' nodes. It also adds one self-loop per node, in order to ensure that every node is plotted even when their degree is zero—that is, even if the node is not connected to any other node of the network, and is therefore absent from the edge list. The data frame created by this process contains one row per edge as well as one additional row per node, and features all edge-level and vertex-level attributes initially present in the network.⁴

The `ggnetwork` function also accepts the arguments `arrow.gap` and `by`. Like in **geomnet**, `arrow.gap` slightly shortens the edges of directed networks in order to avoid overplotting edge arrows and nodes. The argument `by` is intended for use with plot facets. Passing an edge attribute as a grouping variable to the `by` argument will cause `ggnetwork` to return a data frame in which each node appears as many times as there are unique values of that edge attribute, using the same coordinates for all occurrences. When that same edge attribute is also passed to either `facet_wrap` or `facet_grid`, each edge of the network will show in only one panel of the plot, and all nodes will appear in each of the panels at the same position. This makes the panels of the plot comparable to each other, and allows the user to visualize the network structure as a function of a specific edge attribute, like a temporal attribute.

Examples

In this section, we demonstrate some of the current capabilities of `ggnet2`, **geomnet**, and **ggnetwork** in a series of side by side examples. While the output is nearly identical for each method of network visualization, the code and implementations differ across the three methods. For each of these examples, we present the code necessary to produce the network visualization in each of the three packages, and discuss each application in detail.

For the following examples we will be loading all three packages under comparison. In practice, only one of these packages would be needed to visualize a network in the **ggplot2** framework:

```
library(ggplot2)
library(GGally)
```

⁴One limitation of this process is that it requires some reserved variable names (x, y, xend and yend), which should not also be present as edge-level or vertex-level attributes (otherwise the function will simply break). Similarly, if an edge attribute and a vertex attribute have the same name, like 'na', which the **network** package defines as an attribute for both edges and vertices in order to flag missing data, `ggnetwork` will rename them to 'na.x' (for the edge-level attribute) and 'na.y' (for the vertex-level attribute).

Functionality	ggnet2 (GGally)	geom_net (geomnet)	geom_nodes, geom_edges, etc (ggnetwork)
Data	object of class "network" or object easily converted to that class (i.e. incidence or adjacency matrices, edge list) or object of class "igraph"	a fortified "network", "igraph", "edgdf", or "adjmat" object OR one edge data frame and one node data frame to be merged internally	same as ggnet2
Naming conventions	node._, edge._, label._, edge.label._ for alpha, color, etc.	arguments identical to ggplot2 with exception of ecolor, ealpha	same as ggplot2
Layout package & default	sna , Fruchterman-Reingold	sna , Kamada-Kawai	sna , Fruchterman-Reingold
Aesthetic mappings to variables	all alpha, color, shape, size for nodes, edges, labels	colour, size, shape, x, y, linetype, linewidth, label, group, fontsize	same as ggplot2
Arrows	directed = TRUE, arrow.size, gap	arrowsize, gap, arrow = arrow() like ggplot2	specify arrows in geom_edge like in code-geom_segment, arrow.gap
Theme or palette changes	done in the function with arguments like <code>_.legend</code> , <code>_.palette</code> , etc. and adding ggplot2 elements	adding ggplot2 elements	adding ggplot2 elements
Creating small multiples	created separately, use <code>grid.arrange</code> from gridExtra	add group argument to <code>fortify()</code> and use <code>facet_*</code> () from ggplot2	use by argument in <code>ggnetwork()</code> and <code>facet_*</code> () from ggplot2
Edge labelling?	Yes	No	Yes
Draw self-loops?	No	Yes	No

Table 1: Comparing the three different package side-by-side.

```
library(geomnet)
library(ggnetwork)
```

Blood donation

We begin with a very simple example that most should be familiar with: blood donation. In this directed network, there are eight vertices and 27 edges. The vertices represent the eight different blood types in humans that are most important for donation: the ABO blood types A, B, AB, and O, combined with the RhD positive (+) and negative (-) types. The edges are directed: a person whose blood type is that of a *from* vertex can donate blood to a person whose blood type is that of a corresponding *to* vertex. This network is shown in Figure 2. The code to produce each one of the networks is shown above Figure 2. We take advantage of each approach's ability to assign identity values to the aesthetic values. The color is changed to a dark red, the size of the nodes is changed to be large enough to accommodate the blood type label, which we also change the color of, and we use the directed and arrow arguments of each implementation to show the precise blood donation relationships. Additionally, we change the node layout to circle, and the placement of the labels with the `hjust` and `vjust` options.

```
# make data accessible
data(blood, package = "geomnet")

# plot with ggnet2 (Figure 5a)
set.seed(12252016)
ggnet2(network(blood$edges[, 1:2]), directed=TRUE,
        mode = "circle", size = 15, label = TRUE,
        arrow.size = 10, arrow.gap = 0.05, vjust = 0.5,
        node.color = "darkred", label.color = "grey80")

head(blood$edges,3) # glance at the data
## from to group_to
## 1 AB- AB+ same
## 2 AB- AB- same
## 3 AB+ AB+ same
# plot with geomnet (Figure 5b)
set.seed(12252016)
ggplot(data = blood$edges, aes(from_id = from, to_id = to)) +
  geom_net(colour = "darkred", layout.alg = "circle", labelon = TRUE, size = 15,
           directed = TRUE, vjust = 0.5, labelcolour = "grey80",
           arrowsize = 1.5, linewidth = 0.5, arrowgap = 0.05,
           selfloops = TRUE, ecolour = "grey40") +
  theme_net()

# plot with ggnetwork (Figure 5c)
set.seed(12252016)
ggplot(ggnetwork(network(blood$edges[, 1:2]),
                    layout = "circle", arrow.gap = 0.05),
        aes(x, y, xend = xend, yend = yend)) +
  geom_edges(color = "grey50",
             arrow = arrow(length = unit(10, "pt"), type = "closed")) +
  geom_nodes(size = 15, color = "darkred") +
  geom_nodetext(aes(label = vertex.names), color = "grey80") +
  theme_blank()
```

In this example every vertex has a self-reference, as blood between two people of matching ABO and RhD type can always be exchanged. The **geomnet** approach shows these self-references as circles looping back to the vertex, which is controlled by using the parameter setting `selfloops = TRUE`.

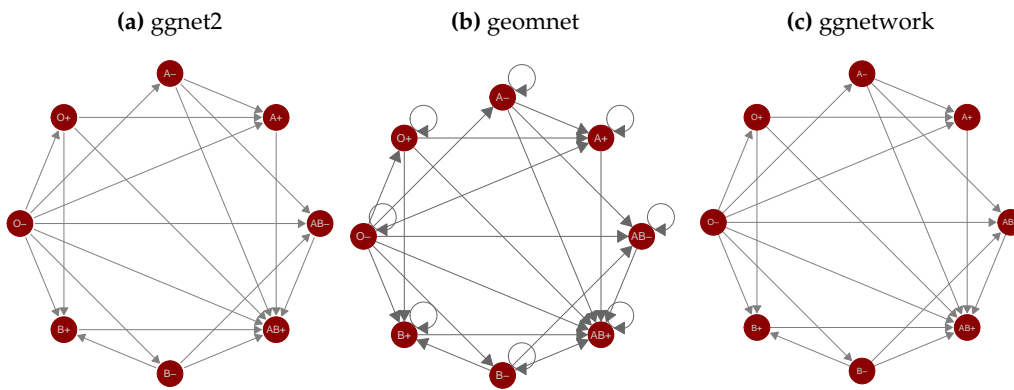


Figure 2: Network of blood donation possibilities in humans by ABO and RhD blood types.

colour and size aesthetics in Figure 2 are set to identity values to change the size and color of all vertices. We have also used the `layout` and `label` arguments to change the default Kamada-Kawai layout to a circle layout and to print labels for each of the blood types. The circle layout places blood types of the same ABO type next to each other and spreads the vertices out far enough to distinguish between the various “in” and “out” types. We can tell clearly from this plot that the O-type is the universal donor: it has an out-degree of seven and an in-degree of zero. Additionally, we can see that the AB+ type is the universal recipient, with an in-degree of seven and an out-degree of zero. Anyone looking at this plot can quickly determine which type(s) of blood they can receive and which type(s) can receive their blood.

Email network

The email network comes from the 2014 VAST Challenge (Cook et al., 2014). It is a directed network of emails between company employees with 55 vertices and 9,063 edges. Each vertex represents an employee of the company, and each edge represents an email sent from one employee to another. The arrow of the directed edge points to the recipient of the email. If an email has multiple recipients, multiple edges, one for each recipient, are included in the network. The network contains two business weeks of emails across the entire company. In order to better visualize the structure of the communication network between employees, emails that were sent out to all employees are removed. A glimpse of the data objects used is below.

```
em.net # ggnet2 and ggnetwork
## Network attributes:
## vertices = 55
## directed = TRUE
## hyper = FALSE
## loops = FALSE
## multiple = FALSE
## bipartite = FALSE
## total edges= 4743
## missing edges= 0
## non-missing edges= 4743
##
## Vertex attribute names:
## curr_empl_type vertex.names
##
## Edge attribute names not shown
emailnet[1,c(1:2,7,21)] # geomnet
## from_id
## 1 Ada.Campo-Corrente@gastech.com.kronos
## to_id day
## 1 Ingrid.Barranco@gastech.com.kronos 10
## CurrentEmploymentType
## 1 Executive
```

Emails taken by themselves form an event network, i.e. edges do not have any temporal duration.

(a) ggnet2

```
# make data accessible
data(email, package = 'geomnet')

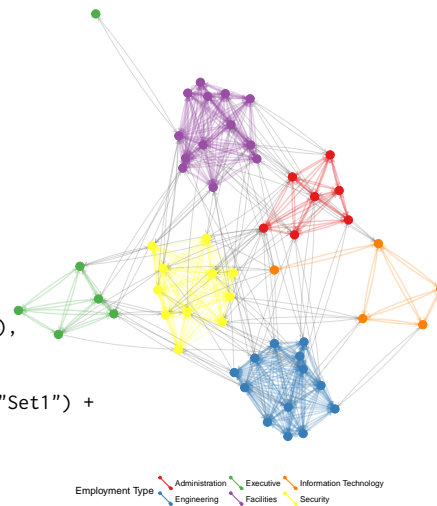
# create node attribute data
em.cet <- as.character(
  email$nodes$CurrentEmploymentType)
names(em.cet) = email$nodes$label

# remove the emails sent to all employees
edges <- subset(email$edges, nrecipients < 54)
# create network
em.net <- edges[, c("From", "to")]
em.net <- network(em.net, directed = TRUE)
# create employee type node attribute
em.net %v% "curr_empl_type" <-
  em.cet[ network.vertex.names(em.net) ]
set.seed(10312016)
ggnet2(em.net, color = "curr_empl_type",
  size = 4, palette = "Set1", arrow.gap = 0.
  arrow.size = 5, edge.alpha = 0.25,
  mode = "fruchtermanreingold",
  edge.color = c("color", "grey50"),
  color.legend = "Employment Type") +
  theme(legend.position = "bottom")
```



(b) geomnet

```
# data step for the geomnet plot
email$edges <- email$edges[, c(1,5,2:4,6:9)]
emailnet <- fortify(
  as.edgdf(subset(email$edges, nrecipients < 54),
    email$nodes)
)
set.seed(10312016)
ggplot(data = emailnet,
  aes(from_id = from_id, to_id = to_id)) +
  geom_net(layout.alg = "fruchtermanreingold",
  aes(colour = CurrentEmploymentType,
  group = CurrentEmploymentType,
  linewidth = 3 * (...samegroup.. / 8 + .125)),
  ealpha = 0.25, size = 4, curvature = 0.05,
  directed = TRUE, arrowsize = 0.5) +
  scale_colour_brewer("Employment Type", palette = "Set1") +
  theme_net() +
  theme(legend.position = "bottom")
```



(c) ggnetwork

```
# use em.net created in ggnet2step
set.seed(10312016)
ggplot(ggnetwork(em.net, arrow.gap = 0.02,
  layout = "fruchtermanreingold"),
  aes(x, y, xend = xend, yend = yend)) +
  geom_edges(
  aes(color = curr_empl_type),
  alpha = 0.25,
  arrow = arrow(length = unit(5, "pt"),
  type = "closed"),
  curvature = 0.05) +
  geom_nodes(aes(color = curr_empl_type),
  size = 4) +
  scale_color_brewer("Employment Type",
  palette = "Set1") +
  theme_blank() +
  theme(legend.position = "bottom")
```

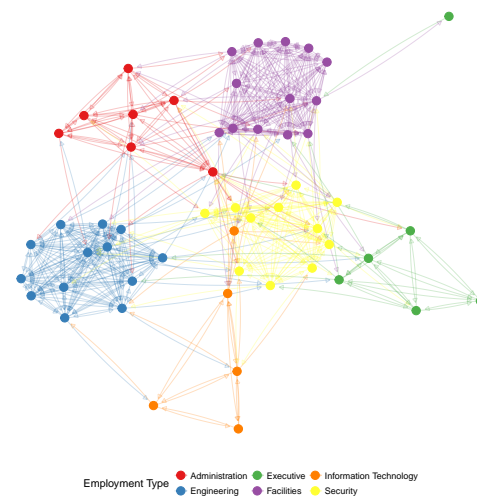


Figure 3: Email network within a company over a two week period.

Here, however, we can think of emails as observable expressions of the underlying, unobservable, relationship between employees. We can think of this network as a dynamic temporal network, i.e. this network has the potential to change over time. The `ndtv` package by Bender-deMoll (2016) allows the analysis of such networks and provides impressive animations of the underlying dynamics. Here, we are using two static approaches to visualize the network: first, we aggregate emails across the whole time frame (shown in Figure 3), then we aggregate emails by day and use small multiples to allow a comparison of day-to-day behavior (shown in Figure 4).

For all of the email examples, we have colored the vertices by the variable `CurrentEmploymentType`, which contains the department in the company of which each employee is a part of. There are six distinct clusters in this network which almost perfectly correspond to the six different types of employees in this company: administration, engineering, executive, facilities, information technology, and security. Other features in the code include using `alpha` arguments to change the transparency of the edges, curvature arguments to show mutual communication as two edges instead of one edge with two arrowheads, and the addition of `ggplot2` functions like `scale_colour_brewer` and `theme` to customize the colors of the nodes and their corresponding legend.

In Figure 3 we can clearly see the varying densities of communications within departments and the more sparse communication between employees in different departments. We also see that one of the executives only communicates with employees in Facilities, while one of the IT employees frequently communicates with security employees.

A comparison of the results of `ggnet`, `geomnet` and `ggnetwork` reveals some of the more subtle differences between the implementations:

- In the `ggnet2` implementation, the opacity of the edges between employees in the same cluster is higher than it is for the edges between employees in different clusters. This is due to the fact that the email network does not make use of edge weights: instead, every email between two employees is represented by an edge, resulting in edge overplotting. The `edge.alpha` argument has been set to a value smaller than one, therefore multiple emails between two employees create more opaque edges between them. Multiple emails are also taken into account in the `geomnet` package. When there is more than one edge connecting two vertices, the `stat_net` function adds a weight variable to the edge list, which is passed automatically to the layout algorithms and taken into account during layout. This is thanks to the `sna` package, which supports the use of weights in its edge list. In addition to taking weights into account in the layout, we can also make use of them in the visualization. `geomnet` allows to access all of the internal variables created in the visualization process, such as coordinates `..x..`, `..y..` and edge weights `..weight..`. Note the use of the `ggplot2` notation `..` for internal variables.
- In the first two layouts of Figure 3, edges between employees who share the same employment type are given the color of that employment type, while edges between employees belonging to different types are plotted in grey. This feature is particularly useful to visualize the amount of within-group connectedness in a network. By contrast, in the last layout, edges are colored according to the sender's employment type, because the `ggnetwork` package does not support coloring edges as a function of node-level attributes.
- Finally, in the last two layouts of Figure 3, the curvature argument has been set to 0.05, resulting in slightly curved edges in both plots. This feature, which takes advantage of the `geom_curve` geometry released in `ggplot2` 2.1.0, makes it possible to visualize which edges correspond to reciprocal connections; in an email communication network, as one might expect, most edges fall into that category.

To give some insight into how the relations between employees change over time, we facet the network by day: each panel in Figure 4 shows email networks associated with each day of the work week. The code for these visualizations is below. The different approaches create small multiples in different ways. The `ggnet2` approach requires that the network be separated, each plot created individually, then placed together using the `grid.arrange` function from the `gridExtra` package (Auguie, 2016). The `geomnet` approach uses the `facet_*` family of functions just as they are used in `ggplot2`, and the `ggnetwork` approach uses the `by` argument in the `ggnetwork` function in combination with the `facet_*` functions. We present the full code for each of these approaches below.

First, the code for the `ggnet2` approach, which results in Figure 4(a):

```
# data preparation. first, remove emails sent to all employees
em.day <- subset(email$edges, nrecipients < 54)[, c("From", "to", "day")]
# for small multiples by day, create one element in a list per day
# (10 days, 10 elements in the list em.day)
em.day <- lapply(unique(em.day$day),
  function(x) subset(em.day, day == x)[, 1:2 ])
```

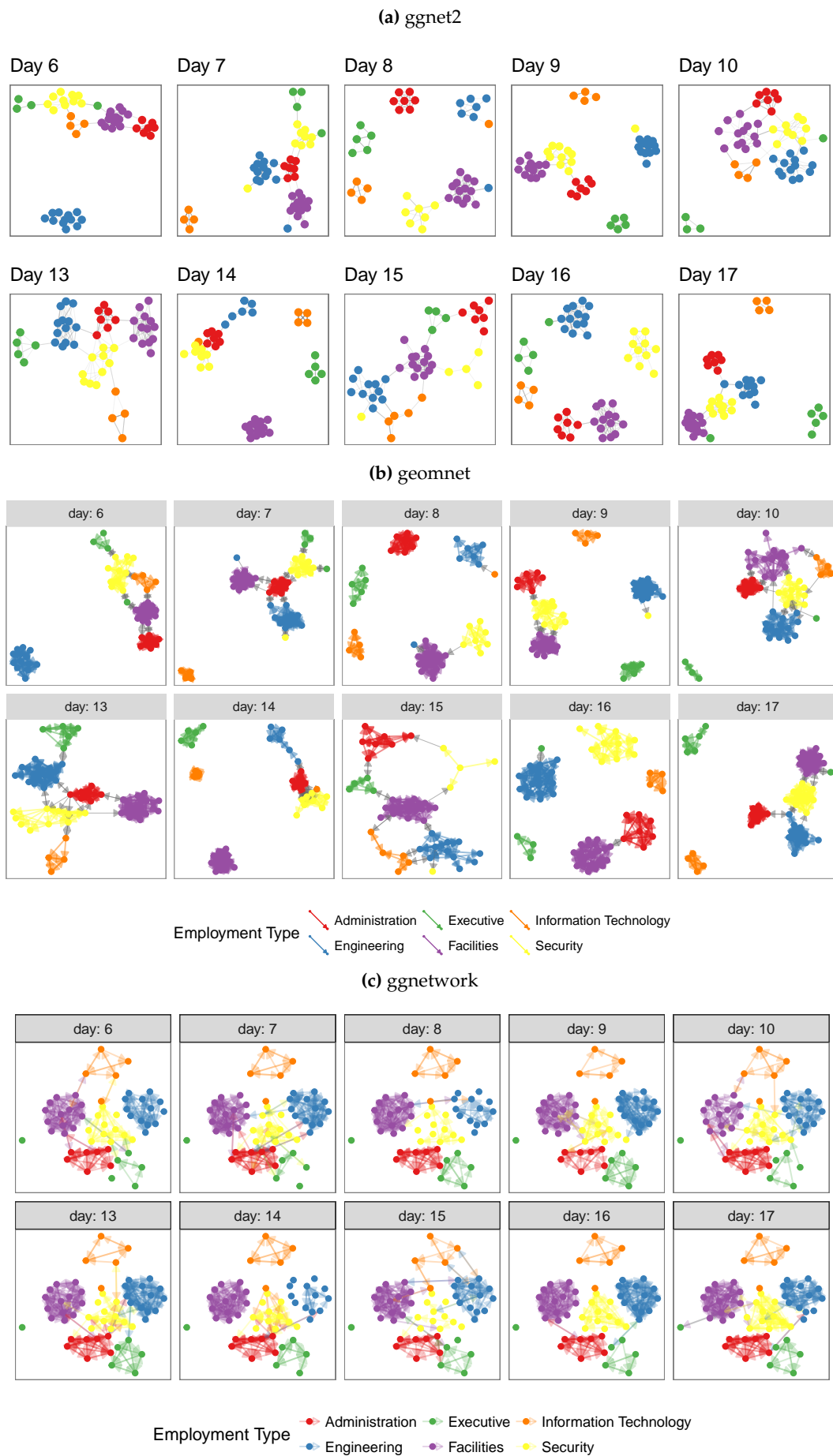


Figure 4: The same email network as in Figure 3 faceted by day of the week.


```

# make the list of edgelist a list of network objects for plotting with ggnet2
em.day <- lapply(em.day, network, directed = TRUE)
# create vertex (employee type) and network (day) attributes for each element in list
for (i in 1:length(em.day)) {
  em.day[[ i ]] %v% "curr_empl_type" <-
    em.cet[ network.vertex.names(em.day[[ i ]]) ]
  em.day[[ i ]] %n% "day" <- unique(email$edges$day)[ i ]
}

# plot ggnet2
# first, make an empty list containing slots for the 10 days (one plot per day)
g <- list(length(em.day))
set.seed(7042016)
# create a ggnet2 plot for each element in the list of networks
for (i in 1:length(em.day)) {
  g[[ i ]] <- ggnet2(em.day[[ i ]], size = 2,
                    color = "curr_empl_type",
                    palette = "Set1", arrow.size = 0,
                    arrow.gap = 0.01, edge.alpha = 0.1,
                    legend.position = "none",
                    mode = "kamadakawai") +
    ggtitle(paste("Day", em.day[[ i ]] %n% "day")) +
    theme(panel.border = element_rect(color = "grey50", fill = NA),
          aspect.ratio = 1)
}
# arrange all of the network plots into one plot window
gridExtra::grid.arrange(grobs = g, nrow = 2)

```

Second, the code for the **geomnet** approach, which results in Figure 4(b):

```

# data step: use the fortify.edgedf group argument to
# combine the edge and node data and allow all nodes to
# show up on all days. Also, remove emails sent to all
# employees
emailnet <- fortify(as.edgedf(subset(email$edges, nrecipients < 54)), email$nodes, group = "day")

# creating the plot
set.seed(7042016)
ggplot(data = emailnet, aes(from_id = from, to_id = to_id)) +
  geom_net(layout.alg = "kamadakawai", singletons = FALSE,
          aes(colour = CurrentEmploymentType,
              group = CurrentEmploymentType,
              linewidth = 2 * (...samegroup.. / 8 + .125)),
          arrowsize = .5,
          directed = TRUE, fiteach = TRUE, ealpha = 0.5, size = 1.5, na.rm = FALSE) +
  scale_colour_brewer("Employment Type", palette = "Set1") +
  theme_net() +
  facet_wrap(~day, nrow = 2, labeller = "label_both") +
  theme(legend.position = "bottom",
        panel.border = element_rect(fill = NA, colour = "grey60"),
        plot.margin = unit(c(0, 0, 0, 0), "mm"))

```

Finally, the code for the **ggnetwork** approach, which results in Figure 4(c):

```

# create the network and aesthetics
# first, remove emails sent to all employees
edges <- subset(email$edges, nrecipients < 54)
edges <- edges[, c("From", "to", "day")]
# Create network class object for plotting with ggnetwork
em.net <- network(edges[, 1:2])
# assign edge attributes (day)
set.edge.attribute(em.net, "day", edges[, 3])

```

```

# assign vertex attributes (employee type)
em.net %v% "curr_empl_type" <- em.cet[ network.vertex.names(em.net) ]

# create the plot
set.seed(7042016)
ggplot(ggnetwork(em.net, arrow.gap = 0.02, by = "day",
  layout = "kamadakawai"),
  aes(x, y, xend = xend, yend = yend)) +
  geom_edges(
    aes(color = curr_empl_type),
    alpha = 0.25,
    arrow = arrow(length = unit(5, "pt"), type = "closed")) +
  geom_nodes(aes(color = curr_empl_type), size = 1.5) +
  scale_color_brewer("Employment Type", palette = "Set1") +
  facet_wrap(~day, nrow = 2, labeller = "label_both") +
  theme_facet(legend.position = "bottom")

```

Note the two key differences in the visualizations of Figure 4: whether singletons (isolated nodes) are plotted (as in the **ggnetwork** method), and whether *one* layout is used across all panels (as for the **ggnetwork** example) or whether individual layouts are fit to each of the subsets (as for the **ggnet2** and the **geomnet** examples). Plotting isolated nodes in **geomnet** is possible by setting `singletons = TRUE`, and it would be possible in **ggnet2** by including all nodes in the creation of the list of networks. Using the same layout for plotting small multiples in **geomnet** is controlled by the argument `fiteach`. By default, `fiteach = TRUE`, but `fiteach = FALSE` results in all panels sharing the same layout. Having the same layout in each panel makes seeing specific differences in ties between nodes easier, while having a different layout in each panel emphasizes the overall structural differences between the sub-networks. It would be interesting to be able to have a hybrid of these two approaches, but at the moment this is beyond the capability of any of the methods. Through the faceting it becomes obvious that there are several days where one or more of the departments does not communicate with any of the other departments. There are only two days, day 13 and day 15, without any isolated department communications. Faceting is one of the major benefits of implementing tools for network visualization in **ggplot2**. Faceting allows the user to quickly separate dense networks into smaller sub-networks for easy visual comparison and analyses, a feature that the other network visualization tools do not have.

ggplot2 theme elements

This example comes from the `theme()` help page in the **ggplot2** documentation (Wickham, 2016). It is a directed network which shows the structure of the inheritance of theme options in the construction of a **ggplot2** plot. There are 53 vertices and 36 edges in this network. Each vertex represents one possible theme option. There is an arrow from one theme option to another if the element represented by the 'to' vertex inherits its values from the 'from' vertex. For example, the `axis.ticks.x` option inherits its value from the `axis.ticks` value, which in turn inherits its value from the `line` option. Thus, setting the `line` option to a value such as `element_blank()` sets the entire inheritance tree to `element_blank()`, and no lines appear anywhere on the plot background.

Code and plots of the inheritance structure are shown in Figure 5. A glimpse of the data is below.

```

te.net
## Network attributes:
##   vertices = 53
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 48
##   missing edges= 0
##   non-missing edges= 48
##
## Vertex attribute names:
##   size vertex.names
##
## No edge attributes

```

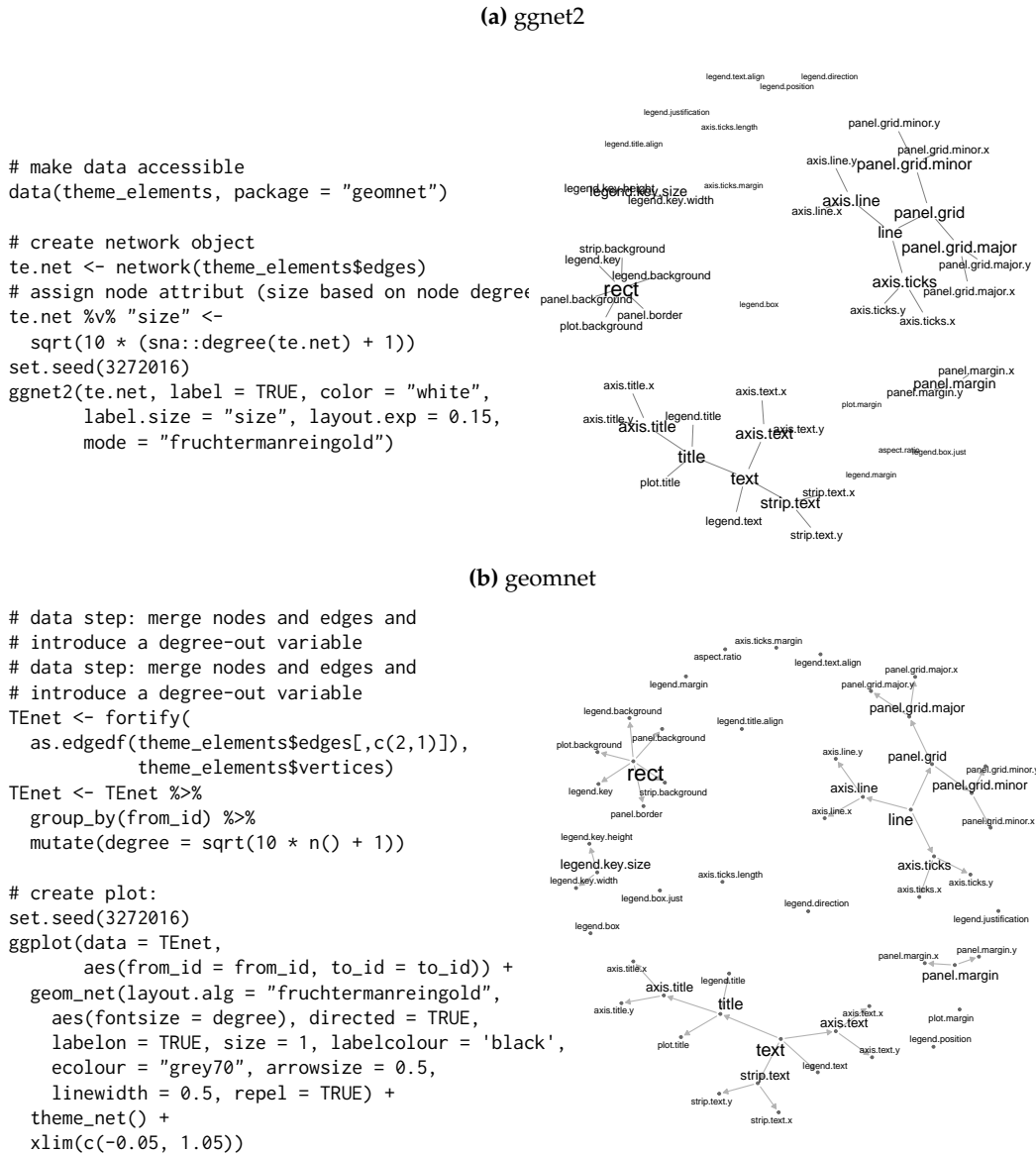


Figure 5: Inheritance structure of ggplot2 theme elements. This is a recreation of the graph found at <http://docs.ggplot2.org/current/theme.html>.

(c) ggnetwork

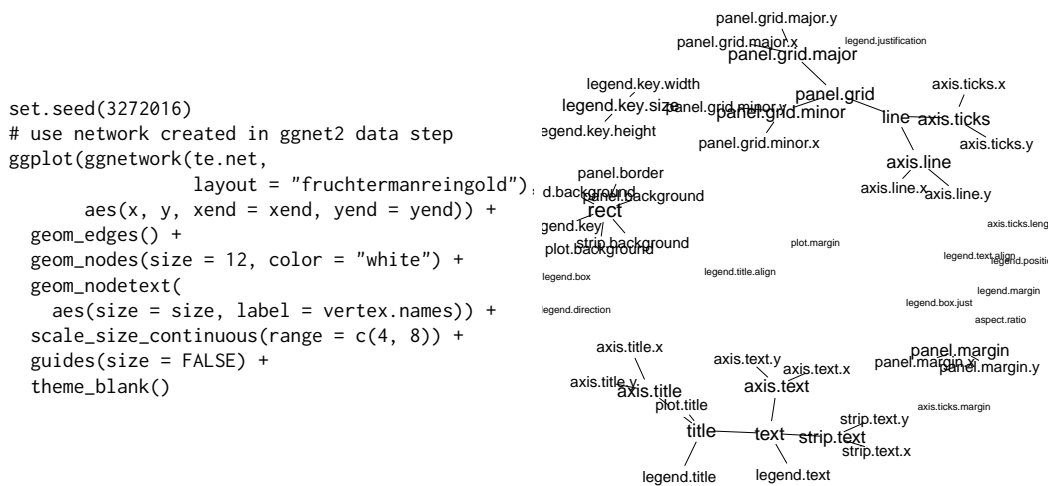


Figure 5: (continued) Inheritance structure of `ggplot2` theme elements. This is a recreation of the graph found at <http://docs.ggplot2.org/current/theme.html>.

```

head(TENet)
## Source: local data frame [6 x 3]
## Groups: from_id [2]
##
##   from_id    to_id  degree
##   <fctr>    <fctr> <dbl>
## 1   text      title 6.403124
## 2   text legend.text 6.403124
## 3   text axis.text 6.403124
## 4   text strip.text 6.403124
## 5   line axis.line 5.567764
## 6   line axis.ticks 5.567764

```

Note the various ways the packages adjust the side of the labels to correspond to the outdegree of the nodes, including the use of the `scale_size_continuous` function in Figure 5(c). In each of these plots, it is easy to quickly determine parent-child relationships, and to assess which theme elements are unrelated to all others. Nodes with the most children are the `rect`, `text`, and `line` elements, so we made their labels larger in order to emphasize their importance. In each case, the label size is a function of the out degree of the vertices.

College football

This next example comes from M.E.J. Newman's network data web page (Girvan and Newman, 2002). It is an undirected network consisting of all regular season college football games played between Division I schools in Fall of 2000. There are 115 vertices and 613 edges: each vertex represents a school, and an edge represents a game played between two schools. There is an additional variable in the vertex data frame corresponding to the conference each team belongs to, and there is an additional variable in the edge data frame that is equal to one if the game occurred between teams in the same conference or zero if the game occurred between teams in different conferences. We take a look at the data used in the plots below.

```

fb.net
## Network attributes:
## vertices = 115
## directed = TRUE
## hyper = FALSE
## loops = FALSE
## multiple = FALSE

```

```

## bipartite = FALSE
## total edges= 613
## missing edges= 0
## non-missing edges= 613
##
## Vertex attribute names:
## conf vertex.names
##
## Edge attribute names:
## same.conf
head(ftnet)
## from_id to_id same.conf value
## 1 AirForce NevadaLasVegas 1 Mountain West
## 2 Akron MiamiOhio 1 Mid-American
## 3 Akron VirginiaTech 0 Mid-American
## 4 Akron Buffalo 1 Mid-American
## 5 Akron BowlingGreenState 1 Mid-American
## 6 Akron Kent 1 Mid-American
## schools
## 1
## 2
## 3
## 4
## 5
## 6

```

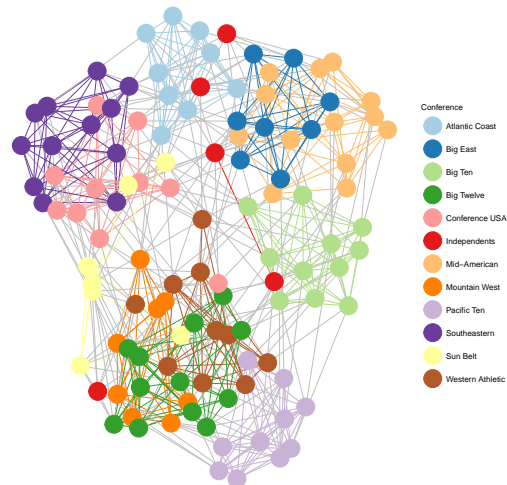
The network of football games is given in Figure 6. Here, the linetype aesthetic corresponds to games that occur between teams in the same conference or different conferences.

```

#make data accessible
data(football, package = 'geomnet')
rownames(football$vertices) <-
  football$vertices$label
# create network
fb.net <- network(football$edges[, 1:2],
  directed = TRUE)
# create node attribute
# (what conference is team in?)
fb.net %v% "conf" <-
  football$vertices[
    network.vertex.names(fb.net), "value"
  ]
# create edge attribute
# (between teams in same conference?)
set.edge.attribute(
  fb.net, "same.conf",
  football$edges$same.conf)
set.seed(5232011)
ggnet2(fb.net, mode = "fruchtermanreingold",
  color = "conf", palette = "Paired",
  color.legend = "Conference",
  edge.color = c("color", "grey75"))

```

(a) ggnet2



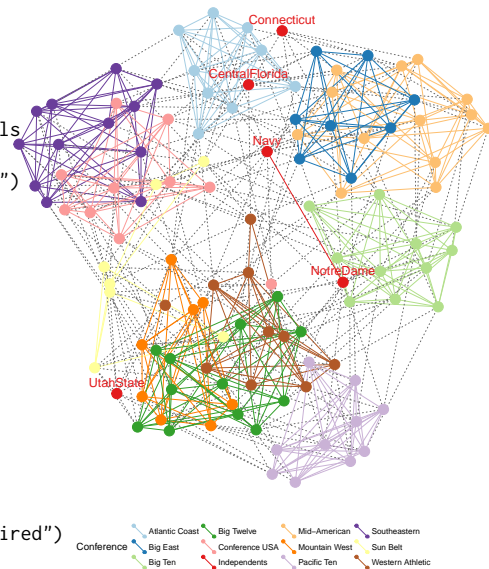
These lines are dotted and solid, respectively. We have also assigned a different color to each conference, so that the vertices and their labels are colored according to their conference. Additionally, in the first two implementations, the edges between two teams in the same conference share that conference color, while edges between teams in different conferences are a default gray color. This coloring and changing of the line types make the structure of the game network easier to view. Additionally, we use the label aesthetic in Figure 6(b) to label only a few schools that are of interest to us. This is the conference consisting of Navy, Notre Dame, Utah State, Central Florida, and Connecticut, which is spread out, whereas most other conferences' teams are all very close to each other because they play within conference much more than they play out of conference. At the time, these five schools were all independents and did not have a home conference. Without the coloring capability, we would not have been able to pick out that difference as easily.

(b) geomnet

```
# data step: merge vertices and edges
# data step: merge vertices and edges
ftnet <- fortify(as.edgdf(football$edges),
                football$vertices)

# create new label variable for independent schools
ftnet$schools <- ifelse(
  ftnet$value == "Independents", ftnet$from_id, "")

# create data plot
set.seed(5232011)
ggplot(data = ftnet,
        aes(from_id = from_id, to_id = to_id)) +
  geom_net(layout.alg = 'fruchtermanreingold',
           aes(colour = value, group = value,
               linetype = factor(same.conf != 1),
               label = schools),
           linewidth = 0.5,
           size = 5, vjust = -0.75, alpha = 0.3) +
  theme_net() +
  theme(legend.position = "bottom") +
  scale_colour_brewer("Conference", palette = "Paired")
guides(linetype = FALSE)
```



(c) ggnetwork

```
# use network from ggnet2 step
set.seed(5232011)
ggplot(
  ggnetwork(
    fb.net,
    layout = "fruchtermanreingold"),
  aes(x, y, xend = xend, yend = yend)) +
  geom_edges(
    aes(linetype = as.factor(same.conf)),
    color = "grey50") +
  geom_nodes(aes(color = conf), size = 4) +
  scale_color_brewer("Conference",
                    palette = "Paired") +
  scale_linetype_manual(values = c(2,1)) +
  guides(linetype = FALSE) +
  theme_blank()
```

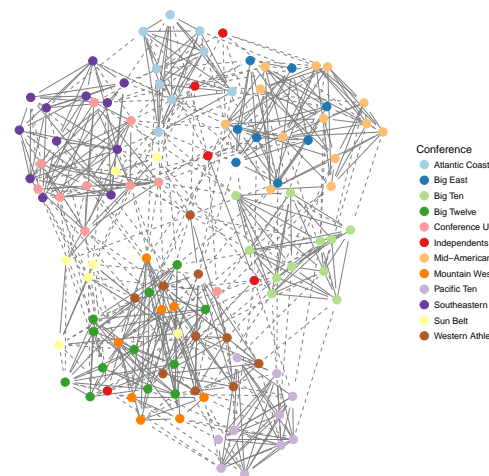


Figure 6: (continued) The network of regular season Division I college football games in the season of fall 2000. The vertices and their labels are colored by conference.

Southern women

Bipartite (or ‘two-mode’) networks are networks with two different kinds of nodes and where all ties are formed between these two kinds. Affiliation networks, which represent the ties between individuals and the groups to which they belong, are examples of such networks (see Newman, 2010, p. 53-54 and p. 123-127).

One of the classic examples for a two-mode network is the network of 18 Southern women attending 14 social events as collected by Davis et al. (1941) and published e.g. as part of the `tnet` package (Opsahl, 2009). In this data, a woman is linked by an edge to an event if she attended it. One of the questions for these type of networks is gain insight in the interplay between the two different sets of nodes.

The data for the example of the Southern women is reported as edge list in form of ‘lady X attending event Y’. With a bit of data preparation as detailed below, we can visualize the graph as shown in Figure 7. In creating the plots, we use the shape and colour aesthetics to map the two different modes to two different shapes and colours.

```
# access the data and rename it for convenience
library(tnet)

data(tnet)
elist <- data.frame(Davis.Southern.women.2mode)
names(elist) <- c("Lady", "Event")
```

The edge list for the Southern women’s data consists of women attending events:

```
head(elist,4)
##   Lady Event
## 1    1     1
## 2    1     2
## 3    1     3
## 4    1     4
```

In order to distinguish between nodes from different types, we have to add an additional identifier element, so that we can tell the ‘first’ woman *L1* apart from the first event, *E1*.

```
elist$Lady <- paste("L", elist$Lady, sep="")
elist$Event <- paste("E", elist$Event, sep="")

davis <- elist
names(davis) <- c("from", "to")
davis <- rbind(davis, data.frame(from=davis$to, to=davis$from))
davis$type <- factor(c(rep("Lady", nrow(elist)), rep("Event", nrow(elist))))
```

The two different types of nodes are shown by different shapes and colors. We see the familiar relationship between events and groups of women attending these events. Women attending the same events then form a tighter knit subset, while events are also thought of as more similar, if they are attended by the same women. This defines the cluster of events *E1* through *E5*, which are only attended by women 1 through 9, while events *E6* through *E9* are attended by (almost) everybody making them the core group of events.

Bike sharing in Washington D.C.

The data shows trips taken with bikes from the bike share company Capital Bikeshare⁵ during the second quarter of 2015. While this bike sharing company is located in the heart of Washington D.C. the company offers a set of bike stations just outside of Washington in Rockville, MD and north of it. Each station is shown as a vertex, and edges between stations indicate that at least five trips were taken between these two stations; the wider the line, the more trips have been taken between stations. In order to reflect distance between stations, we use as an additional restriction that the fastest trip was at most ten minutes long. Figure 8 shows four renderings of this data. The first is a geographically true

⁵<https://secure.capitalbikeshare.com/>

representation of the area overlaid by lines between bike stations, the other three are networks drawn with **geomnet**, **ggnet2**, and **ggnetwork**, respectively. The code for these renderings is shown below:

```
# make data accessible
data(bikes, package = 'geomnet')
# data step for geomnet
tripnet <- fortify(as.edgedf(bikes$trips), bikes$stations[,c(2,1,3:5)])
# create variable to identify Metro Stations
tripnet$Metro = FALSE
idx <- grep("Metro", tripnet$from_id)
tripnet$Metro[idx] <- TRUE

# plot the bike sharing network shown in Figure 7b
set.seed(1232016)
ggplot(aes(from_id = from_id, to_id = to_id), data = tripnet) +
  geom_net(aes(linewidth = n / 15, colour = Metro),
    labelon = TRUE, repel = TRUE) +
  theme_net() +
  xlim(c(-0.1, 1.1)) +
  scale_colour_manual("Metro Station", values = c("grey40", "darkorange")) +
  theme(legend.position = "bottom")

# data preparation for ggnet2 and ggnetwork
bikes.net <- network(bikes$trips[, 1:2 ], directed = FALSE)
# create edge attribute (number of trips)
network::set.edge.attribute(bikes.net, "n", bikes$trips[, 3 ] / 15)
# create vertex attribute for Metro Station
bikes.net %v% "station" <- grepl("Metro", network.vertex.names(bikes.net))
bikes.net %v% "station" <- 1 + as.integer(bikes.net %v% "station")
rownames(bikes$stations) <- bikes$stations$name
# create node attributes (coordinates)
bikes.net %v% "lon" <-
  bikes$stations[ network.vertex.names(bikes.net), "long" ]
bikes.net %v% "lat" <-
  bikes$stations[ network.vertex.names(bikes.net), "lat" ]
bikes.col <- c("grey40", "darkorange")

# Non-geographic placement
set.seed(1232016)
ggnet2(bikes.net, mode = "fruchtermanreingold", size = 4, label = TRUE,
  vjust = -0.5, edge.size = "n", layout.exp = 1.1,
  color = bikes.col[ bikes.net %v% "station" ],
  label.color = bikes.col[ bikes.net %v% "station" ])

# Non-geographic placement. Use data from ggnet2 step.
set.seed(1232016)
ggplot(data = ggnetwork(bikes.net, layout = "fruchtermanreingold"),
  aes(x, y, xend = xend, yend = yend)) +
  geom_edges(aes(size = n), color = "grey40") +
  geom_nodes(aes(color = factor(station)), size = 4) +
  geom_nodetext(aes(label = vertex.names, color = factor(station)),
    vjust = -0.5) +
  scale_size_continuous("Trips", breaks = c(2, 4, 6), labels = c(30, 60, 90)) +
  scale_colour_manual("Metro station", labels = c("FALSE", "TRUE"),
    values = c("grey40", "darkorange")) +
  theme_blank() +
  theme(legend.position = "bottom", legend.box = "horizontal")
```


To plot the geographically correct bike network layout in **geomnet**, we use the `'layout.alg = NULL'` option and provide the latitude and longitude coordinates of the bike stations from the company's data. A glance of the data that we used in the examples is shown below.

```
bikes.net
## Network attributes:
## vertices = 20
## directed = FALSE
## hyper = FALSE
## loops = FALSE
## multiple = FALSE
## bipartite = FALSE
## total edges= 53
## missing edges= 0
## non-missing edges= 53
##
## Vertex attribute names:
## lat lon station vertex.names
##
## Edge attribute names:
## n
head(tripnet[, -c(4:5, 8)])
##           from_id
## 1 Broschart & Blackwell Rd
## 2 Crabbs Branch Way & Calhoun Pl
## 3 Crabbs Branch Way & Calhoun Pl
## 4 Crabbs Branch Way & Calhoun Pl
## 5 Crabbs Branch Way & Calhoun Pl
## 6 Crabbs Branch Way & Calhoun Pl
##           to_id n      lat      long
## 1           <NA> NA 39.10210 -77.20032
## 2 Crabbs Branch Way & Redland Rd 11 39.10771 -77.15207
## 3 Needwood Rd & Eagles Head Ct 14 39.10771 -77.15207
## 4 Rockville Metro East 51 39.10771 -77.15207
## 5 Rockville Metro West 8 39.10771 -77.15207
## 6 Shady Grove Metro West 36 39.10771 -77.15207
## Metro
## 1 FALSE
## 2 FALSE
## 3 FALSE
## 4 FALSE
## 5 FALSE
## 6 FALSE
```

Because all three approaches result in the same picture, we only show one of these in Figure 8a. The code for creating the map is given here:

```
library(ggmap)
metro_map <- get_map(location = c(left = -77.22257, bottom = 39.05721,
                                right = -77.11271, top = 39.14247))

# geomnet: overlay bike sharing network on geographic map
ggmap(metro_map) +
  geom_net(data = tripnet, layout.alg = NULL, labelon = TRUE,
           vjust = -0.5, ealpha = 0.5,
           aes(from_id = from_id,
               to_id = to_id,
               x = long, y = lat,
               linewidth = n / 15,
```

```

    colour = Metro)) +
  scale_colour_manual("Metro Station", values = c("grey40", "darkorange")) +
  theme_net() %>% replace_theme(aspect.ratio=NULL, legend.position = "bottom") +
  coord_map()

```

We can also make use of the option `'layout.alg = NULL'` whenever we do not want to use an in-built layout algorithm but make use of a user-defined custom layout. In this case, the coordinates of the layout have to be created outside of the visualization and x and y coordinates have to be made available instead.

Some considerations of speed

In our examples thus far, we have focused on rather small social or relationship networks and one larger communication network. Now we present an example of a biological network, which comes from Jeong et al. (2001). It is the complete protein-protein interaction network in the yeast species *S. cerevisiae*. There are 2,113 proteins that make up the vertices of this network, with a total of 4480 edges between them. These edges represent "direct physical interactions" between any two proteins (Jeong et al., 2001, p. 42), resulting in a relatively large network. When these interactions and their associated proteins are plotted using the Fruchterman-Reingold layout algorithm, the runtime is extremely long, about 9.5 minutes for 50,000 iterations through the algorithm. The resulting layout is shown in Figure 9. When testing the three approaches with the larger network, we decided to use a random layout to save time. Despite its size, each one of the approaches in the **ggplot2** framework can be drawn in a few hundred milliseconds.

Another benefit that emerges from using **ggplot2** for network visualization is the speed at which it can plot fairly large networks. In order to assess the speed gain procured by our three approaches, we ran two separate tests, both of which designate **ggplot2**-based approaches as faster than the plotting functionality offered in the **network** package. They also show the **ggplot2** approaches to be largely on par with the speed provided by the **igraph** package. We first investigate average random layout plotting time of the protein network

shown in Figure 9, and then consider average plotting times of increasingly larger random networks. Note that in all tests, default package settings were used. The code to create benchmark results for both of these situations is provided in the vignette of the package **ggCompNet** (Tyner and Hofmann, 2016b). See the Supplementary Material section at the end of this paper for more information.

We plotted the protein interaction network of Figure 9 100 times using the **network** and **igraph** packages, and compared their run times to 100 runs each of the three visualization approaches introduced in this paper. The results are shown in Figure 10. We can see that on average, the **ggplot2** framework provides a two to three-fold increase in speed over the **network** package, and that **geomnet** and **ggnetwork** are faster than package **igraph**. The three **ggplot2** approaches also have considerably less variability in time than the **network** package. Despite the large number of vertices, the protein interaction network has a relatively small number of edges (4480 out of over 2.2 million theoretically possible connections resulting in an edge probability of just over 0.0020). Next, we examine networks with a higher edge probability.

The second test relies on random undirected networks in which the probability of an edge between two nodes was set to $p = 0.2$. We generated 100 of these networks at network sizes from 25 to 250 nodes, using increments of 25.

Figure 11 summarizes the results of these benchmarks using a convenience sample of machines accessible to the authors, including authors' hardware and additional results from friends' and colleagues' machines. Network sizes are plotted horizontally, execution times of 100 runs under each visualization approach are plotted on the y -axis. Each panel shows a different machine as indicated by the facet label. Note that each panel is scaled separately to account for differences in the overall speed of these machines. What these plots indicate is that we have surprisingly large variability in relative run times across different machines. However, the results support some general findings. The **network** plotting routine is by far the slowest across all machines, while the **igraph** plotting is generally among the fastest. Our three approaches generally feature in between **igraph** and **network** with **ggnet2** being as fast or faster than **igraph** plotting, followed by **ggnetwork** and **geomnet**, which is generally the slowest among the three. These differences become more pronounced as the size of the network increases.

Although speed was not the main rationale for our inquiry into **ggplot2**-based approaches to network visualization, a speed-based comparison shows a clear advantage of these approaches over

the plotting function included in the **network** package, which very quickly becomes much slower as network size increases.

Summary and discussion

At first glance, the three visualization approaches may seem nearly identical. However, each one brings unique strengths to the visualization of networks. Out of our three approaches, **ggnetwork** is most flexible and allows for a re-ordering of layers to emphasize one over the other. The flexibility is useful but does require the user to specify every single part of the network visualization. The **geomnet** implementation most closely aligns with the existing **ggplot2** paradigm because it provides a single layer that can be added to other **ggplot2** layers. **ggnet2** requires the user to know the least about the **ggplot2** framework, while resulting in a valid and extensible **ggplot2** object. Many features of the packages would not have been possible, or would have at least been difficult to implement, in prior versions of **ggplot2**. The increased flexibility of the current development version as well as the added geoms `geom_curve` and `geom_label` provided us with a strong, yet flexible, foundation for network visualization. Our approaches also benefit from the speed of **ggplot2**, making network visualization more efficient than the existing framework of **network** for a lot of the benchmark examples.

All three approaches rely on the package **sna** for layouts. This allows the user to access the many layout algorithms available for networks, and in the event that new layouts are implemented in **sna**, our packages will accommodate them seamlessly. A larger range of layouts is available through **igraph**, and can be implemented into our packages by setting the respective layout arguments to `NULL` and passing `x, y` coordinates calculated from **igraph**. There are some notable differences between the packages, such as in the parameters used for specific layout algorithms, e.g. **igraph** allows the use of weights for Fruchterman-Reingold placement, even though it is unclear from the original article how these are supposed to affect the layout. In all three approaches, it is feasible to tap into **igraph**'s functionality in a future version so that the user does not need to calculate the layout separately. Additional future work will explore the implementation of other network data structures, such as the `networkDynamic` class from **statnet**, which would benefit from the faceting capabilities of our implementations. This work will likely incorporate the `fortify` approach of **ggnetwork** and `geomnet::fortify.network()` for converting network data structures to a **ggplot2**-friendly format.

We have found that none of our approaches is unequivocally the best. We can, however, provide some guidance as to which approach is best for which type of user. The main differences between the three methods are in the way that network information is passed into the functions. For **ggnet2** and **ggnetwork**, data management and attribute handling is done through network operators on nodes and edges, while the **geomnet** approach does not require any knowledge of networks or existing network analysis packages from the user. This likely affects the user base of each package. We think that users who are well-versed with networks will find **ggnet2** and **ggnetwork** more intuitive to use than **geomnet**. These users might be looking to **ggplot2** as another avenue to create high-quality visualizations that tap into **ggplot2** advantages such as facetting and, for **ggnetwork**, layering. Users who are already familiar with **ggplot2** and some of the other **tidyverse** packages (see Wickham (2017)), and who find themselves dealing with network data will likely be more attracted to the **geomnet** implementation of network plotting. The data management skills needed for using **geomnet** are basic: some familiarity with the split-apply-combine paradigm, in the form of familiarity with **plyr** or **dplyr**, would be sufficient in order to make full use of the features of `geom_net` (Wickham, 2011). All in all, the three approaches we have presented here provide a wealth of resources to users of all skill sets who are looking to create beautiful network visualizations.

On a personal level we discovered that the collaboration on this paper has helped us to improve upon our initial versions of each of these packages. For instance, the edge coloring in the `ggnet2` function was designed so that edges between two vertices in the same group were colored with that group's vertex color. This inspired an implementation of it in **geomnet** through the traditional **ggplot2** group operator. During the process of writing the paper the authors collaborated on a solution for the problem of nodes being plotted on top of arrow tips. This solution was implemented in the **geomnet** `arrow.gap` parameter, which allows to re-track the tip of an arrow on a directed edge, and was also added to **ggnetwork**. In addition, the implementation of a **ggplot2** geom for networks within **geomnet** inspired the creation of the aliased geoms of the **ggnetwork** package.

Finally, curious users may be interested in how these three packages can fit together and replicate each other, since they are in fact so similar. Thanks to the flexibility inherent to **ggnetwork**, it is possible to write wrapper functions around **ggnetwork** functions in order to recreate the behavior and functionality of **ggnet2** and **geomnet**. Simple examples of such wrapper functions, called `ggnetwork2` and `geom_network`, respectively are shown below.

```
library(ggnetwork)
```

```

# mimics geom_net behavior
geom_network <- function(edge.param, node.param) {
  edge_ly <- do.call(geom_edges, edge.param)
  node_ly <- do.call(geom_nodes, node.param)
  list(edge_ly, node_ly)
}
# mimics ggnet2 behavior
ggnetwork2 <- function() { ggplot() + geom_network() }

```

Similarly, **geomnet** can mimic the the behavior of **ggnet2**, as shown below.

```

library(geomnet)
geomnet2 <- function(net) {
  ggplot(data = fortify(net),
    aes(from_id = from_id, to_id = to_id)) +
  geom_net()
}

```

Mimicking **ggnetwork** with **geomnet** requires a little bit more work because the native data input for **geomnet** is a "data.frame" object fortified with **geomnet** methods, not a "network" object. Instead, the internal **ggplot2** function `ggplot_build` allows a plot created with **geomnet** function calls to be recreated with **ggnetwork**-like syntax. An example of using a **geomnet** plot to create a similar plot in the style of **ggnetwork** follows to reproduce Figure 2(c).

```

library(geomnet)
library(ggnetwork)
library(dplyr)
# a ggnetwork-like creation using a geomnet plot
data("blood")
# first, create the geomnet plot to access the data later
geomnetplot <- ggplot(data = blood$edges, aes(from_id = from, to_id =
  to)) +
  geom_net(layout.alg = "circle", selfloops = TRUE) +
  theme_net()
# get the data
dat <- ggplot_build(geomnetplot)$data[[1]]
# ggnetwork-like construction for re-creating network shown in Figure 5
ggplot(data = dat, aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_segment(arrow = arrow(type = 'closed'), colour = 'grey40') +
  geom_point(size = 10, colour = 'darkred') +
  geom_text(aes(label = from), colour = 'grey80', size = 4) +
  geom_circle() +
  theme_blank() + theme(aspect.ratio = 1)

```

Supplementary Material

Software: **ggnetwork** 0.5.1 and **geomnet** 0.2.0 were used to create the visualizations. **ggnet2** is part of **GGally** 1.3.0.

Reproducibility: All the code used in the examples is available as a vignette in the CRAN package **ggCompNet**. There are two vignettes: one for the speed comparisons and one for the visualizations provided in the Examples section. The package also provide our speed test data for creating Figure 11. We created this package to accompany this paper with the hope that interested users will compare these methods on their own systems and against their own code. Finally, all of the data we use in the examples, with the exception of the bipartite network example, is included as a part of the **geomnet** package.

Acknowledgements

The authors would like to thank the reviewers for their thoughtful input to and thorough reviews of our manuscript. We would also like to thank the editor of The R Journal for his enduring patience.

Bibliography

- B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2016. URL <https://CRAN.R-project.org/package=gridExtra>. R package version 2.2.1. [p39]
- S. Bender-deMoll. *Ndtv: Network Dynamic Temporal Visualizations*, 2016. URL <https://CRAN.R-project.org/package=ndtv>. R package version 0.10.0. [p39]
- M. Bojanowski. *Intergraph: Coercion Routines for Network Data Objects*, 2015. URL <http://mbojan.github.io/intergraph>. R package version 2.0-2. [p30]
- F. Briatte. *Ggnetwork: Geometries to Plot Networks with 'ggplot2'*, 2016. URL <https://github.com/briatte/ggnetwork>. R package version 0.5.1. [p27, 29]
- S. V. Buldyrev, R. Parshani, G. Paul, H. E. Stanley, and S. Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025–1028, 2010. [p28]
- C. T. Butts. network: a Package for Managing Relational Data in R. *Journal of Statistical Software*, 24(2), 2008. [p27]
- C. T. Butts. *Sna: Tools for Social Network Analysis*, 2014. URL <http://CRAN.R-project.org/package=sna>. R package version 2.3-2. [p27]
- C. T. Butts, M. S. Handcock, and D. R. Hunter. *Network: Classes for Relational Data*. Irvine, CA, 2014. URL <http://statnet.org/>. R package version 1.10.2. [p27, 31]
- W. Chang. *Gcookbook: Data for "R Graphics Cookbook"*, 2012. URL <https://CRAN.R-project.org/package=gcookbook>. R package version 1.0. [p28]
- W. Chang. *R Graphics Cookbook*. O'Reilly, Sebastopol, CA, 2013. ISBN 978-1449316952. [p28]
- K. Cook, G. Grinstein, and M. Whiting. VAST Challenge 2014. <http://hci12.cs.umd.edu/newwarepository/benchmarks.php>, 2014. [p37]
- G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*:1695, 2006. URL <http://igraph.org>. [p27]
- A. Davis, B. B. Gardner, and M. R. Gardner. *Deep South: A Social Anthropological Study of Caste and Class*. The University of Chicago Press, Chicago, IL, 1941. [p47]
- T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. [p30, 34]
- M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99(12):7821–7826, 2002. [p28, 44]
- M. S. Handcock, D. R. Hunter, C. T. Butts, S. M. Goodreau, and M. Morris. Statnet: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, 24(1):1–11, 2008. URL <http://www.jstatsoft.org/v24/i01>. [p27]
- M. Horikoshi and Y. Tang. *Ggfortify: Data Visualization Tools for Statistical Analysis Results*, 2016. URL <http://CRAN.R-project.org/package=ggfortify>. R package version 0.4.1. [p28]
- H. Jeong, S. P. M. A.-L. Barabási, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411:41–42, 2001. [p50]
- B. H. Junker and F. Schreiber. *Analysis of Biological Networks*. Wiley Series in Bioinformatics. John Wiley & Sons, 2008. ISBN 9780470253465. URL <https://books.google.com/books?id=2D1oLXaXSNgC>. [p27]
- 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>. [p28]
- T. Kamada and S. Kawai. An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, 31(1):7–15, 1989. [p32]
- M. E. J. Newman. *Networks : An Introduction*. Oxford University Press, Oxford New York, 2010. ISBN 978-0199206650. [p28, 47]
- T. Opsahl. *Structure and Evolution of Weighted Networks*. University of London (Queen Mary College), London, UK, 2009. URL <http://toreopsahl.com/publications/thesis/>. [p47]

- C. Prell. *Social Network Analysis: History, Theory and Methodology*. SAGE Publications, 2011. ISBN 9781446290132. URL <https://books.google.com/books?id=wZYQAgAAQBAJ>. [p27]
- B. Schloerke, J. Crowley, D. Cook, H. Hofmann, H. Wickham, F. Briatte, M. Marbach, and E. Thoen. *GGally: Extension to Ggplot2.*, 2016. R package version 1.3.0. [p28, 29]
- K. Slowikowski. *Ggrepel: Repulsive Text and Label Geoms for 'ggplot2'*, 2016. URL <https://CRAN.R-project.org/package=ggrepel>. R package version 0.5. [p33]
- R. Tamassia, editor. *Handbook of Graph Drawing and Visualization*. CRC Press, 2013. [p27]
- Y. Tang, M. Horikoshi, and W. Li. Ggfortify: Unified interface to visualize statistical result of popular r packages. *The R Journal*, 2016. URL <http://CRAN.R-project.org/package=ggfortify>. [p28]
- S. Tyner and H. Hofmann. *Geomnet: Network Visualization in the 'ggplot2' Framework*, 2016a. URL <http://github.com/sctyner/geomnet>. R package version 0.2.0. [p27]
- S. Tyner and H. Hofmann. *ggCompNet: Compare Timing of Network Visualizations*, 2016b. URL <https://CRAN.R-project.org/package=ggCompNet>. R package version 0.1.0. [p50]
- D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998. [p28]
- D. J. Watts. The “New” Science of Networks. *Annual Review of Sociology*, 30:243–270, 2004. [p27]
- H. Wickham. The Split-Apply-Combine Strategy for Data Analysis. *Journal of Statistical Software*, 40(1): 1–29, 2011. URL <http://www.jstatsoft.org/v40/i01/>. [p51]
- H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2016. ISBN 978-3-319-24277-4. URL <http://ggplot2.org>. [p27, 34, 42]
- H. Wickham. *Tidyverse: Easily Install and Load 'tidyverse' Packages*, 2017. URL <https://CRAN.R-project.org/package=tidyverse>. R package version 1.1.1. [p51]
- L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag, New York, 1999. [p27]
- T. Yin, D. Cook, and M. Lawrence. Ggbio: An R package for extending the grammar of graphics for genomic data. *Genome Biology*, 13(8):R77, 2012. [p28]
- G. Yu, D. K. Smith, H. Zhu, Y. Guan, and T. T.-Y. Lam. Ggtree: An R package for visualization and annotation of phylogenetic trees with their covariates and other associated data. *Methods in Ecology and Evolution*, 8(1):28–36, 2017. ISSN 2041-210X. URL <https://doi.org/10.1111/2041-210x.12628>. [p28]

Samantha Tyner
Department of Statistics and Statistical Laboratory
Iowa State University
United States
sctyner@mail.iastate.edu

François Briatte
European School of Political Sciences
Catholic University of Lille
France
francois.briatte@univ-catholille.fr

Heike Hofmann
Department of Statistics and Statistical Laboratory
Iowa State University
United States
hofmann@mail.iastate.edu

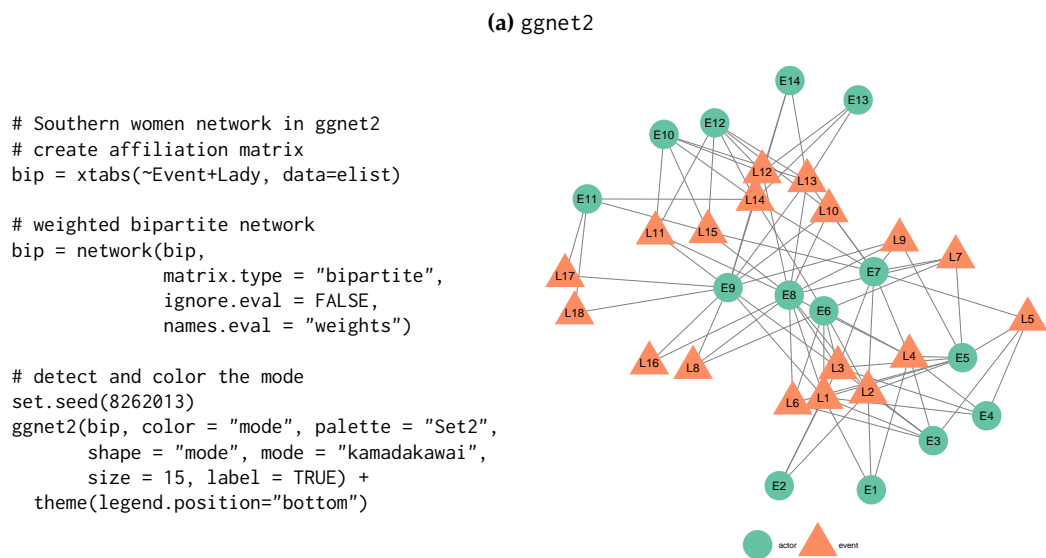
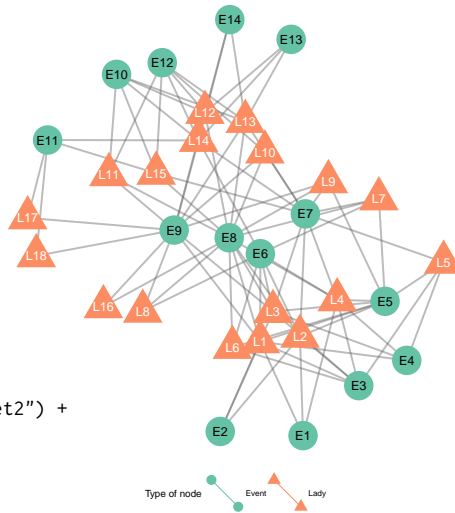


Figure 7: Graph of the Southern women data. Women are represented as orange triangles, events as green circles.

(b) geomnet

```
# Southern women network in geomnet
# change labelcolour
davis$labelcolour <-
  c("white", "black")[as.numeric(davis$type)]

set.seed(8262013)
ggplot(data = davis) +
  geom_net(layout.alg = "kamadakawai",
    aes(from_id = from, to_id = to,
      colour = type, shape = type),
    size = 15, labelon = TRUE, ealpha = 0.25,
    vjust = 0.5, hjust = 0.5,
    labelcolour = davis$labelcolour) +
  theme_net() +
  scale_colour_brewer("Type of node", palette = "Set2") +
  scale_shape("Type of node") +
  theme(legend.position = "bottom")
```



(c) ggnetwork

```
# Southern women network in ggnetwork. Use data f
# assign vertex attributes (Node type and label)
set.vertex.attribute(bip, "mode",
  c(rep("event", 14), rep("woman", 18)))
set.seed(8262013)
ggplot(data = ggnetwork(bip,
  layout = "kamadakawai"),
  aes(x = x, y = y, xend = xend, yend = yend))
  geom_edges(colour = "grey80") +
  geom_nodes(aes(colour = mode, shape = mode),
    size = 15) +
  geom_nodetext(aes(label = vertex.names)) +
  scale_colour_brewer(palette = "Set2") +
  theme_blank() +
  theme(legend.position = "bottom")
```

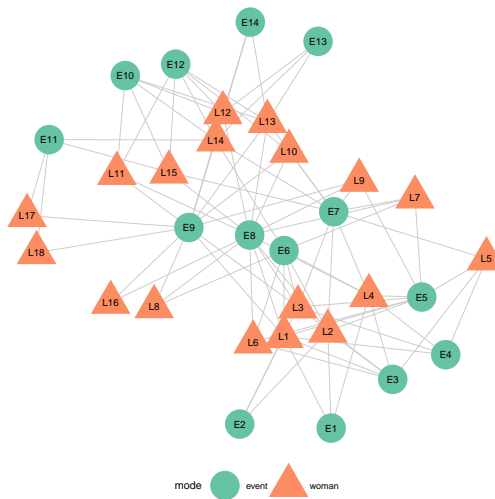


Figure 7: Graph of the Southern women data. Women are represented as orange triangles, events as green circles.

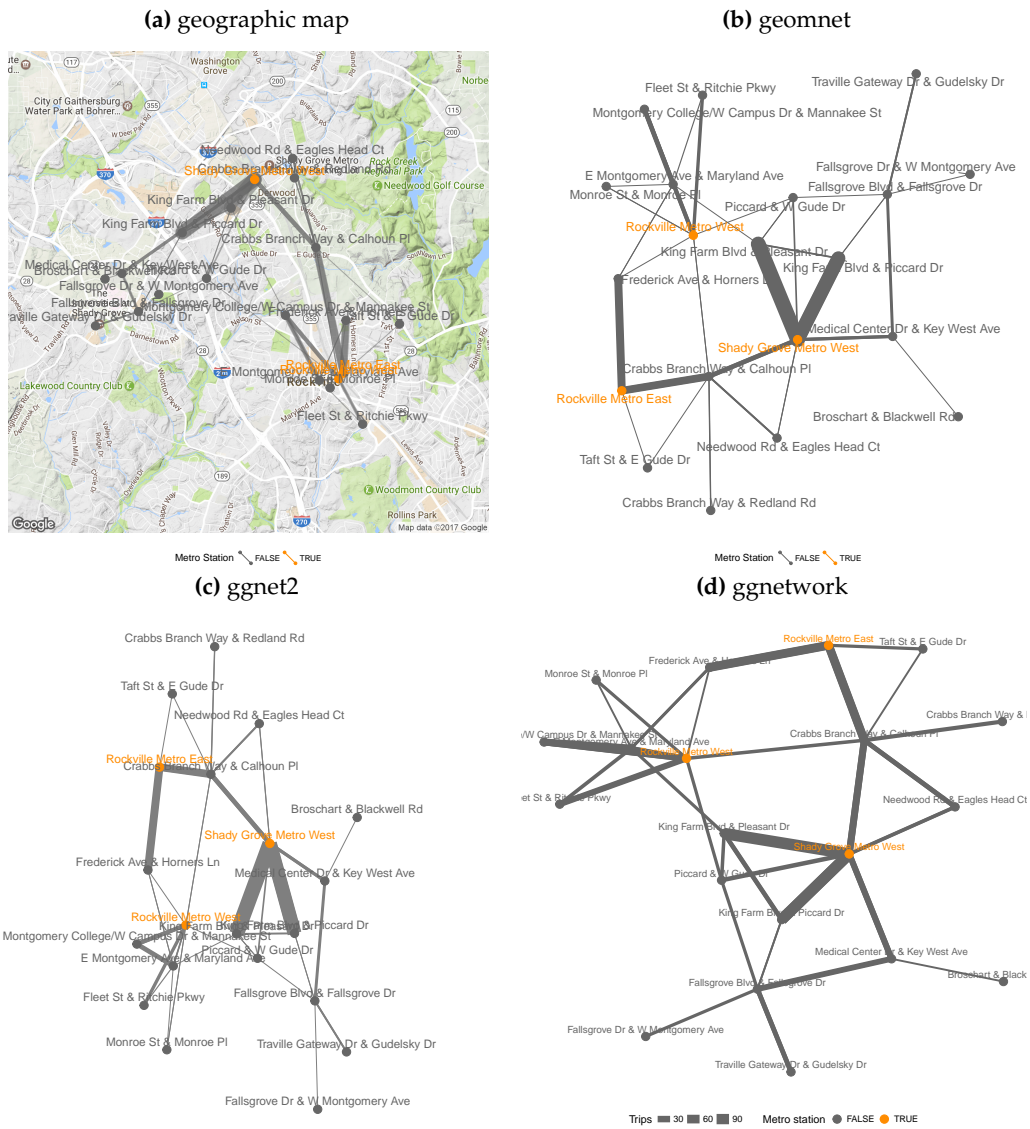


Figure 8: Network of bike trips using a geographically true representation (top left) overlaid on a satellite map, a Kamada-Kawai layout in **geomnet** (top right), a Fruchterman-Reingold layout in **ggnet2** (bottom left) and **ggnetwork** (bottom right). Metro stations are shown in orange. In both the Kamada-Kawai and the Fruchterman-Reingold layouts, metro stations take a much more central position than in the geographically true representation.



Figure 9: Protein-protein interaction network in *S. cerevisiae*. A Fruchterman-Reingold algorithm allowed to run for 50,000 iterations produced the coordinates for the nodes.

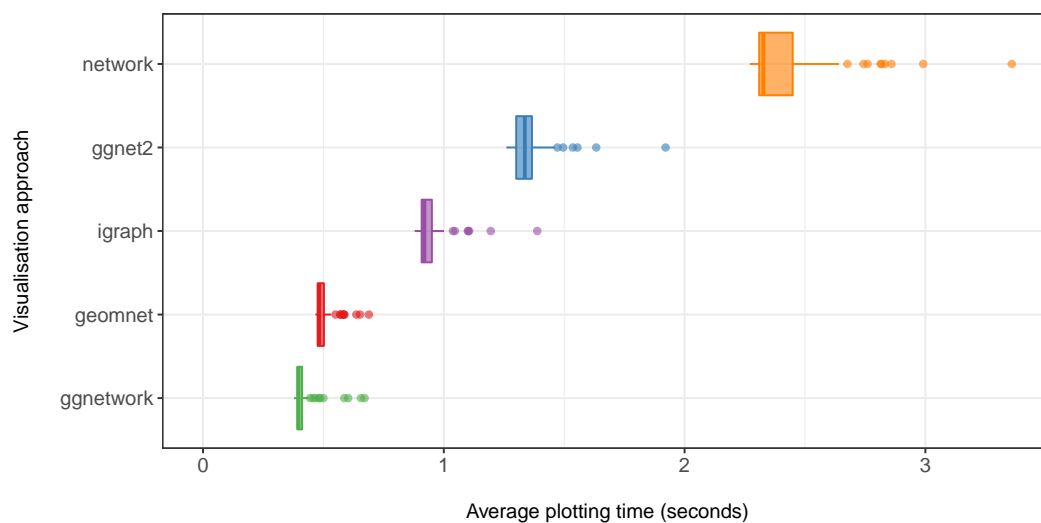


Figure 10: Comparison of the times needed for calculating and rendering the previously discussed protein interaction network in the three **ggplot2** approaches and the standard plotting routines of the **network** and **igraph** packages based on 100 evaluations each.

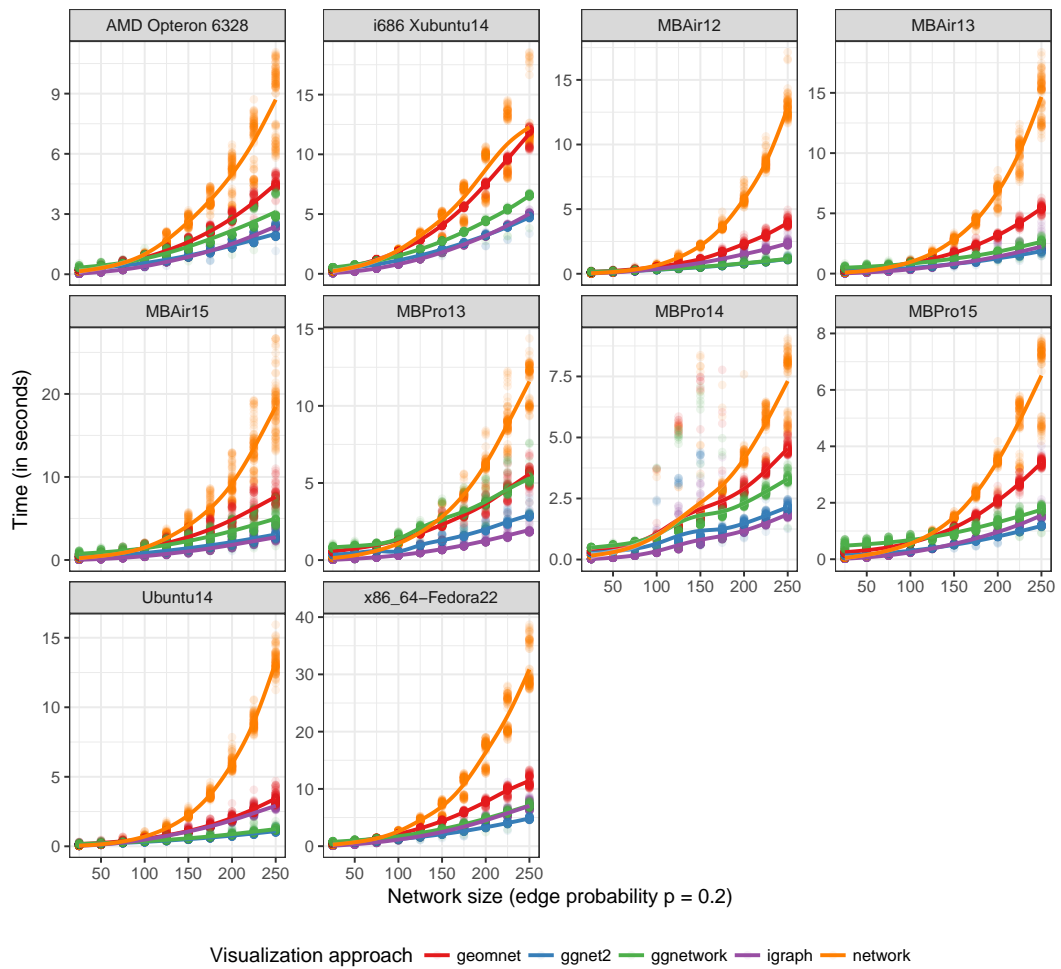


Figure 11: Plotting times of random undirected networks of different sizes under each of the available visualization approaches using their default settings. Note that each panel is scaled independently to highlight relative differences in the visualization approaches rather than speed of different hardware.

OrthoPanels: An R Package for Estimating a Dynamic Panel Model with Fixed Effects Using the Orthogonal Reparameterization Approach

by Mark Pickup, Paul Gustafson, Davor Cubranic and Geoffrey Evans

Abstract This article describes the R package **OrthoPanels**, which includes the function `opm()`. This function implements the orthogonal reparameterization approach recommended by Lancaster (2002) to estimate dynamic panel models with fixed effects (and optionally: wave specific intercepts). This article provides a statistical description of the orthogonal reparameterization approach, a demonstration of the package using real-world data, and simulations comparing the estimator to the known-to-be-biased OLS estimator and the commonly used GMM estimator.

Introduction

Panel data includes observations on N cases repeated over multiple, $T > 1$, time points (waves). A dynamic panel model is one that includes one or more lagged dependent variables. When fixed effects are included in such models, the OLS estimator is biased when T is fixed (small). This has become known as Nickell bias (Nickell, 1981). A maximum likelihood estimation of such a model leads to an incidental parameters problem (Neyman and Scott, 1948; Lancaster, 2000). Lancaster proposes an orthogonal reparameterization approach to produce a conditional likelihood estimator that is exact and consistent as $N \rightarrow \infty$ for $T \geq 2$ (Lancaster, 2002).¹

This article provides an introduction to the R package **OrthoPanels**, which includes the function `opm()`. This function implements the orthogonal reparameterization approach recommended by Lancaster (2002) to estimate dynamic panel models with fixed effects (and optionally: wave specific intercepts). In this article, we first provide a brief review of the methods implemented in **OrthoPanels**, and then discuss an empirical illustration using some of the features of the function `opm()`. Finally, we conduct some Monte Carlo simulations to demonstrate the performance of the `opm()` function under different assumptions about the data generating process.

Review of methods

Panel models are used when we have observations on N cases repeated over T time points. These models allow us to account for unobservable individual effects (unobserved heterogeneity) that can neither be identified nor controlled with cross-sectional models (Finkel, 2008; Hsiao, 2014, pages 4–10). The most general approach to accounting for unobservable individual effects, in that it makes the fewest assumptions, is a fixed effects panel model (Hsiao, 2014, pages 47–56).

A dynamic panel model contains one or more lags of the dependent variable on the right-hand side. A dynamic model is necessary if the dependent variable is autoregressive (Arellano, 2003, page 129). We would expect a dependent variable to be autoregressive if we believe that subsequent to something happening that temporarily changes the value of the dependent variable (e.g., a temporary shift in an independent variable), the dependent variable will return partly but not entirely to its original value before the next observation. The consequence is that we have a dependent variable in which values that are above/below average at one observation are more likely than not to be above/below average at the next observation. This autoregressive relationship between current and past values of the dependent variable is not due to an omitted variable that links current and past values and not due to correlation between the current and past values of the errors. The autoregressive relationship is due to past values of the dependent variable having a direct effect on current values (Finkel, 2008, page 486). A simple example of a data generating process that might require a dynamic model with fixed effects (and wave specific intercepts) is:

$$\begin{aligned} y_{it} &= \rho y_{it-1} + \beta_1 x_{it} + \tau_t + \mu_{it} \\ \mu_{it} &= \eta_i + \epsilon_{it}, \epsilon_{it} \sim NID(0, \sigma^2) \end{aligned} \quad (1)$$

¹Note this means two waves of data used in the estimation and data on the dependent variable from a third earlier wave ($T = 0$). This is due to the need to have values for the lagged dependent variable at $T = 1$.

where $i = 1, \dots, N$ and $t = 1, \dots, T$; y_{it} is the value of the dependent variable y for case i at time t ; x_{it} is the value of the independent variable x for case i at time t ; and ρ and β are parameters. The process is first order autoregressive with one lag of the dependent variable on the right-hand side. For each case, there is a case-specific fixed effect η_i , also known as unobservable individual effects. These represent (over time) average differences between cases. There may also be over time changes in the average value of y_{it} across cases, τ_t . These are changes common to all cases at a particular point in time and include trending as a special case. The following discussion holds whether or not τ_t are included. They are a useful control for dynamics like trending and global wave effects but need not be included if such dynamics do not exist in the data.

The estimation of (1) presents a challenge. The OLS fixed-effects estimator with a lag dependent variable is biased with a fixed (small) T . In the analysis of econometric panel data this has become known as Nickell bias. Nickell (1981) provided the analytical expressions of the bias that had been previously well documented by Monte Carlo work (Nerlove, 1967, 1971). Nickell showed that if the autoregressive parameter, ρ in (1), is positive, the bias will be negative. He also demonstrated that this bias persists even as ρ goes to zero. As we will see, the bias in the estimation of the autoregressive parameter has important biasing effects on the estimation of the short and long-run effects (Greene, 2012, pages 422–423) of dynamic variables (e.g., x_{it}).

A popular alternative to the OLS estimation is the Anderson and Hsiao (1981) instrumental variable approach and more generally the Arellano and Bond GMM estimator (1991), and other GMM estimators (Arellano and Bover, 1995; Blundell et al., 2000). The downside of these estimators is that they are inefficient (Behr, 2003). Further, the GMM approach uses approximate inference methods and requires assumptions about the appropriateness of past values of the dependent variable (and possibly independent variables) as instruments. These assumptions may or may not be valid. Lancaster (2000) argues the GMM estimator contains no data or information that is not already contained in the likelihood for the model. Hsiao et al. (2002) propose a transformed-likelihood approach to dealing with the incidental parameters problem. Lancaster (2002) proposes a conditional likelihood estimator (also a type of transformed-likelihood approach) that can analytically compute the conditional probability distributions, over our variables of interest, exactly. Such likelihood-based estimators require no instrumental variable assumptions. We refer to the approach suggested by Lancaster as orthogonal reparameterization (or OPM for orthogonalized panel model). This approach can be conceived of as a Bayesian version of the frequentist, transformed-likelihood approach proposed and tested by Hsiao, Pesaran and Tahmiscioglu (2002). See (Pesaran, 2015, pages 692–695) and (Hsiao, 2014, pages 80–135) for a review of likelihood-based methods developed to estimate linear dynamic panel models with fixed effects. The transformed-likelihood approach has been shown to perform better than GMM estimators (Hsiao, 2014; Hsiao et al., 2002), particularly when the autoregressive parameter is close to 1. The advantage of the OPM approach, over the transformed-likelihood estimators proposed by Hsiao et al. (2002) is that the latter requires knowing the appropriate assumptions regarding the initial conditions for y_{it} at $t = 0$ and their relationship with the unobservable individual effects. This is often not known and if the wrong assumptions are made, the estimator is no longer consistent (Anderson and Hsiao, 1981; Hsiao, 2014, pages 86–98). This is not a requirement of the OPM approach.

The orthogonal reparameterization (OPM) approach

The Lancaster likelihood-based estimator proceeds as follows. The maximum likelihood estimation of model (1) with a fixed (small) T leads to an *incidental parameters problem*. Maximum likelihood estimation is consistent as N increases relative to the number of parameters estimated. With fixed T , the number of fixed-effects (η_i) approaches infinity at the same rate as $N \rightarrow \infty$. In other words, each time we add a case, we add a parameter to be estimated. Therefore, N relative to the number of parameters cannot increase and we cannot rely on asymptotics as $N \rightarrow \infty$ and the application of maximum likelihood leads to inconsistent estimates. Lancaster (2002) suggested a solution to this particular incidental parameters problem.

The key to this approach is that we are not actually interested in estimates of the incidental parameters (η_i). We are interested in estimates of the “common parameters” – in (1) this is the effect of dynamic variable x_{it} (β_1), and τ_t , ρ , and σ^2 . Therefore, we seek a reparameterization of the incidental parameters so that the incidental and common parameters are information orthogonal. This puts us in a position to produce an $N \rightarrow \infty$ consistent likelihood-based estimate of the parameters of interest that is independent of the values of the incidental parameters. Therefore, we continue to have incidental parameters but not an incidental parameter problem.

A straightforward example of the OPM approach (Lancaster, 2002) is as follows. We wish to estimate a model that includes parameters η_i , τ_t , ρ and β_1 and error variance σ^2 – where η_i represents the fixed effects, τ_t represents wave specific effects, ρ is the first order autoregressive parameter and β_1 represents the effect of dynamic variable x_{it} . We denote the likelihood function for the data for a

single case as $\ell_i(\eta_i, \tau_t, \rho, \beta_1, \sigma^2)$.

Suppose the fixed effects can be reparameterized so that the likelihood function for the data for a single case factors as:

$$\ell_i(\eta_i, \tau_t, \rho, \beta_1, \sigma^2) = \ell_{i1}(\eta_i)\ell_{i2}(\tau_t, \rho, \beta_1, \sigma^2) \tag{2}$$

Where ℓ_{i1} and ℓ_{i2} are themselves likelihood functions. If the parameters (η_i) and $(\tau_t, \rho, \beta_1, \sigma^2)$ are also variation independent, they are orthogonal. If $\prod \ell_{i2}$ is the product of the ℓ_{i2} for all observations, it can then be shown that the application of maximum likelihood to $\prod \ell_{i2}$ produces consistent estimates of $(\tau_t, \rho, \beta_1, \sigma^2)$ (Lancaster, 2002) as $N \rightarrow \infty$ for any $T \geq 2$.

Not all likelihoods can be transformed so that the incidental parameters are orthogonalized. However, a solution with an equivalent intuition may be available. It may be possible to reparameterize the fixed effects so that they are information orthogonal. If we denote the log of the likelihood for the data for observation i as L_i , then the fixed effects are information orthogonal to a parameter (e.g., β_1) if the following condition is met:

$$E \left(\frac{\partial^2 L_i}{\partial \eta_i \partial \beta_1} \right) = 0 \tag{3}$$

This can be interpreted as meaning that the slope of the log likelihood with respect to η_i is independent of the slope of the log likelihood with respect to β_1 . If a transformation of the fixed effects can be found that meets this condition (and the equivalent condition for the other parameters – in this case τ_t, ρ and σ^2), it may be possible to place priors on the parameters and integrate out the fixed effects. Specifically, we use flat priors for the η_i and the remaining parameters. This is essentially a Bayesian estimation technique. This gives us the marginal posterior for the remaining parameters. Lancaster (2002) demonstrates how this can be done for the lagged dependent variable model with fixed effects.

Once we have the marginal posterior for our parameters of interest, we can use Monte Carlo methods to sample values from the marginal posterior to produce estimates and credible intervals for the parameters, as follows. We wish to estimate (1). To generalize a little, let us allow for K dynamic variables, so that $\beta X_{i,t} = \sum_{k=1}^K \beta_k X_{i,t,k}$. To simplify the notation, we express $X_{i,t}$, $y_{i,t}$ and $y_{i,t-1}$ in vector terms X_i , y_i and y_{i-} . As the τ_t are optional, depending on the data generating process, we leave them out of the following discussion. The appropriate reparameterization of the fixed effects, forming uniform priors on ρ, β_1, σ^2 , and $\{\eta_i\}$, and integrating out the fixed effects results in the following posterior density function²:

$$p(\rho, \beta, \sigma^2 | data) \propto \sigma^{-(N(T-1)-2)} \exp \left\{ \frac{N}{T} \sum_{t=1}^{T-1} \left(\frac{T-t}{t} \rho^t \right) - \frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - \rho y_{i-} - \beta X_i)' H (y_i - \rho y_{i-} - \beta X_i) \right\} \tag{4}$$

where H is defined as an operator that subtracts the mean. For example, if

$$\omega_i = y_i - \rho y_{i-} - \beta_1 X_i$$

then $H(\omega_i) \equiv \omega_i - \bar{\omega}$.

Sampling from this posterior (4) gives us estimates (distributions) for ρ, β and σ^2 . To do this we begin by integrating β out of (4). This gives us the following joint posterior density:

$$p(\rho, \sigma^2 | data) \propto \sigma^{-(N(T-1)-K-2)} \exp \left\{ \frac{N}{T} \sum_{t=1}^{T-1} \left(\frac{T-t}{t} \rho^t \right) \right\} \exp \left\{ - \frac{1}{2\sigma^2} \left(\left(\sum_{i=1}^N (X_i)' H (y_i - \rho y_{i-}) \right)' \left(\sum_{i=1}^N (X_i)' H (X_i) \right)^{-1} \left(\sum_{i=1}^N (X_i)' H (y_i - \rho y_{i-}) \right) \right\} \tag{5}$$

Next, we integrate out σ^2 from (5) giving us the marginal posterior density:

$$p(\rho | data) \propto \frac{\exp \left\{ \frac{N}{T} \sum_{t=1}^{T-1} \left(\frac{T-t}{t} \rho^t \right) \right\}}{\left((y_i - \rho y_{i-})' H (y_i - \rho y_{i-}) - \left(\sum_{i=1}^N (X_i)' H (y_i - \rho y_{i-}) \right)' \left(\sum_{i=1}^N (X_i)' H (X_i) \right)^{-1} \left(\sum_{i=1}^N (X_i)' H (y_i - \rho y_{i-}) \right) \right)^{\frac{(N(T-1)-K)}{2}}} \tag{6}$$

We can now proceed to sample triplet values $(\rho, \beta, 1/\sigma^2)$ by: first sampling ρ from (6); then, given ρ , sample $1/\sigma^2$ from (5); then given ρ and $1/\sigma^2$, sample β from (4). When we sample values of $1/\sigma^2$ from (5) given ρ , we are sampling from a gamma distribution. When we sample values of β from (4) given ρ and $1/\sigma^2$, we are sampling from a multivariate normal. The medians of the sampled

²This varies slightly from 3.24 in Lancaster (2002). The prior density for σ^2 had not been included.

values give us our parameter estimates and the 2.5 and 97.5 percentiles give us our 95 percent credible intervals.

The limitation of this approach is that it must be worked out for each family of models. For example, the necessary reparameterization will differ for a panel probit model. Lancaster (2000) provides reparameterizations for Poisson count models, static linear panel models and dynamic (stationary and nonstationary) linear panel models. Estimates of dynamic, binary models by this approach are still in their early days (é and Kyriazidou, 2000; Arellano and Bonhomme, 2009). The approach relies on the Bayesian idea of integrating out the fixed effects to give us the marginal posterior distribution for the remaining parameters – although, this has many similarities to the frequentist idea of a conditional likelihood (Cox and Reid, 1987). For a review and comparison of these approaches, see Lancaster (2000).

The OrthoPanels package: empirical example

In this section, we demonstrate the use of the **OrthoPanels** package with an empirical example.³ The data we will use is from the 2010 British Election Study, using 3 waves of panel survey data with 1845 respondents. Our dependent variable is government approval. The wording of the survey question that produced the variable is: “On a scale that runs from 0 to 10, where 0 means strongly dislike and 10 means strongly like, how do you feel about the Labour Party?” Response categories were from 0 ‘Strongly dislike’ to 10 ‘Strongly like’, and ‘Don’t Know’. Independent variables (x_{it}) included in the model are:

1. Evaluations of the leaders of the Conservative, Liberal Democrat and Labour parties, measured using the question: “On a scale that runs from 0 to 10, where 0 means strongly dislike and 10 means strongly like, how do you feel about (David Cameron/Nick Clegg/Gordon Brown)?” ‘0<Strongly dislike’; ‘10>Strongly like’; ‘Don’t Know’.
2. A standard retrospective assessment of the national economic situation, or ‘sociotropic’ evaluation, using the question: “How do you think the general economic situation in this country has changed over the last 12 months? Has it:” Response options are: ‘got a lot worse’ (1); ‘got a little worse’ (2); ‘stayed the same’ (3); ‘got a little better’ (4); ‘got a lot better’ (5); ‘Don’t Know’.
3. Two policy evaluation questions. These are evaluations of how the government has handled the NHS and terrorism.⁴ For example: “How well do you think the present government has handled the National Health Service?” ‘Very well’ (5); ‘Fairly well’ (4); ‘Neither well nor badly’ (3); ‘Fairly badly’ (2); ‘Very badly’ (1); ‘Don’t Know’.
4. A measure of the respondent’s ideology based on preference for increasing or cutting taxes, using the question: “Using the 0 to 10 scale below, where the end marked 0 means that government should cut taxes a lot and spend much less on health and social services, and the end marked 10 means that government should raise taxes a lot and spend much more on health and social services, where would you place yourself on this scale?” ‘0<Government should cut taxes a lot and spend much less on health and social services’; ‘10>Government should increase taxes a lot and spend much more on health and social services’; ‘Don’t Know’.
5. And a measure of whether the respondent identifies with the governing Labour party, using the question: “Generally speaking, do you think of yourself as Labour, Conservative, Liberal Democrat or what?” This is recoded 1 for Labour and 0 otherwise.

For each variable, a ‘Don’t know’ response was identified as missing and the corresponding case was deleted. Note though that **OrthoPanels** can accommodate unbalanced panel data under the assumption that the data is missing at random.

Data structure

The variables for the model to be estimated can be contained in a frame, list, or environment and the model can be specified symbolically with a formula as described below.

The variables for our empirical example are included in the data frame `BES_panel`. This data frame is included with the **OrthoPanels** package.

```
> head(BES_panel)
  n t Econ Clegg Brown Cameron Approve NHS Terror PID Tax
1 1 1   3     0     9     0     7     0     0  1  6
```

³We include a second empirical example in Appendix 1.

⁴The terrorism question in 2010 is replaced with a handling of the war in Afghanistan question.

```

2 2 1 4 0 10 0 8 0 0 1 8
3 3 1 3 0 5 4 7 0 0 0 6
4 4 1 2 0 7 3 4 0 0 1 6
5 5 1 2 0 0 0 0 0 0 0 5
6 6 1 2 0 7 0 8 0 0 1 4

```

In this data frame, "Econ" refers to evaluations of the national economy; "Clegg" refers to evaluations of the Liberal Democrat leader; "Brown" refers to evaluations of the Labour leader; "Cameron" refers to evaluations of the Conservative leader; "Approval" refers to government approval; "NHS" refers to evaluations on how the government has handled the NHS; "Terror" refers to evaluations on how the government has handled terrorism; "PID" refers to whether the individual identifies as Labour; and "Tax" refers to the respondent's ideology based on their preference for cutting taxes. In addition to the variables, the data frame contains the vectors "n" and "t" indicating the case number and wave for each observation. If the independent and dependent variables are contained in 3- and 2-dimension arrays, these vectors are unnecessary because the index is implicit in the organization of the arrays.

Interface

We begin by specifying our model using a formula, indicating the data to be used, and the case and time variables:

```
> BES.opm.model<-opm(Approve ~ Econ + Clegg + Brown + Cameron + NHS + Terror + PID + Tax,
data = BES_panel, index = c('n','t'), n.samp = 10000, add.time.indicators = TRUE)
```

The first argument is a formula specifying the model symbolically: $\text{response} \sim \text{term1} + \text{term2}$. This is consistent with the `lm()` function. It is not necessary to include the lagged dependent variable or the fixed effects in the model specification. This is done automatically.

The other arguments are: `data` which specifies the data frame, list or environment containing the variables in the model; `n.samp` which specifies the number of Monte Carlo iterations to use to estimate the parameters; `index` which is a two-element vector containing the names of the case and time variables, respectively; and `add.time.indicators` which is a logical argument. If the `add.time.indicators` is `TRUE`, the model includes dummy variables for each wave (time point). The default is `FALSE`. The `data` and `index` arguments are optional. If `data` is not specified, the variables are taken from the environment from which `opm()` is called. If `index` is not specified, the first two vectors are assumed to be the case and time indices, respectively. An additional optional argument is `subset`. This is a vector specifying a subset of observations to be used in the estimation.

The function `opm()` returns an object of class "opm" which includes a list, `samples`, with the following elements: "rho", a vector of `n.samp` parameter samples of ρ ; "v", a vector of `n.samp` parameter samples of σ^2 ; and "beta", a `n.samp` by `x` variable matrix of parameter samples of β . If included in the model, the parameters for the time dummies are included in this matrix. The summary of the object provides us with the median, 68% equal tailed credible intervals and 95% equal tailed credible intervals for each parameter.

```
> summary(BES.opm.model)
```

Call:

```
opm(x = Approve ~ Econ + Clegg + Brown + Cameron + NHS + Terror +
PID + Tax, data = BES_panel, index = c("n", "t"), n.samp = 10000,
add.time.indicators = TRUE)
```

Parameter estimates:

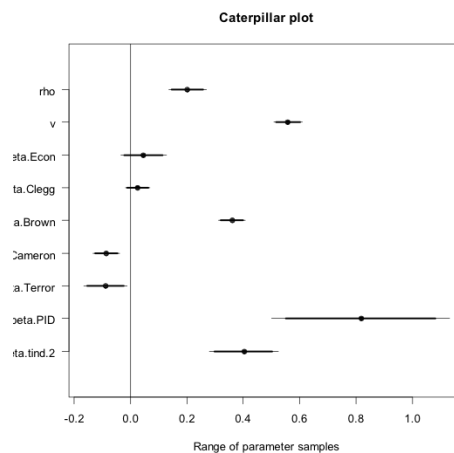
	<--95CI	<--68CI	med	68CI-->	95CI-->
rho	0.128000	0.15800000	0.189000	0.224000	0.25800000
sig2	1.611477	1.68193076	1.759734	1.846594	1.93906618
beta.Econ	-0.038086	0.00077932	0.041545	0.081118	0.12235574
beta.Clegg	-0.020535	0.00060795	0.022996	0.045020	0.06678845
beta.Brown	0.297306	0.31959250	0.343490	0.367214	0.39072453
beta.Cameron	-0.124476	-0.10195910	-0.077744	-0.054332	-0.03195979
beta.NHS	-0.235810	-0.18692841	-0.136087	-0.085502	-0.03647700
beta.Terror	-0.153162	-0.11566089	-0.077034	-0.037453	-0.00036214
beta.PID	0.488818	0.64660493	0.804710	0.959541	1.10995704
beta.Tax	0.017598	0.04022762	0.063176	0.086574	0.10903010
beta.tind.2	0.217953	0.27900281	0.342908	0.406506	0.46890678

The package includes functions that may be of use in exploring the results. The function `confint()` computes equal tailed credible intervals for one or more parameters in the fitted `opm` model. We can calculate 90% equal tailed credible intervals as follows:

```
> confint(BES.opm.model, level = 0.90)
              5%          95%
rho           0.13800000 0.24700000
sig2          1.63217694 1.90996203
beta.Econ     -0.02625512 0.10794038
beta.Clegg    -0.01312387 0.06009414
beta.Brown     0.30435125 0.38292909
beta.Cameron  -0.11665040 -0.03883916
beta.NHS      -0.21928907 -0.05272413
beta.Terror   -0.14110536 -0.01271258
beta.PID       0.53904739 1.05947755
beta.Tax       0.02570175 0.10189246
beta.tind.2    0.23669235 0.4483367
```

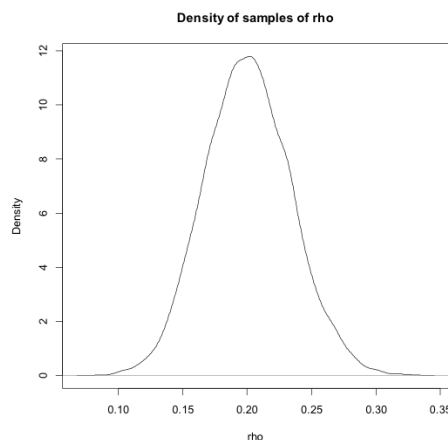
The function `caterplot()` creates side-by-side plots of credible intervals of the `opm` model parameters. The intervals are displayed as horizontal lines, with the 90% interval using a thicker line width and the 95% interval a thinner one. The posterior median is indicated with a dot.

```
caterplot(BES.opm.model)
abline(v=0)
```



We can use the function `plot()` to obtain the posterior density of each parameter.

```
plot(BES.opm.model, "rho")
```



In a dynamic model, the β coefficients are the immediate effects of a change in each of the covariates. This is known as the short-run effect. However, in a dynamic model, the short-run effect is not the full effect of a change in a covariate. As the future value of the dependent variable depends on the current value, the effect of a change in a covariate has a knock on effect into the future. Assuming a stationary

process, the effects asymptotically declined to zero over time. The cumulation of all these effects is known as the long-run effect and is estimated as $\beta/(1 - \rho)$, where ρ is the autoregressive parameter (Wooldridge, 2013; Pickup, 2014). We can calculate the median and 95% equal tailed credible intervals for the long-run effects from the parameter samples. For example, for the first independent variable (the economy):

```
> quantile(BES.opm.model$samples$beta[, 1] / (1 - BES.opm.model$samples$rho),
+ probs = c(0.025, 0.5, 0.975))
      2.5%      50%      97.5%
-0.04723233  0.05098789  0.15025695
```

The opm() function: Monte Carlo simulations

To demonstrate the performance of the OPM estimator operationalized by the `opm()` function, we use it to estimate models from simulated data with a known data generating process.⁵ We use the following data generating process to generate our simulated data sets:

$$\begin{aligned}y_{it} &= \rho y_{it-1} + \beta x_{it} + \mu_{it} \\ \mu_{it} &= \eta_i + \varepsilon_{1it} \\ x_{it} &= 0.75\eta_i + \varepsilon_{2it} \\ \varepsilon_{1it} &\sim NID(0,1); \quad \varepsilon_{2it} \sim NID(0,16)\end{aligned}$$

We use $\beta = 0.50$ and we generate data sets with two values for ρ (0.5 and 0.9). The unobserved fixed values η_i , are generated from a uniform distribution: $U(-1, 1)$. Note that both the y_{it} and x_{it} are a function of the fixed effects, with x_{it} containing 75 percent of the fixed effect for y_{it} . As a consequence, it is necessary to account for the η_i in our data model in order to get an unbiased estimate of β and ρ .⁶

We generate panel data sets with an N of 1000 and a T of 2, 3, 4, and 9. A T of 2 means we collected three waves of data but we lose the first wave in order to have a measure of the lagged dependent variable in the second wave. This effectively means we have 2 waves of data for the purposes of estimation. This is as small of a T as we can get and still estimate a dynamic model. We generate 1000 data sets for each value of T and ρ .

Table 1, panels A through D, report the results of the simulations when the autoregressive parameter, ρ , is 0.9. This is a ‘close to worst case scenario’ in that the autoregressive parameter is very high. This is when Nickell bias and the incidental parameters problem will be at their worst. However, the autoregressive parameter is not so high that we have to begin worrying about issues of near integrated data. The tables present the results when the number of waves of data available is 3, 4, 5 and 10 respectively (i.e., T of 2, 3, 4, and 9).

The table reports the average value estimated for the autoregressive parameter (ρ), the β and the long-run effect. This is an indication of how accurate the estimator is on average. The tables also include the median value estimated for the long-run effect, for reasons described later. The tables include the proportion of estimates in which the true value for each parameter is included in the 95 percent confidence/credible interval (the coverage probability). This is an indicator of how useful the estimator is for hypothesis testing. Finally, tables report the root mean squared error for each parameter. This is an indication of how far off the true value the estimator will be on average. If an estimator is on average correct but could, for any given estimation, produce estimates far off the true value, its functionality is greatly diminished.

In addition to presenting the results for the `opm()` function (denoted OPM), the tables also present the results from an OLS fixed effects estimation and from a GMM estimation (Blundell and Bond, 1998; Arellano and Bover, 1995). The OLS fixed effects results demonstrate the estimation bias that the OPM estimator is designed to rectify. It is also a commonly used estimator for this type of panel data, despite the known bias (Nickell, 1981). The GMM approach is probably the most commonly used alternative and appears to be the next best performing estimator compared to likelihood-based estimators (Hsiao et al., 2002). There are a number of variations on the GMM approach (Croissant and Millo, 2008). We used a difference GMM with a two-step estimator. We use the available lags of the dependent variable as instruments. This specification was chosen as it seemed to produce the best results given our data generating process. We used the `plm` package to produce the GMM estimates (Croissant and Millo, 2008). This is a very versatile package for estimating panel models within R.

⁵Example code for running such simulations is provided in the vignette included in Appendix 2.

⁶When generating the data sets, we set y_{i0} equal to the series equilibrium and include a burn-in period of 50 waves.

A (T=2; N=1000; Rho=0.9; Beta=0.5; LR Effect=5)										
Approach	Rho (Average)	Beta (Average)	LR Effect (Average)	LR Effect (Median)	Rho (95% CI)	Beta (95% CI)	LR Effect (95% CI)	Rho (RMSE)	Beta (RMSE)	LR Effect (RMSE)
OLS, FE	0.59	0.42	1.05	1.04	0	0	0	0.31	0.08	3.95
GMM	0.92	0.5	1.98	2.21	0.97	0.97	0.72	0.18	0.05	36.57
OPM	0.9	0.5	5.89	5.17	0.93	0.94	0.92	0.03	0.01	2.79
B (T=3; N=1000; Rho=0.9; Beta=0.5; LR Effect=5)										
Approach	Rho (Average)	Beta (Average)	LR Effect (Average)	LR Effect (Median)	Rho (95% CI)	Beta (95% CI)	LR Effect (95% CI)	Rho (RMSE)	Beta (RMSE)	LR Effect (RMSE)
OLS, FE	0.71	0.45	1.54	1.54	0	0	0	0.19	0.05	3.46
GMM	0.9	0.5	6.17	3.67	0.95	0.96	0.80	0.09	0.02	60.75
OPM	0.9	0.5	5.29	5.03	0.92	0.94	0.91	0.02	0.007	1.26
C (T=4; N=1000; Rho=0.9; Beta=0.5; LR Effect=5)										
Approach	Rho (Average)	Beta (Average)	LR Effect (Average)	LR Effect (Median)	Rho (95% CI)	Beta (95% CI)	LR Effect (95% CI)	Rho (RMSE)	Beta (RMSE)	LR Effect (RMSE)
OLS, FE	0.76	0.47	1.93	1.92	0	0	0	0.14	0.03	3.07
GMM	0.9	0.5	50.22	4.46	0.94	0.94	0.85	0.06	0.01	1633.78
OPM	0.9	0.5	5.09	4.98	0.94	0.94	0.93	0.01	0.005	0.68
D (T=9; N=1000; Rho=0.9; Beta=0.5; LR Effect=5)										
Approach	Rho (Average)	Beta (Average)	LR Effect (Average)	LR Effect (Median)	Rho (95% CI)	Beta (95% CI)	LR Effect (95% CI)	Rho (RMSE)	Beta (RMSE)	LR Effect (RMSE)
OLS, FE	0.84	0.49	3.07	3.07	0	0.02	0	0.06	0.01	1.93
GMM	0.9	0.5	4.92	4.81	0.92	0.92	0.88	0.02	0.005	0.92
OPM	0.9	0.5	5.02	5.02	0.95	0.94	0.94	0.005	0.003	0.26
E (T=2; N=1000; Rho=0.5; Beta=0.5; LR Effect=1)										
Approach	Rho (Average)	Beta (Average)	LR Effect (Average)	LR Effect (Median)	Rho (95% CI)	Beta (95% CI)	LR Effect (95% CI)	Rho (RMSE)	Beta (RMSE)	LR Effect (RMSE)
OLS, FE	0.29	0.45	0.63	0.63	0	0	0	0.21	0.05	0.37
GMM	0.5	0.5	1.01	1.00	0.96	0.95	0.96	0.04	0.01	0.11
OPM	0.5	0.5	1.01	1.01	0.93	0.94	0.92	0.03	0.01	0.07
F (T=3; N=1000; Rho=0.5; Beta=0.5; LR Effect=1)										
Approach	Rho (Average)	Beta (Average)	LR Effect (Average)	LR Effect (Median)	Rho (95% CI)	Beta (95% CI)	LR Effect (95% CI)	Rho (RMSE)	Beta (RMSE)	LR Effect (RMSE)
OLS, FE	0.37	0.47	0.75	0.75	0	0.008	0	0.13	0.03	0.25
GMM	0.5	0.5	1.00	1.00	0.94	0.95	0.94	0.02	0.01	0.06
OPM	0.5	0.5	1.00	1.00	0.92	0.94	0.92	0.02	0.01	0.04

Table 1: Simulation Results

Beginning with a T of 2 (Table 1, Panel A), we can see just how badly biased the OLS fixed effects estimates are. With true values $\rho = 0.9$, $\beta = 0.5$ and LR effect of 5. The OLS estimator produces, on average, values of $\rho = 0.59$, $\beta = 0.42$ and LR effect of 1.05. The bias is substantial for all parameters. Partly as a result of these very large biases, this approach virtually never produces 95 percent confidence intervals that include the true values of the parameters. The root mean squared errors are particularly large for the long-run effects and ρ . Clearly, this approach is not appropriate under these circumstances.

When we look at the results for the OPM and GMM approaches (again with $T = 2$), both look very good for the average estimates of ρ and β . There is essentially no bias for either. The 95 percent confidence intervals include the true values around 95 percent of the time and the root mean squared errors are generally small, although the root mean squared error for rho is 83 percent smaller for the OPM approach. When we look at the results for the long-run effects, the two approaches diverge. Using the average estimate of the long-run effect as a measure of performance, the OPM estimator overestimates the long-run effect with a bias of 0.89, while the GMM estimator underestimates the long-run effect with a bias of 3.02. Further, while the 95 percent confidence intervals of the OPM estimator include the true value about 92 percent of the time, the GMM confidence intervals only include the true value about 72 percent of the time.

It may seem curious that the GMM estimator is so poor at estimating the long-run effect, when, on average, it is good at estimating ρ and β – especially since the long-run effect is simply a function of ρ and β : $\beta/(1 - \rho)$. The problem occurs because of the reasonable but slightly large root mean squared error on the estimates of ρ . With the true value of 0.9 for ρ , the root mean squared error of the GMM estimates means it sometimes produces values very close to 1 for ρ . The division of $1 - \rho$ in the calculation of the long-run effect means that such values for ρ can translate into large errors in the estimate of the long-run effect.⁷ Because of a smaller root mean squared error, the OPM estimator suffers far less from this problem.

It must be noted then that for the reasons just discussed the mean estimate is not a very stable measure of the performance of the GMM estimates of the long-run effect. For the GMM estimator, under these circumstances, small differences in the estimates of ρ produce large differences in the average estimate of the long-run effect. A more stable measure is the median estimated long-run effect. For the GMM approach this is closer to the true value (2.21). However, the median OPM estimate is much better (5.17). Further, while the median GMM estimate is better than the mean GMM estimate, it does not change the fact that in any given application of the estimator, a very large error in the long-run effect can occur. This is the story of the root mean squared error for the long-run effect. These are moderate for the OPM estimator, 2.79, but extremely problematic for the GMM estimator: 36.57. This last result is worse than that for the OLS estimator. As we will see, when the true value of ρ is smaller or T is bigger, these problems are not so great but when T is very small and ρ is close to 1, the OPM estimator has the advantage of avoiding these large errors to a much greater degree.

As T increases to 3 and then 4 (Table 1, Panels B & C), the OLS fixed effects estimator continues to exhibit substantial biases and confidence intervals that are essentially useless for hypothesis testing. The OPM and GMM approaches continue to produce good estimates of ρ and β . As for the long-run effects, the OPM approach improves quickly, while the GMM approach continues to suffer from large root mean squared errors. By the time T increases to 9 (Table 1, Panel D), however, both the OPM and GMM approaches are performing well for all parameters and the OLS approach continues to perform poorly.

Table 1, Panels E & F, present the results with a reduced ρ of 0.5. With a smaller ρ , we would expect less extreme estimation problems. Unfortunately, the OLS fixed effects estimator continues to perform badly at very small values of T . The GMM and OPM estimators, on the other hand, perform very well for all parameters. The only real distinction is that the OPM estimator performs a little better, in terms of the root mean squared error, than the GMM estimator in the estimation of the long-run effect when T is at its smallest possible value of 2.

Overall, the simulations demonstrate that under the conditions considered (specifically a low T with a large N), the OLS fixed effects estimator exhibits considerable bias for all parameters. The GMM estimator is a large improvement on OLS with fixed effects but the OPM estimator consistently performs as well as or better. The superiority of the OPM results is in the estimation of the long-run effects when T is very small and ρ is close to 1, which appears to be the primary weakness of the GMM estimator.

Beyond the poor estimates for the long-run effect under specific conditions, we have also indicated that one of the downsides of GMM estimators is the need to make assumptions about the appropriateness of instrumental variables. Up to this point, we have been using lagged levels of the

⁷One should always be cautious whenever the estimate of ρ is close to 1. This suggests that either a poor estimation has occurred or that the data is (near) integrated and therefore not stationary, for the purposes of estimation.

Rho (Average)	Beta (Average)	LR Effect (Average)	LR Effect (Median)	Rho (95% CI)	Beta (95% CI)	LR Effect (95% CI)	Rho (RMSE)	Beta (RMSE)	LR Effect (RMSE)
(T=3; N=1000; Rho=0.9; Beta=0.5; LR Effect=5)									
0.93	0.51	7.74	7.63	0.06	0.62	0.17	0.03	0.01	2.95
(T=4; N=1000; Rho=0.9; Beta=0.5; LR Effect=5)									
0.93	0.51	7.45	7.36	0.02	0.48	0.04	0.03	0.01	2.56
(T=9; N=1000; Rho=0.9; Beta=0.5; LR Effect=5)									
0.91	0.5	-6.99	4.11	0.05	0.06	0.07	0.07	0.03	147.93
(T=3; N=1000; Rho=0.5; Beta=0.5; LR Effect=1)									
0.53	0.51	1.08	0.52	0.48	0.29	0.28	0.03	0.01	0.09

Table 2: Simulation Results, GMM using lagged IVs and instruments

dependent variable as instruments in the GMM estimator. An extension of this is to use the lagged differences of the dependent variable as instruments (Arellano and Bover 1995; Blundell and Bond 1998). This is possible once the researcher has a $T > 2$. This can, under the right conditions, improve the performance of the GMM estimator but it makes assumptions about the suitability of variables as instruments. If these assumptions are not met, the inclusion of these additional moment conditions may produce worse results. Table 2 reports the results of using these additional moment conditions. We see that for the data generating processes we have been using in our simulations, these additional moment conditions do indeed result in poorer estimates. The additional moment conditions result in greater bias and confidence intervals that rarely contain the true value. Clearly we do not want to include these additional moment conditions for our simulated data but the problem is that we typically do not know if it is appropriate to do so or not. The orthogonal reparameterization approach, on the other hand, does not require the researcher to make such decisions — decisions which can have a substantial impact on the validity of the estimation results. This is one further advantage of the OPM estimator.

Bibliography

- T. W. Anderson and C. Hsiao. Estimation of dynamic models with error components. *Journal of the American Statistical Association*, 76(375):598–606, 1981. [p61]
- M. Arellano. *Panel Data Econometrics*. Advanced Texts in Econometrics. Oxford University Press, Oxford, England, 2003. [p60]
- M. Arellano and S. Bond. Some tests of specification for panel data: Monte carlo evidence and an application to employment equations. *Review of Economic Studies*, 58:277–297, 1991. [p61, 71]
- M. Arellano and S. Bonhomme. Robust priors in nonlinear panel data models. *Econometrica*, 77(2): 489–536, 2009. [p63]
- M. Arellano and O. Bover. Another look at the instrumental variable estimation of error-components models. *Journal of Econometrics*, 68:29–51, 1995. [p61, 66]
- A. Behr. A comparison of dynamic panel data estimators: Monte carlo evidence and an application to the investment function. Discussion paper 05/03. Economic Research Center of the Deutsche Bundesbank., 2003. [p61]
- R. Blundell and S. Bond. Initial conditions and moment restrictions in dynamic panel data models. *Journal of Econometrics*, 87:115–143, 1998. [p66]
- R. Blundell, S. Bond, and F. Windmeijer. Estimation in dynamic panel data models: Improving on the performance of the standard gmm estimator. In B. H. Baltagi, editor, *Nonstationary Panels, Cointegrating Panels and Dynamic Panels*, pages 53–92. Elsevier, New York, 2000. [p61]
- D. R. Cox and N. Reid. Parameter orthogonality and approximate conditional inference. *Journal of the Royal Statistical Society B*, 49(1):1–39, 1987. [p63]
- Y. Croissant and G. Millo. Panel data econometrics in R: The plm package. *Journal of Statistical Software*, 27(2), 2008. URL <http://www.jstatsoft.org/v27/i02/>. [p66]
- B. E. H. é and E. Kyriazidou. Panel data discrete choice models with lagged dependent variables. *Econometrica*, 68(4):839–874, 2000. [p63]
- S. E. Finkel. Linear panel analysis. In S. Menard, editor, *Handbook of Longitudinal Research*, pages 53–92. Elsevier, New York, 2008. [p60]

- W. H. Greene. *Econometric Analysis*. Seventh. Prentice Hall, Upper Saddle River, NJ, 2012. [p61]
- C. Hsiao. *Analysis of Panel Data*. Cambridge University Press, New York, NY, 3rd edition, 2014. [p60, 61]
- C. Hsiao, M. H. Pesaran, and A. K. Tahmiscioglu. Maximum likelihood estimation of fixed effects dynamic panel data models covering short time periods. *Journal of Econometrics*, 109:107–150, 2002. [p61, 66]
- T. Lancaster. The incidental parameter problem since 1948. *Journal of Econometrics*, 95:391–413, 2000. [p60, 61, 63]
- T. Lancaster. Orthogonal parameters and panel data. *Review of Economic Studies*, 69:647–666, 2002. [p60, 61, 62]
- M. Nerlove. Experimental evidence on the estimation of dynamic economic relations from a time series of cross-sections. *Economic Studies Quarterly*, 18:42–74, 1967. [p61]
- M. Nerlove. Further evidence on the estimation of dynamic economic relations from a time series of cross-sections. *Econometrica*, 39:359–382, 1971. [p61]
- J. Neyman and E. L. Scott. Consistent estimation from partially consistent observations. *Econometrica*, 16:1–32, 1948. [p60]
- S. Nickell. Biases in dynamic models with fixed effects. *Econometrica*, 49:1417–1426, 1981. [p60, 61, 66]
- M. H. Pesaran. *Time Series and Panel Data Econometrics*. Oxford University Press, Oxford, England, 2015. [p61]
- M. Pickup. *Introduction to Time Series Analysis*. Quantitative Applications in the Social Sciences. Sage publications, Inc., Thousand Oaks, California, 2014. [p66]
- J. M. Wooldridge. *Introductory Econometrics: a Modern Approach*. Fifth. South-Western, Cengage Learning, Mason, Ohio, 2013. [p66]

Mark Pickup
Simon Fraser University
Department of Political Science
8888 University Drive
Burnaby BC V5A 1S6
Canada mark.pickup@sfu.ca

Paul Gustafson
The University of British Columbia
Department of Statistics
3182-2207 Main Mall
Vancouver BC V6T 1Z4
Canada gustaf@stat.ubc.ca

Davor Cubranic
Department of Statistics
The University of British Columbia
3182-2207 Main Mall
Vancouver BC V6T 1Z4
Canada cubranic@stat.ubc.ca

Geoffrey Evans
Nuffield College
University of Oxford
New Road, Oxford, OX1 1NF
United Kingdom geoffrey.evans@nuffield.ox.ac.uk

Appendix 1

As a second empirical example, we model the dynamics of labour demand of firm i in the United Kingdom in year t as a function of real product wages, gross capital stock and industry output. This is done using the data used by [Arellano and Bond \(1991\)](#). The variables for our second empirical example are included in the data frame 'abond_panel'. This dataframe is included with the **OrthoPanels** package.

We estimate the following:

```
> abond.opm.model <- opm(n ~ w + l_w + k + l_k + l2_k + ys + l_ys + l2_ys + yr1980
+ yr1981 + yr1982 + yr1983 + yr1984, data = abond_panel, index = c('id','year'),
n.samp = 10000, add.time.indicators = FALSE)
```

where n is the log of employment in firm i at time t ; w is the log of the real product wage; k is the log of the gross capital stock; and ys is the log of the gross industry output.

```
> summary(abond.opm.model)
Call:
opm(x = n ~ w + l_w + k + l_k + l2_k + ys + l_ys + l2_ys + yr1980 +
yr1981 + yr1982 + yr1983 + yr1984, data = abond_panel, index = c("id",
"year"), n.samp = 10000, add.time.indicators = FALSE)
```

Parameter estimates:

	<--95CI	<--68CI	med	68CI-->	95CI-->
rho	0.123000	0.1400000	0.160000	0.179000	0.198000
sig2	0.042789	0.0452270	0.047862	0.050739	0.053676
beta.w	-0.741396	-0.6200462	-0.497854	-0.376953	-0.262932
beta.l_w	-0.321563	-0.2077651	-0.088837	0.030969	0.146837
beta.k	0.452300	0.5184428	0.585200	0.655670	0.718653
beta.l_k	-0.088889	0.0042183	0.097407	0.188744	0.279613
beta.l2_k	-0.031549	0.0319986	0.097301	0.163611	0.229459
beta.ys	0.207566	0.4594758	0.714674	0.971791	1.234787
beta.l_ys	-1.055967	-0.6944647	-0.329579	0.030539	0.392799
beta.l2_ys	-0.240888	0.0202597	0.301264	0.580882	0.844360
beta.yr1980	-0.040064	-0.0050531	0.029723	0.064730	0.099077
beta.yr1981	-0.057071	-0.0135131	0.029364	0.073729	0.115364
beta.yr1982	-0.019645	0.0192744	0.063018	0.106132	0.146820
beta.yr1983	0.031298	0.0759822	0.121946	0.167622	0.211210
beta.yr1984	0.050123	0.1021425	0.156051	0.207324	0.258260

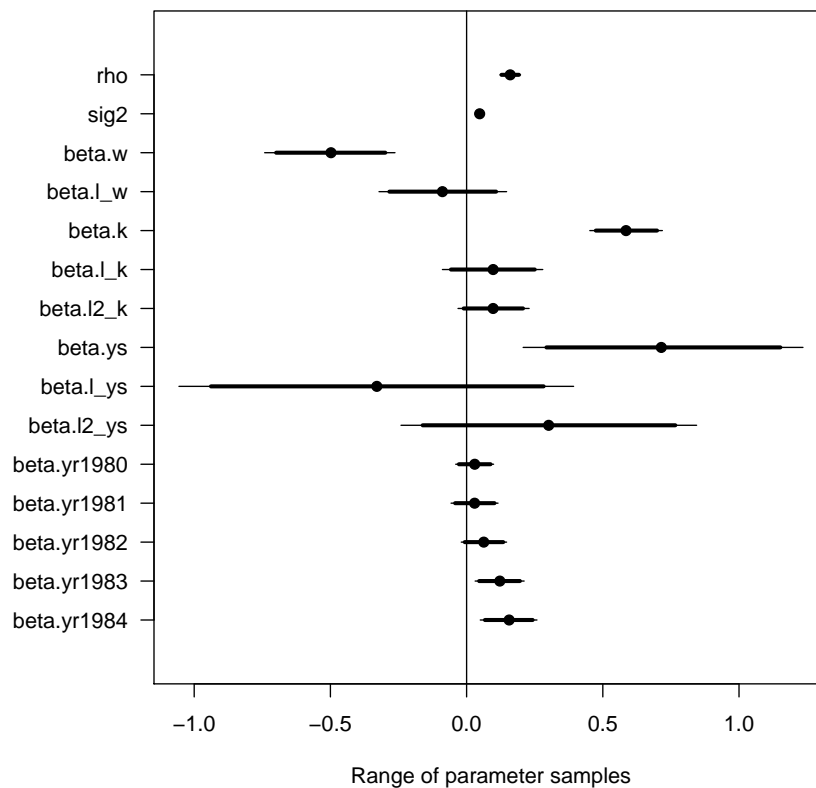
We calculate 90% equal tailed credible intervals as follows:

```
> confint(abond.opm.model, level = 0.90)
      5%      95%
rho    0.128000000 0.192000000
sig2    0.043604899 0.05267575
beta.w  -0.699216408 -0.29845898
beta.l_w -0.283471542 0.10825151
beta.k    0.474339992 0.69903010
beta.l_k -0.057501600 0.25006782
beta.l2_k -0.010780489 0.20708355
beta.ys    0.293130802 1.15175284
beta.l_ys -0.938412547 0.28266139
beta.l2_ys -0.161339807 0.76612707
beta.yr1980 -0.028266743 0.08774348
beta.yr1981 -0.042205375 0.10218775
beta.yr1982 -0.006818364 0.13445195
beta.yr1983 0.045990409 0.19575660
beta.yr1984 0.067247279 0.24236211
```

We create side-by-side plots of credible intervals of the OPM model parameters. The intervals are displayed as horizontal lines, with the 90% interval using a thicker line width and the 95% interval a thinner one. The posterior median is indicated with a dot.

```
caterplot(abond.opm.model)
abline(v=0)
```

Caterpillar plot of abond.opm.model



Appendix 2

Let's investigate the accuracy of `opm()`'s parameter estimates on 200 simulated datasets.

First, let's define the parameters used by the data-generating process:

```
rho <- .5
beta <- .5
sig2 <- 1
set.seed(321)
```

The following function generates a synthetic dataset of desired dimensions (N cases and T time points) and distribution parameters ($(\rho = \text{rho})$, $(\beta = \text{beta})$, and $(\sigma^2 = \text{sig2})$):

```
generate <- function(N, T, rho, beta, sig2) {
  LT <- T + 50
  f <- runif(N, -1, 1)
  x <- array(.75 * f, dim = c(N, LT)) + rnorm(N * LT, sd = 4)
  y <- matrix(0, N, LT)
  for (t in 1:LT) {
    yy <- if (t > 1)
      y[, t - 1]
    else
      ((f + beta * .75 * f)/(1 - rho))
    y[, t] <- rho * yy + f + x[, t] * beta +
      rnorm(N, sd = sqrt(sig2))
  }
  data.frame(i = rep(seq(N), LT - 50),
            t = rep(seq(LT - 50), each = N),
```



```

      x1 = c(x[(50 * N + 1):(LT * N)]),
      y = c(y[(50 * N + 1):(LT * N)])
}

```

Now we generate 200 datasets with $N = 1000$ cases and $T = 3$ time points (after a 50 wave burn-in):

```

N <- 1000
T <- 3
reps <- 200

ds <- replicate(n = reps,
               generate(N = N, T = T, rho = rho, beta = beta, sig2 = sig2),
               simplify = FALSE)

```

Now we fit the OPM model to the datasets and save the results:

```

library(OrthoPanels)
library(knitr)

set.seed(421)
opms <- lapply(ds, function(d) {
  opm(y ~ x1, data = d, n.samp = 1000)
})

```

Let's check the sampled parameters:

```

true_param <- c(rho = rho, sig2 = sig2, beta = beta)
est_param <- sapply(opms, coef)
resid <- sweep(est_param, 1, true_param)
rmse <- sqrt(rowMeans(resid ^ 2))
kable(rbind(`True` = true_param,
            `Est` = rowMeans(est_param),
            `Bias` = rowMeans(resid),
            `RMSE` = rmse))

```

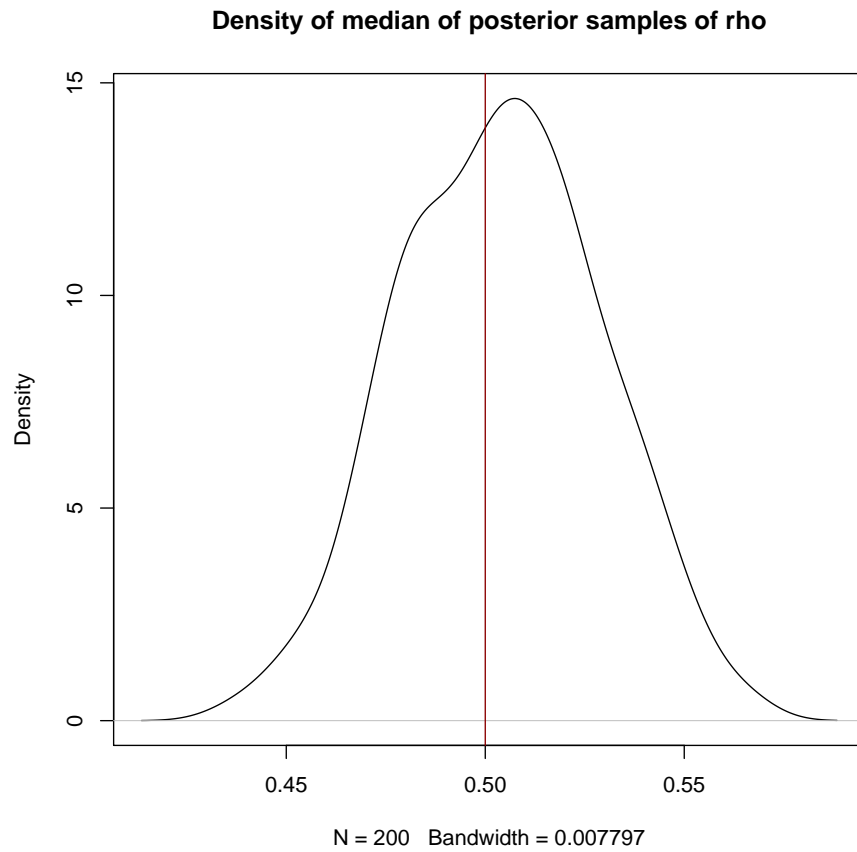
	rho	sig2	beta
True	0.5000000	1.0000000	0.5000000
Est	0.5038800	1.0050657	0.5018072
Bias	0.0038800	0.0050657	0.0018072
RMSE	0.0252336	0.0517390	0.0106620

Density plot for each parameter, with true value marked with a vertical line:

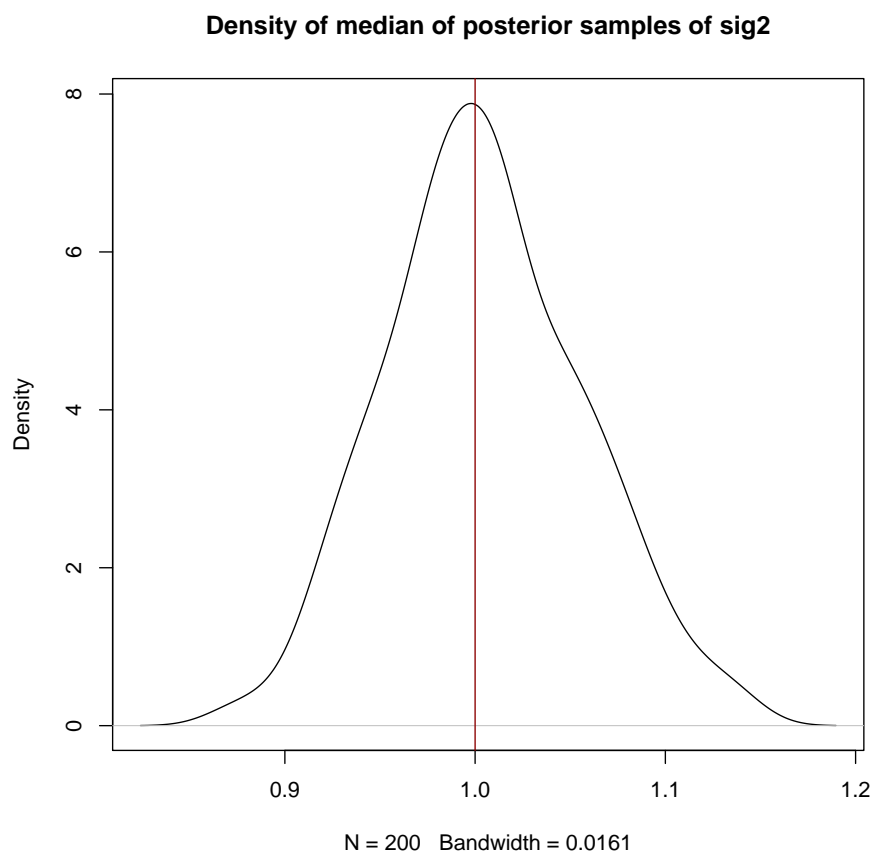
```

plot(density(sapply(opms, coef)[1,]),
     main = 'Density of median of posterior samples of rho')
abline(v = rho, col = 'darkred')

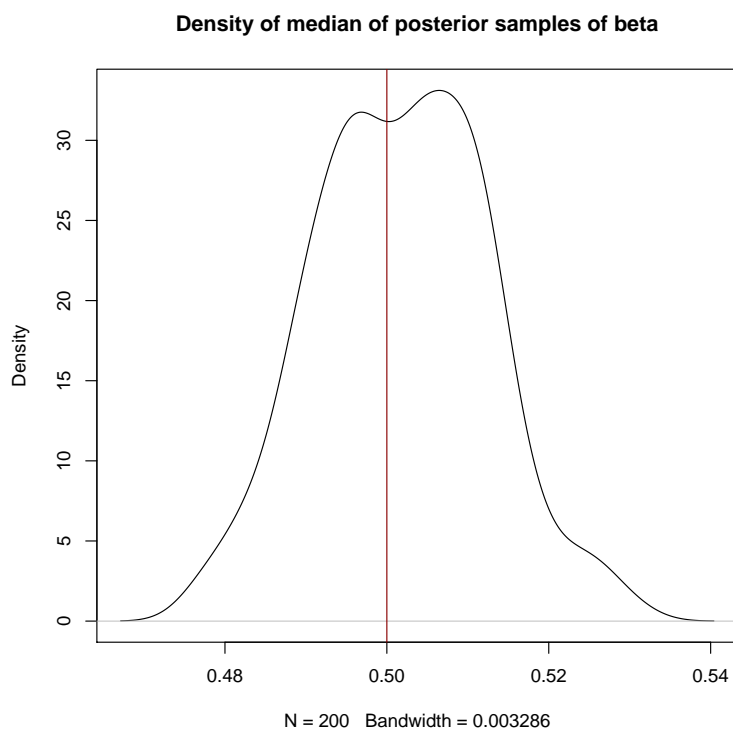
```



```
plot(density(sapply(opms, coef)[2,]),  
      main = 'Density of median of posterior samples of sig2')  
abline(v = sig2, col = 'darkred')
```



```
plot(density(sapply(opms, coef)[3,]),  
     main = 'Density of median of posterior samples of beta')  
abline(v = beta, col = 'darkred')
```



The proportion of time the 95% credible interval includes the true value of the parameter:

```
cis <- sapply(lapply(opms, confint),
             function(ci) {
               ci[, '2.5%'] <= c(rho, sig2, beta) &
               ci[, '97.5%'] >= c(rho, sig2, beta)
             })
rowSums(cis) / reps
```

rho	sig2	beta
0.930	0.955	0.925

The mosaic Package: Helping Students to ‘Think with Data’ Using R

by Randall Pruim, Daniel T Kaplan, Nicholas J Horton

Abstract The mosaic package provides a simplified and systematic introduction to the core functionality related to descriptive statistics, visualization, modeling, and simulation-based inference required in first and second courses in statistics. This introduction to the package describes some of the guiding principles behind the design of the package and provides illustrative examples of several of the most important functions it implements. These can be combined to help students “think with data” using R in their early course work, starting with simple, yet powerful, declarative commands.

Motivation

In order to make sense of rich data that is increasingly available, students need computational tools and facility for data management, exploratory analysis, visualization, and modeling (e.g., Nolan and Lang (2010); Horton et al. (2015); Horton and Hardin (2015); Ridgway (2016)). To extract useful information from the complex systems that generate this rich data, students should learn to “think with data” (in the phrase coined by Diane Lambert of Google): using previous results from data to drive statistical investigations and to inform the choice of analysis and presentation possibilities.

Yet many students enter statistics courses with little or no computational experience. Software such as R that encompass the tools for thinking with data are sometimes regarded as off-putting and inaccessible to students. With the **mosaic** package, we have sought to remove unnecessary difficulty in the use of R by students. Our students have demonstrated that it is feasible to integrate computing into our curricula early and often, in a way that provides students with success, confidence, and room to grow.

A guiding principle: Less volume, more creativity

The **mosaic** (Pruim et al., 2016b) package reflects attempts by each of the authors to make the power of R accessible and rewarding to students, especially in the context of undergraduate statistics courses, and, in one case, also in calculus. One of the guiding principles behind the development of the **mosaic** package has been “Less volume, more creativity.” By “less volume,” we mean reducing the cognitive load involved in using R. By “more creativity” we mean two things. First, we want to provide access to R tools in a way that fosters choices and decision making by students. Such decisions might be about modes of analysis or visualization or about the variables and covariates to consider when exploring relationships with data. Second, we want to encourage students to creatively engage with statistical inference methods via simulation techniques such as randomization and resampling with data analysis and presentation.

Our route to “less volume” is to provide a set of three command templates — formulas, functions, and extractors — that standardize usage across many tasks and that highlight the connections between graphical summaries, numerical summaries, models, and inference. The templates themselves are designed to be consistent and concise. Consistency enables a student to generalize from a specific task to a wide set of possibilities. Conciseness means avoiding unnecessary elements so that the essential inputs to a computation are made clear and organized according to a syntax that makes the role of each input understandable and predictable.

The importance of multivariate thinking

The importance of giving students experience with multivariable thinking is a point of emphasis in the newly revised ASA Guidelines for assessment and instruction in statistics education (GAISE) report (ASA GAISE College working group et al., 2016):

When students leave an introductory course, they will likely encounter situations within their own fields of study in which multiple variables relate to one another in intricate ways. We should prepare our students for challenging questions that require investigating and exploring relationships among more than two variables. (page 16)

Perhaps the best place to start is to consider how a third variable can change our understanding of the relationship between two variables. . . . Simple approaches (such as

stratification) can help to discern the true associations. Stratification requires no advanced methods, nor even any inference, though some instructors may incorporate other related concepts and approaches such as multiple regression. These examples can help to introduce students to techniques for assessing relationships between more than two variables. Including one or more multivariable examples early in an introductory statistics course may help to prepare students to deal with more than one or two variables at a time. . . (page 34)

The **mosaic** package helps support these goals by providing a unified framework within which multivariable graphical and numerical summaries are easy to create, even for beginners.

The formula template

Our most important template makes use of a “formula interface” that has long been used by familiar R functions like `t.test()`, `lm()`, and the plotting functions in **lattice** (Sarkar, 2008). Our initial example uses the `Births78` data set, which records the number of live births in the United States for each day of 1978.

```
library(mosaic)
head(Births78, 3)

##      date births dayofyear wday
## 1 1978-01-01  7701         1 Sun
## 2 1978-01-02  7527         2 Mon
## 3 1978-01-03  8825         3 Tues
```

We will use this example to illustrate the difference between using a consistent interface across graphical and numerical summaries and the limitations of basic R functions that do *not* use formulas.

We typically introduce the formula template in the context of exploring the relationship between two variables, e.g.

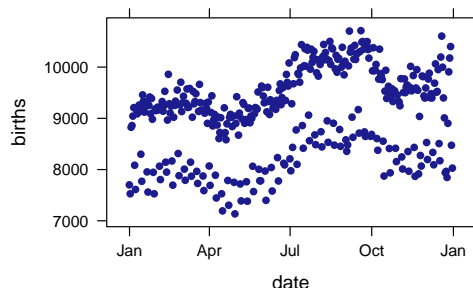
```
goal( y ~ x, data = MyData, ... ) # pseudo-code for the formula template
```

We teach students to read $y \sim x$ as “y wiggle x” and to interpret this in any of several essentially equivalent forms: “y broken down by x”; “y modeled by x”; “y explained by x”; “y depends on x”; or “y accounted for by x.” For graphics, it’s reasonable to read the formula as “y vs. x”, which is exactly the convention used for coordinate axes. But “y vs. x” does not as clearly convey the asymmetry of the other forms.

This template is not original to **mosaic**. Those familiar with R will recognize this as the template already used by functions such as `lm()` and the **lattice** plotting functions. The **mosaic** package extends this template to numerical summaries and provides some additional features for plotting and fitting models, thereby bringing all of these activities into a consistent, unified approach.

Our first plot of the `Births78` data might be a scatterplot showing how the number of births depends on the date. Using our template and the **lattice** function `xypplot()`, we can create this with the following simple command by filling in the slots in our formula template.

```
library(lattice)
xypplot(births ~ date, data = Births78)
```

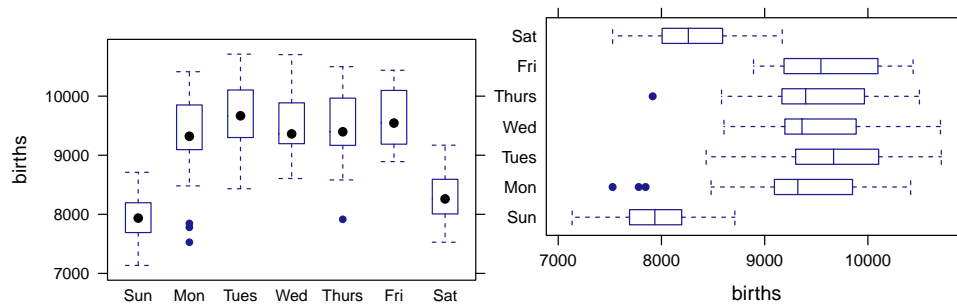


The scatterplot reveals some interesting patterns over the course of the year. Getting students to make conjectures about possible explanations can help diagnose how well students understand the information displayed in the scatterplot. (Students who suggest that there are two parallel waves

because more children are born at certain times of the year reveal that they are misunderstanding this plot, for example.)

One reasonable conjecture for the two parallel waves is a weekend effect. We would like our students to be able to explore this conjecture graphically, numerically, and (eventually) with statistical models. We might begin by creating side-by-side boxplots. The **lattice** package makes this as easy as the scatterplot above. We simply replace `xypplot()` with `bwplot()` because we have a different goal. If we prefer, we can also reverse the order of the variables in the formula to flip which axis is used for which purpose.

```
bwplot(births ~ wday, data = Births78)
bwplot(wday ~ births, data = Births78, pch = "|") # a more common way to show median
```



To quantify the differences observed in these plots, we might like to compute the mean (or median) number of births for each weekday. Basic statistical calculations can be admirably concise in R. For instance, the mean number of daily births (over all days) can be calculated with

```
mean(Births78$births) # or use with() instead of $
## [1] 9132
```

Unfortunately, this is a dead end when it comes to thinking with data. How, for instance, does one explore whether there is a day-of-the-week component to the number of births? Here are a few conventional approaches in R to such a calculation.

```
aggregate(Births78$births, FUN = mean, by = list(Births78$wday))
```

or, equivalently,

```
with(Births78, aggregate(births, FUN = mean, by = list(wday)))
```

```
## Group.1 x
## 1 Sun 7951
## 2 Mon 9371
## 3 Tues 9709
## 4 Wed 9498
## 5 Thurs 9484
## 6 Fri 9626
## 7 Sat 8309
```

or, alternatively,

```
with(Births78, tapply(births, wday, mean))
## Sun Mon Tues Wed Thurs Fri Sat
## 7951 9371 9709 9498 9484 9626 8309
```

None of these show any similarity to `mean(Births78$births)` or to the commands that created the plots that generated our conjecture. All of them bury `mean()` in the center or at the end of the command and require additional structure like `aggregate()`, `tapply()`, lists, and nested parentheses. Even reading the commands is difficult.

The essential elements in each of these calculations are `mean()`, `Births78`, `births` and `wday`: We want to see how the mean number of births depends on the day of the week using data from 1978. The **mosaic** formula template for this calculation mirrors the template used to create plots in **lattice** and highlights these essential elements:

```
library(mosaic)
mean(births ~ wday, data = Births78)

## Sun Mon Tues Wed Thurs Fri Sat
## 7951 9371 9709 9498 9484 9626 8309
```

This form makes clear that the `mean()` is being calculated and that `Births78` holds the data. The relationship between the variables is specified by the formula. Most importantly, it is the same template that was used to create the plots that preceded it – we simply replace `bwplot()` or `xypplot()` with `mean()`.

Using the **mosaic** package, this same template extends to many other numerical summaries, e.g.,

```
median(births ~ wday, data = Births78)

## Sun Mon Tues Wed Thurs Fri Sat
## 7936 9321 9668 9362 9397 9544 8260
```

```
sd(births ~ wday, data = Births78)

## Sun Mon Tues Wed Thurs Fri Sat
## 410 608 527 461 551 488 390
```

```
favstats(births ~ wday, data = Births78)

## wday min Q1 median Q3 max mean sd n missing
## 1 Sun 7135 7691 7936 8196 8711 7951 410 53 0
## 2 Mon 7527 9097 9321 9838 10414 9371 608 52 0
## 3 Tues 8433 9304 9668 10084 10711 9709 527 52 0
## 4 Wed 8606 9196 9362 9880 10703 9498 461 52 0
## 5 Thurs 7915 9171 9397 9958 10499 9484 551 52 0
## 6 Fri 8892 9198 9544 10088 10438 9626 488 52 0
## 7 Sat 7527 8007 8260 8586 9170 8309 390 52 0
```

The **mosaic** package provides formula interfaces for `mean()`, `median()`, `sd()`, `var()`, `cor()`, `cov()`, `quantile()`, `max()`, `min()`, `range()`, `IQR()`, `iqr()`, `fivenum()`, `prod()`, and `sum()`. In each case we have been careful not to break behavior of the underlying functions from **base** and **stats**.

Model-building functions such as `lm()` and `glm()` also employ the same template:

```
births.model <- lm(births ~ wday, data = Births78)
```

so early experience with graphical and numerical summaries prepares students for statistical modeling later in the course.

When working with categorical data, tabulation is an important technique. The `table()` function does not accept formulas and `xtabs()` uses formulas in a different way. The **mosaic** package provides a formula-template `tally()` function for counting categorical variables. We illustrate its use with another data set from **mosaicData** (Pruim et al.). Whickham contains data from a UK study that enrolled subjects in 1972–74 and conducted a follow-up 20 years later.

```
tally(outcome ~ smoker, data = Whickham)

##      smoker
## outcome No Yes
## Alive 502 443
## Dead 230 139

tally(outcome ~ smoker, data = Whickham, margins = TRUE)

##      smoker
## outcome No Yes
## Alive 502 443
## Dead 230 139
## Total 732 582

tally(outcome ~ smoker, data = Whickham, margins = TRUE, format = "proportion")
```



```
##      smoker
## outcome  No  Yes
##  Alive 0.686 0.761
##   Dead 0.314 0.239
##   Total 1.000 1.000
```

Notice that in the final example, conditional proportions are calculated. The conditional probabilities were computed in a way that respects the asymmetric meaning of the formula `outcome ~ smoker`: using `smoker` to account for `outcome`. The probabilities indicate that smokers were more likely to be alive in the follow-up study. More on this in a moment.

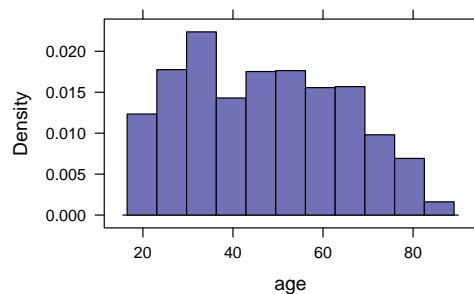
The formula template can be extended to handle one variable or more than two variables, but for several reasons we recommend introducing it in the context of two-variable plots and numerical summaries. We offer this recommendation because (1) two-variable plots and numerical summaries are more “impressive” than one-variable plots and less likely to be something students can as readily do with tools they already know, (2) working with more than one variable from the start (correctly) suggests that the most interesting parts of statistics involve more than one variable (Wild et al., 2011), and (3) the formula syntax for a single variable makes more sense in the context of two-sided formulas than it does in isolation.

Formulas with a single variable correspond to situations where there is no “explanatory” variable to be included, for example in simple numerical summaries or depictions of the distribution of a variable.

```
mean( ~ age, data = Whickham)

## [1] 46.9

histogram( ~ age, data = Whickham)
```



To some instructors, it seems more natural when there is no explanatory variable for the single variable to be on the left-hand side of `~`. But the R parser does not support this: a formula must have a right-hand side. Even if R allowed such formulas, the use of `y ~` would break the analogy to graphical summaries where a single variable is traditionally displayed on the x -axis, as in the histogram above.

The numerical summary functions provided by **mosaic** also allow formulas like `age ~ NULL` to signify there is no explanatory variable.

```
mean( age ~ NULL, data = Whickham)

## [1] 46.9
```

We don’t emphasize this form, however, since it is not supported by the **lattice** functions.

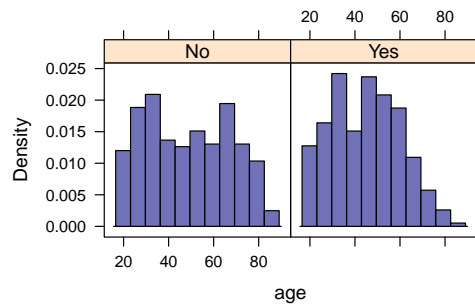
“Thinking with data” implies the ability to examine relationships among multiple variables. The formula template accommodates more than one variable on the right-hand side of the tilde, for instance:

```
mean(age ~ smoker + outcome, data = Whickham)

## No.Alive Yes.Alive No.Death Yes.Death
## 40.0 40.1 67.6 59.2
```

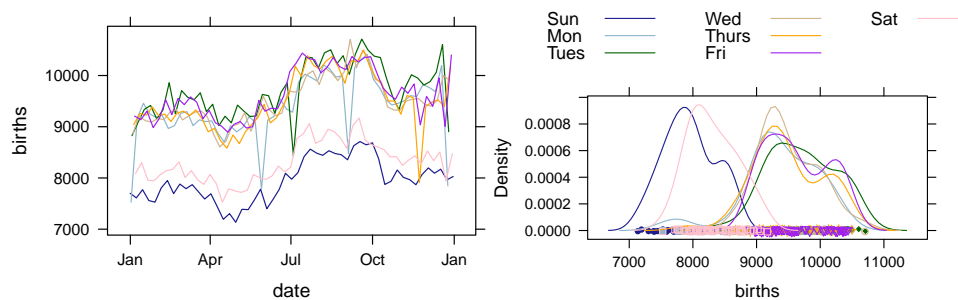
The **lattice** package provides a number of ways to include additional variables in plots. Inclusion of `|` in a formula, for example, specifies a variable to be used for defining subpanels (or facets).

```
histogram( ~ age | smoker, data = Whickham)
```



Another option is to overlay multiple layers in a plot; **lattice** does this via a groups argument.

```
xplot(births ~ date, groups = wday, data = Births78, type = "l")
densityplot(~ births, groups = wday, data = Births78, auto.key = list(columns = 3))
```



Each of these two plots shows a clear difference in the distribution of births on the two weekend days compared to the other five days of the week.

For ease in moving between plots and numerical summaries, **mosaic** functions such as `mean()` and `tally()` also accept this expanded syntax.

```
mean(~ age | smoker, data = Whickham)

## No Yes
## 48.7 44.7

mean(~ births, groups = wday, data = Births78)

## Sun Mon Tues Wed Thurs Fri Sat
## 7951 9371 9709 9498 9484 9626 8309
```

Notice that the mean age for smokers in the Whickham study is substantially lower than for the non-smokers, so an accurate comparison of survival rates must take age into account.

The formula template allows students to think about relationships between and among two or more variables and to test conjectures using graphical and numerical summaries. Having learned the formula interface to graphical and numerical summaries early on, new users are well prepared for modeling with `lm()`, `glm()`, and various “test” functions such as `t.test()` when the time comes. More importantly, they begin early to train their minds to ask questions of the form “How does this depend on that (and some other things)?”.

By emphasizing the formula template, each of the following commands can be viewed as instances of a common template, rather than as separate things to learn.

```
bwplot(age ~ smoker, data = Whickham) # standard function in lattice
mean(age ~ smoker, data = Whickham) # formula interface added in mosaic
sd(age ~ smoker, data = Whickham) # formula interface added in mosaic
lm(age ~ smoker, data = Whickham) # standard function in stats
```

Similarly, by adding additional formula interfaces to `t.test()`, `binom.test()`, and `prop.test()`, and adding some additional plot types, for one-variable situations we have

```
mean(~ age, data = Whickham) # formula interface added in mosaic
sd(~ age, data = Whickham) # formula interface added in mosaic
favstats(~ age, data = Whickham) # new function in mosaic
histogram(~ age, data = Whickham) # standard function in lattice
```

```
t.test( ~ age, data = Whickham)      # formula interface added in mosaic
binom.test( ~ smoker, data = Whickham) # formula interface added in mosaic
prop.test( ~ smoker, data = Whickham) # formula interface added in mosaic
```

Adding covariates to one- or two- variable graphical or numerical summaries fits readily into the template as well.

```
mean( ~ age | smoker, data = Whickham) # formula interface added in mosaic
sd( ~ age | smoker, data = Whickham) # formula interface added in mosaic
histogram( ~ age | smoker, data = Whickham) # standard lattice
t.test( ~ age | smoker, data = Whickham) # expanded formula interface in mosaic
```

While specifying the correct formula can produce some challenges for new users, clearly explaining the roles of each component for plotting, for numerical summaries, and for model fitting helps demystify the situation. Instructors have had students create and interpret bivariate and trivariate graphical displays on the first day of class (Wang et al., 2017). We have also found that explicit, early, low-stakes assessment of student mastery of the formula interface greatly improves student performance. A first quiz consisting of a single item (What is the formula template?) followed by one or two simple pencil-and-paper quizzes asking students to write the commands to recreate a handful of numerical and graphical summaries suffices.

The model-function template

Modeling functions like `lm()` and `glm()` can fit a wide range of statistical models. But functions like `predict()` are challenging for new users (primarily because of the user must create a data frame in order to evaluate the model function on user-specified inputs), and constructing a useful graphical representation of a data set together with a logistic regression fit even more so. The **mosaic** functions `makeFun()`, `plotFun()`, and `plotModel()` make these tasks easier.

In particular, `makeFun()` extracts from a model object created by `lm()` or `glm()` a function that is a wrapper around `predict()` and can be used with standard function syntax that is familiar to students from the way functions are typically described in secondary school and in calculus. This wrapper around `predict()` is easier for beginners to use because (1) it returns a function to which inputs can be supplied without creating a data frame, (2) the resulting function returns values on the response scale by default, and (3) it back transforms a few common transformations of the response variable, including `log()` and `sqrt()` (and allows the user to provide a custom value to the transformation argument to handle other cases).

The functions extracted from models using `makeFun()` (and functions created in other ways) can be plotted with `plotFun()`. In the example below, we illustrate the use of `makeFun()` and `plotFun()` to compare linear and quadratic fits to the same data.

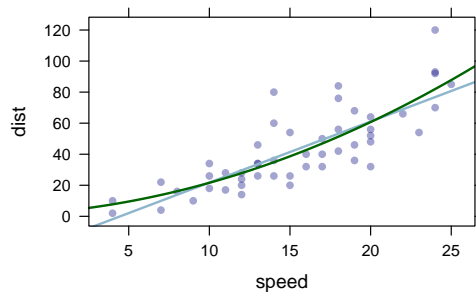
```
cars.mod1 <- lm(dist ~ speed, data = cars)
cars.mod2 <- lm(dist ~ poly(speed,2), data = cars)
# extract functions for each model that compute distance from speed
cars.dist1 <- makeFun(cars.mod1)
cars.dist2 <- makeFun(cars.mod2)
# evaluate these functions as user-specified values of speed
cars.dist2(speed = 15)

##      1
## 38.7

cars.dist2(speed = 15, interval = "confidence")

##      fit lwr upr
## 1 38.7  33 44.3

# add plots of these functions to a scatter plot
xyplot(dist ~ speed, data = cars, alpha = 0.4)
plotFun(cars.dist1(speed) ~ speed, add = TRUE, col = 2, lwd = 2)
plotFun(cars.dist2(speed) ~ speed, add = TRUE, col = 3, lwd = 2)
```



For logistic regression, when the response is coded as a factor, we need to adjust things slightly when plotting because the model function returns values between 0 and 1, but 2-level factors are coded as 1 and 2 in R.

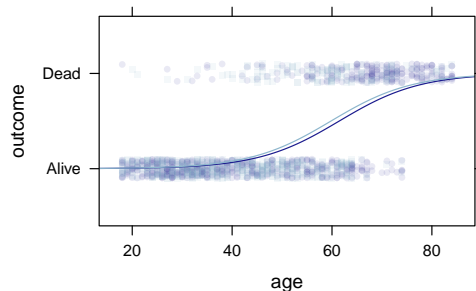
```
smoker.mod <- glm(outcome ~ smoker + age, data = Whickham, family = binomial)
smoker.fun <- makeFun(smoker.mod)
smoker.fun(age = 60, smoker = "Yes")

##      1
## 0.507

smoker.fun(age = 60, smoker = "No")

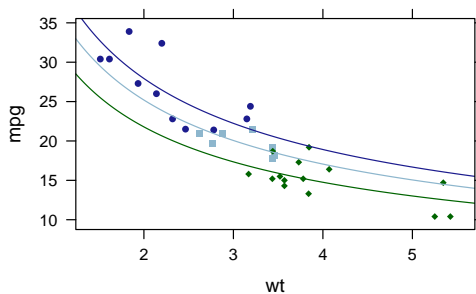
##      1
## 0.456

xyplot(outcome ~ age, groups = smoker, data = Whickham, jitter.y = TRUE, alpha = 0.1)
plotFun(1 + smoker.fun(age, smoker = "No") ~ age, col = 1, add = TRUE)
plotFun(1 + smoker.fun(age, smoker = "Yes") ~ age, col = 2, add = TRUE)
```



When the response variable in the model is transformed, the transformation argument to `makeFun()` can be used to specify the back-transformation. For a few common transformations (e.g., `log()` and `sqrt()`), the value of transformation is determined automatically (by default).

```
mtcars.mod <- lm(log(mpg) ~ log(wt) + factor(cyl), data = mtcars)
mileage <- makeFun(mtcars.mod)
xyplot(mpg ~ wt, data = mtcars, groups = cyl)
plotFun(mileage(wt, cyl = 4) ~ wt, add = TRUE, col = 1)
plotFun(mileage(wt, cyl = 6) ~ wt, add = TRUE, col = 2)
plotFun(mileage(wt, cyl = 8) ~ wt, add = TRUE, col = 3)
```

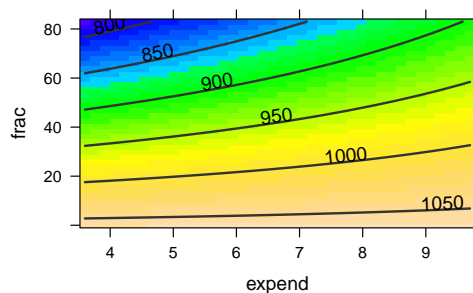


For models with two quantitative predictors, `plotFun()` can create a contour plot. The following model explores how the average SAT score depends on the amount of money spent on education (in thousands of dollars per student) by a US State and the percent of students in that state who take the SAT exam.

```
SAT.mod <- lm(sat ~ expend * frac, data = SAT)
msummary(SAT.mod)

##           Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1057.121    42.040   25.15 < 2e-16 ***
## expend      0.629      7.846    0.08  0.936
## frac       -4.232      0.818   -5.18 4.9e-06 ***
## expend:frac  0.237      0.135    1.75  0.087 .
##
## Residual standard error: 31.8 on 46 degrees of freedom
## Multiple R-squared:  0.831, Adjusted R-squared:  0.82
## F-statistic: 75.2 on 3 and 46 DF, p-value: <2e-16

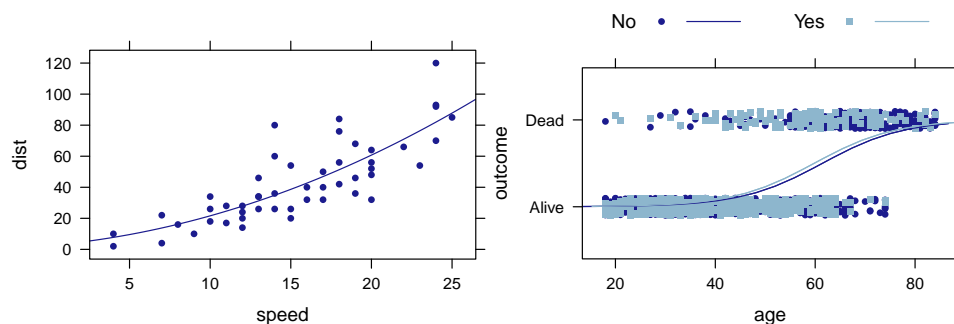
sat.pred <- makeFun(SAT.mod)
plotFun(sat.pred(expend, frac) ~ expend + frac,
        expend.lim=c(3.6,9.7), frac.lim=c(0,83))
```



The `msummary()` function provides a terser summary of the model object than `summary()`. The plot shows that this model predicts performance on the exam to increase with increased expenditure and decreased participation in the exam. (In states with lower participation rates there is a selection bias toward stronger students who are seeking admission into out-of-state colleges and universities.) Furthermore, the effect of increased spending appears to be greater when a larger percent of the students take the exam. Interestingly, a simpler linear model that includes only expenditure as a predictor has a negative slope.

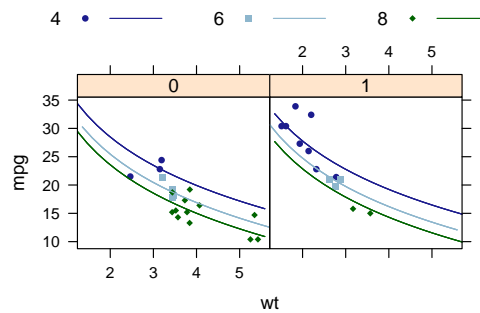
For many simple models, creating a plot can be even simpler using `plotModel()`, which also eliminates the need to manually adjust logistic regression plots when the response is a factor. For models with multiple predictors, we can supply a formula to indicate which predictor we prefer to have on the x-axis.

```
plotModel(cars.mod2)
plotModel(smoker.mod, outcome ~ age, jitter.y = TRUE)
```



The `plotModel()` function can also simplify visualization of more complex models.

```
mtcars.mod2 <- lm(mpg ~ log(wt) + factor(cyl) + factor(am), data = mtcars)
plotModel(mtcars.mod2, mpg ~ wt | factor(am))
```



The extractor template

The use of `makeFun()` to create a function from a model illustrates another important template, the extractor template:

```
object <- { some computation }
extractor(object)           # extract some information from object
```

R has many such extractors which summarize, display, or extract partial information from an object. The `print()`, `plot()`, and `summary()` functions are examples of extractors that can be applied to many types of objects. Other extractors, like `coef()`, `resid()`, and `fitted()` are designed to work with a much smaller set of objects. The **mosaic** package defines several extractors including those listed in Table 1.

```
confint(t.test( ~ age, data = Whickham))      # works for any "htest" object

## mean of x lower upper level
## 1      46.9      46      47.9      0.95

pval(t.test(age ~ smoker, data = Whickham))   # works for any "htest" object

## p.value
## 2.06e-05

stat(t.test(age ~ smoker, data = Whickham))   # works for any "htest" object

## t
## 4.27

rsquared(lm(age ~ smoker, data = Whickham))

## [1] 0.0131
```

The `mplot()` function

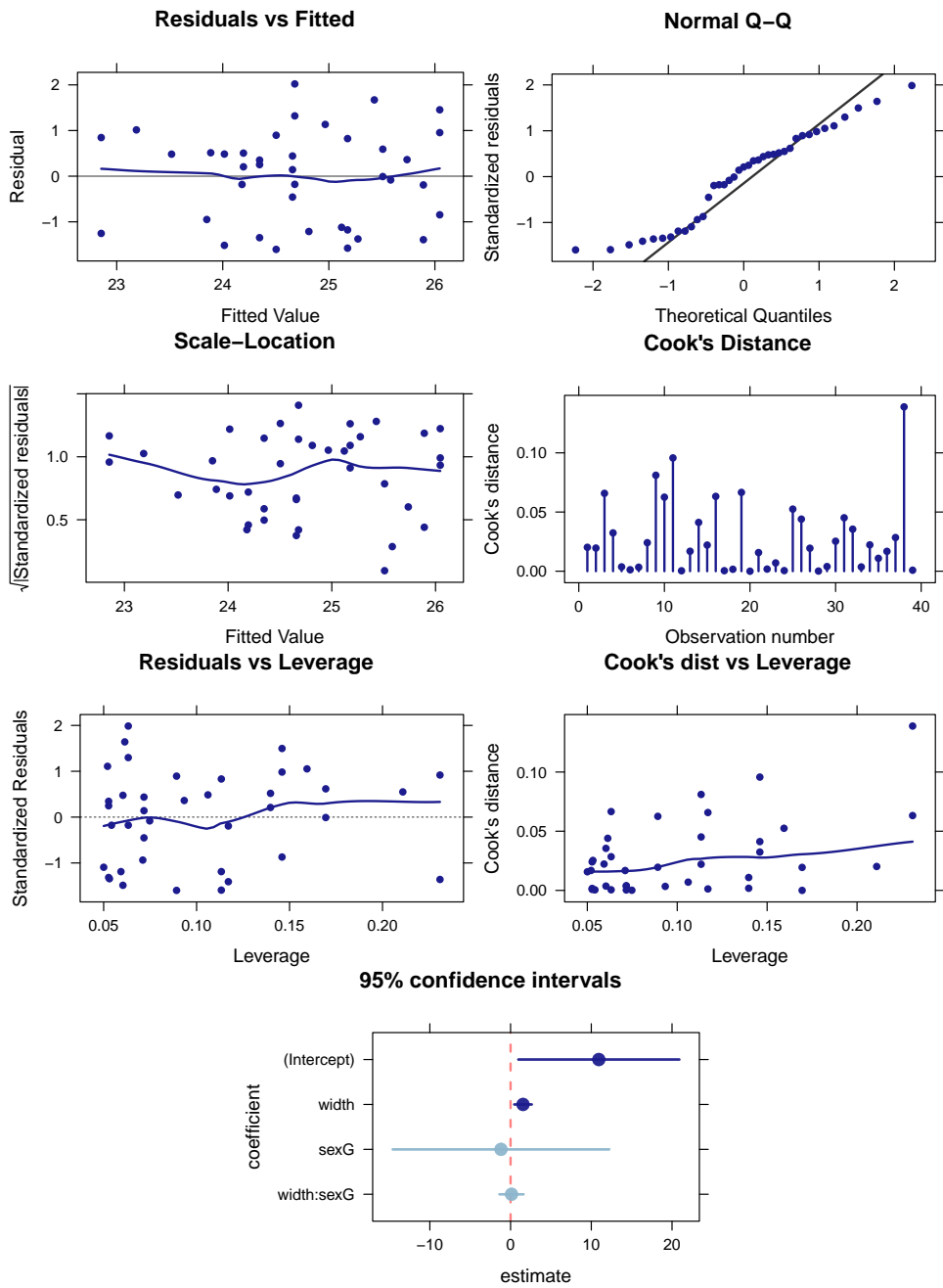
Like `plot()`, `mplot()` has many uses depending on the kind of input it receives. The two primary uses cases are creating diagnostic plots for `lm` and `glm` objects, and interactively creating data visualizations using the variables in a data frame.

Given a model object as its first argument, `mplot()` provides similar diagnostic plots to those produced via `plot()` but with two primary differences: the user may select to use either **lattice** or **ggplot2** (Wickham, 2009) graphics instead of base graphics, and an additional plot type is provided to visualize the confidence intervals for the coefficients of a regression model.

Extractor	Purpose
<code>makeFun()</code>	Extract a fitted function from a model.
<code>rsquared()</code>	Extract r^2 from a linear model.
<code>stat()</code>	Extract the test statistic from a hypothesis test.
<code>pval()</code>	Extract the p-value from a hypothesis test.
<code>confint()</code>	Extract the confidence interval from a hypothesis test.
<code>mplot()</code>	Create a plot from an object.

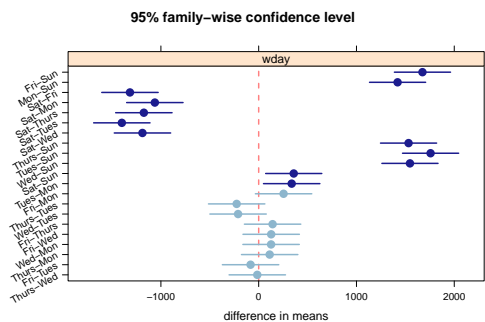
Table 1: Extractors defined in the **mosaic** package.

```
mod <- lm(length ~ width * sex, data = KidsFeet)  
mplot(mod, system = "lattice", which = 1:7)
```



We can also use `mplot()` to visually represent the results of `TukeyHSD()`, which has been modified so that it can be applied directly to objects produced by `lm()`.

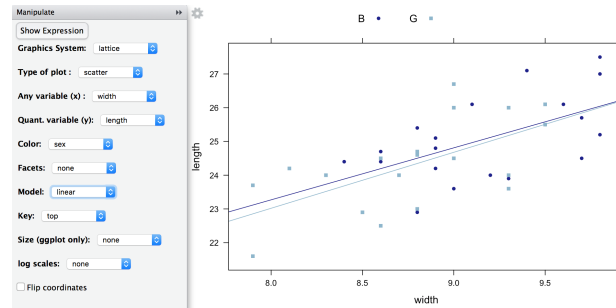
```
mplot(TukeyHSD(lm(births ~ wday, data = Births78)), order = "pval")
```



The resulting plots are formatted in a way that makes them usable in a wider range of scenarios than are those produced using `plot()`.

A second use for `mplot()` is to create **lattice** and **ggplot2** plots interactively within RStudio. Issuing the following command in RStudio will bring up a plot that can be modified by making choices in the accompanying menu.

```
mplot(KidsFeet)
```



The menu allows the user to choose either **lattice** or **ggplot2** graphics, to select the type of plot and the variables used, and to control a few of the most commonly used features that modify a plot (faceting, color, legends, log-scaling, fitting a linear model or LOESS smoother). The “Show Expression” button exports the command used to create the plot into the console. From there it can be edited or copied and pasted into an R Markdown document. This can be very useful for new users working to master the syntax for a particular graphical system.

Randomization and resampling

Resampling approaches have become increasingly important in statistical education (Tintle et al., 2015; Hesterberg, 2015). The **mosaic** package provides simplified functionality to support teaching inference based on randomization tests and bootstrap methods. Our goal was to focus attention on the important parts of these techniques (e.g., where randomness enters in and how to use the resulting distribution) while hiding some of the technical details involved in creating loops and accumulating values.

A first example

As a first example, we often introduce (a version of) the story of the lady tasting tea. (See Salsburg (2002) for the details of this famous story.) But here we will test a coin to see whether it is a “fair coin”. Suppose we flip the coin 20 times and observe only 6 heads, how suspicious should we be that the coin is not fair? The statistical punchline for either the lady tasting tea or testing a coin is that we want to compute the p-value for a binomial test via simulations rather than using formulas for the binomial distribution or normal approximations. But we want to do this on the first day of class, and without using any of the jargon of the preceding sentence.

Because students do not know about sampling distributions or random variables yet, but do understand the idea of a coin toss, we have provided `rflip()` to simulate tossing a coin one or several times:

```
rflip()

##
## Flipping 1 coin [ Prob(Heads) = 0.5 ] ...
##
## H
##
## Number of Heads: 1 [Proportion Heads: 1]

rflip(20)

##
## Flipping 20 coins [ Prob(Heads) = 0.5 ] ...
##
```



```
## H T T T H T H H H H H T T H H T T H H T
##
## Number of Heads: 11 [Proportion Heads: 0.55]
```

To test a null hypothesis of a fair coin, we need to simulate flipping 20 coins many times, recording for each simulation the number of heads that were observed. The `do()` function allows us to do just that using the following template

```
do(n) * {stuff to do}          # pseudo-code
```

where `{stuff to do}` is typically a single R command, but may be something more complicated. We teach this syntax by reading it aloud: “Do n times ...” For example, we can flip 20 coins three times as follows.

```
do(3) * rflip(20)  # do 3 times flip 20 coins

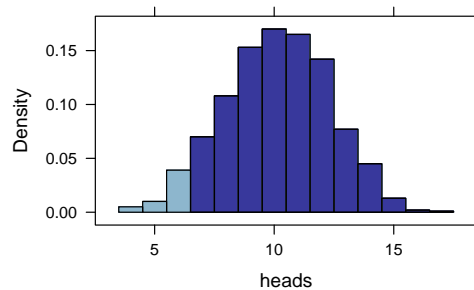
##   n heads tails prop
## 1 20   10   10 0.50
## 2 20    8   12 0.40
## 3 20   15    5 0.75
```

Notice that `do()` (technically `cull_for_do()`) has been clever about what information is stored for each group of 20 coin tosses and that the results are returned in a data frame.

It is now a simple matter to do this many more times and use numerical or graphical summaries to investigate how unusual it is to get so few heads if the coin is indeed a fair coin.

```
Sims <- do (1000) * rflip(20)
histogram( ~ heads, data = Sims, width = 1, groups = heads <= 6)
tally ( ~ (heads <= 6), data = Sims)

## (heads <= 6)
## TRUE FALSE
##    54   946
```



Readers familiar with **lattice** will notice that the **mosaic** package adds some additional arguments to the `histogram()` function. Among these are `width` and `center` which can be used to control the width and position of the bins and are much easier for new users to master than the `breaks` argument supplied by **lattice**.

The `sample()`, `resample()`, and `shuffle()` functions

To facilitate randomization and bootstrapping, **mosaic** extends `sample()` to operate on data frames. The `shuffle()` function is an alternative name for `sample()`, and `resample()` is `sample()` with `replace = TRUE`. With these in hand, all of the tests and confidence intervals seen in a traditional first course in statistics can be performed using a common outline:

1. Do it to your data
2. Do it to a randomized version of your data
3. Do it to lots of randomized versions of your data.

For example, we can use randomization in place of the two-sample t test to obtain an empirical p-value.

```
D <- diffmean(age ~ smoker, data = Whickham); D
```

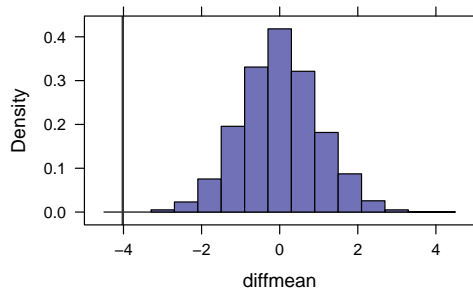
```
## diffmean
##      -4.02

do(1) * diffmean(age ~ shuffle(smoker), data = Whickham)

## diffmean
## 1      -1.69

Null.dist <- do(5000) * diffmean(age ~ shuffle(smoker), data = Whickham)
histogram( ~ diffmean, data = Null.dist, v = D, xlim = c(-5,5))
prop( ~ (diffmean < D), data = Null.dist)

## TRUE
##      0
```



None of the 5000 replications led to a difference in means as large as the one in the original data.

It should be noted that although this is typically not done in simulation-based introductory statistics texts, one might prefer to calculate p-values by including the observed data in the randomization distribution. This avoids an empirical p-value of 0 and guarantees that the actual type I error rate will not exceed the nominal type I error rate. This amounts to adding one to the numerator and denominator. The `prop1()` function automates this for us.

```
prop1( ~ (diffmean < D), data = Null.dist)

## TRUE
## 2e-04

1/5001

## [1] 2e-04
```

For more precise estimation of small p-values, additional replications should be used.

The example above introduces three additional **mosaic** functions. The `prop()` and `prop1()` functions compute the proportion of logical vector that is (by default) TRUE or of a factor that is (by default) in the first level; `diffmean()` is similar to `diff(mean())`, but labels the result differently (`diffprop()` works similarly for differences in proportions).

If we are interested in a confidence interval for the difference in group means, we can use `resample()` and `do()` to generate a bootstrap distribution in one of two ways.

```
Boot1 <- do(1000) * diffmean(age ~ smoker, data = resample(Whickham))
Boot2 <- do(1000) * diffmean(age ~ smoker, data = resample(Whickham, groups = smoker))
```

In the second example, the resampling happens within the smoker groups so that the marginal counts for each group remain fixed. This can be especially important if one of the groups is small, because otherwise some resamples might not include any observations of that group.

```
favstats(age ~ smoker, data = Whickham)

## smoker min Q1 median Q3 max mean sd n missing
## 1 No 18 32 48 65 84 48.7 18.8 732 0
## 2 Yes 18 32 45 57 84 44.7 15.3 582 0

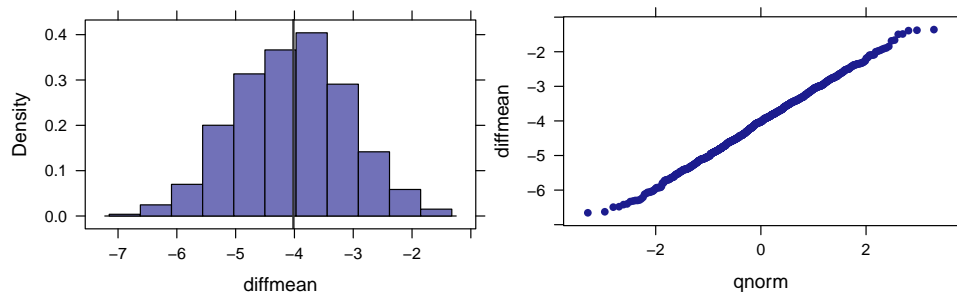
favstats(age ~ smoker, data = resample(Whickham))

## smoker min Q1 median Q3 max mean sd n missing
## 1 No 18 32 48 65 84 48.6 18.8 752 0
## 2 Yes 18 32 45 56 83 44.8 15.0 562 0
```

```
favstats(age ~ smoker, data = resample(Whickham, groups = smoker)) # fix margins
##   smoker min Q1 median Q3 max mean  sd  n missing
## 1   No   18 33  49.0 66  84 49.3 18.9 732      0
## 2   Yes  18 31  44.5 56  82 44.3 15.4 582      0
```

Using either bootstrap distribution, two simple confidence intervals can be computed. We typically introduce percentile confidence intervals first (but note that these can have poor performance for small sample sizes). A percentile confidence interval is calculated by determining the range of a central portion of the bootstrap distribution, which can be automated using `cdata()`. Visually inspecting the bootstrap distribution for skew and bias is an important step to make sure the percentile interval is not being applied in a situation where it may perform poorly.

```
histogram(~ diffmean, data = Boot2, v = D)
qqmath(~ diffmean, data = Boot2)
```



```
cdata(~ diffmean, p = 0.95, data = Boot2)
```

```
##      low      hi central.p
## -5.93 -2.30  0.95
```

Alternatively, we can compute a confidence interval based on a bootstrap estimate of the standard error.

```
SE <- sd(~ diffmean, data = Boot2); SE
```

```
## [1] 0.941
```

```
D + c(-1,1) * 2 * SE
```

```
## [1] -5.90 -2.14
```

The primary pedagogical value of the bootstrap standard error approach is its close connection to the standard formula-based confidence interval methods. How to replace the constant 2 with an appropriate value to create more accurate intervals or to allow for different confidence levels is a matter of some subtlety (Hesterberg, 2015). The simplest method is to use quantiles of a normal distribution, but the resulting intervals will typically undercover. Replacing the normal distribution with an appropriate t-distribution will widen intervals and can improve coverage, but the t-distribution is only correct in a few cases – such as when estimating the mean of a normal population – and can perform badly when the population is skewed.

Calculating simple confidence intervals can be further automated using an extension to `confint()`.

```
confint(Boot2, method = c("percentile", "stderr"))
```

```
##      name lower upper level  method estimate margin.of.error  df
## 1 diffmean -5.93 -2.3  0.95 percentile  -4.02             NA  NA
## 2 diffmean -5.89 -2.2  0.95  stderr    -4.02             1.85 1313
```

Additional examples

One of the package vignettes (Pruim et al., 2016a) contains a list of **mosaic**-related resources. Included in the list are links to companion volumes for several textbooks, including two simulation-based texts (Lock et al., 2013; Tintle et al., 2016) and several traditional textbooks (De Veaux et al., 2015; Moore et al., 2014; De Veaux et al., 2014; Ramsey and Schafer, 2013). Each of these companion volumes demonstrates how to use R and the **mosaic** package to recreate the analyses for the examples in the text.

Vignettes

The **mosaic** package includes several vignettes that provide additional material on using the package and on the “less volume, more creativity” approach.

A particularly useful vignette is a one-page list of commands that are more than sufficient for a first course, originally presented as part of a roundtable discussion at the Joint Statistics Meetings. (Pruim, 2011)

RMarkdown

“Thinking with data” goes hand-in-hand with communicating those thoughts. The R Markdown system provides valuable facilities for reliable and reproducible reporting of computational ideas and results. See Baumer et al. (2014) for a discussion of how R Markdown can be used in statistics courses.

The **mosaic** package contains three templates for creating R Markdown documents in RStudio. Each ensures that the **mosaic** package is attached, sets the default theme for **lattice** graphics to `theme.mosaic()`, chooses a somewhat smaller default size for graphics, and includes a comment reminding users to attach any packages they intend to use. The “fancy” template demonstrates several features of R Markdown, and the “plain” templates allow users to start with a clean slate.

Workarounds for unfortunate name collisions

The **mosaic** package depends on **lattice** and **ggplot2** so that plots can be made using either system whenever the **mosaic** package is attached. It also depends on **dplyr** (Wickham and Francois, 2015), but for a different reason. The functions in **dplyr** implement a “less volume, more creativity” approach to data transformation and we encourage its use alongside **mosaic**. Unfortunately, there are several function names – most notably `do()` and `tally()` – that exist in both packages. After the release of **dplyr** we modified the functions in **mosaic** so that the two packages can coexist amicably as long as **mosaic** comes before **dplyr** in the search path.

Discussion

Advantages of the mosaic approach

One of the keys to successfully empowering students to think with data is providing them both a conceptual framework that allows them to know what to look for and how to interpret what they find, and a computational toolbox that allows them to do the looking. The approach made possible with the **mosaic** package simplifies the transition from thinking to computing by reducing the number of computational templates students learn so that cognitive effort can be spent elsewhere, and by having those templates reflect, support, and deepen the underlying thinking (Grolemund and Wickham, 2014).

Because of the connection between conceptual understanding and these computational tools, the use of R can also help reveal misunderstandings that might otherwise go unnoticed. For example, if a student attempts to use `t.test()` or to create a histogram using a categorical variable, the student will receive error or warning messages that are an indication that either the student does not understand the current data set or still has confusion regarding what it means for a variable to be categorical or continuous and which operations are suited for each kind of variable. We encourage students to make sure they can answer two important questions before attempting to issue a command in R:

1. What do I want the computer to do for me?
2. What does it need to know in order to do that?

If these two questions can be answered clearly and correctly, then the student’s primary issue is one of creating the correct R code. If they cannot, then the problem lies elsewhere. In our experience, students who can consistently answer these two questions have relatively little trouble translating the answers into R code using the commands we teach.

R has the capability to support the increasing complexity of the data and analyses students encounter in subsequent courses and research projects. Eventually, students will need to learn more about the structure of R as a language, the types of objects it supports, and alternative ways of approaching the same task. But early on, it is more important that students can successfully and independently exercise computational and statistical creativity.

Challenges of using R in introductory courses

Using R is not without some challenges. The first challenge is to get all of the students up and running. The use of an RStudio server allows an institution or instructor to install and configure R and its packages and students to work within a web browser, essentially eliminating the start-up costs for the students. Otherwise, instructors must assist students as they navigate installation of R and whichever additional packages are required.

Once students have access to R, the **mosaic** package reduces, but does not eliminate, the amount of syntax students need to learn. It is important to emphasize the similarity among commands within a template, to remind students that R is case sensitive, to show them how to take advantage of shortcuts like tab completion and code history navigation, and to explicitly teach students how to interpret some of the most common R error messages. This goes a long way toward smoothing the transition to a command line interface that is not as forgiving as Google search (which may be many students' only other experience with computerized response to things they type).

In our experience, the most commonly occurring struggles for students using **mosaic** are

1. General anxiety over typing commands.

Although students are very familiar with using computers and computerized devices like smart phones, many of them have little experience typing commands that require following syntax rules. The "less volume, more creativity" approach helps with this, by reducing the cognitive burden. But it remains important to highlight repeatedly the similarities among commands. Students should also be taught how to interpret the most common error messages R produces so that they can quickly, easily, and comfortably recover from inevitable typing errors. Even if a class does not typically meet in a computer laboratory or take advantage of student laptops, it can be useful to arrange some sessions early in the course where students are using RStudio while someone is there to quickly help them when they get stuck. Avoiding frustration in students' early experience with R goes a long way in overcoming anxiety.

As a bonus instructional method, the authors make frequent typing mistakes in front of the class. While we could not avoid this if we tried, it does serve to demonstrate both how to recover from errors and that nothing drastic has happened when an error message is displayed.

One big advantage of the command line interface is that it is much easier to help students by email or in a discussion forum. Encourage students to copy both their commands and the error messages or output that were produced. Even better, have them share their work in the form of an R Markdown file. We find students are much more capable of doing this than they are of correctly describing the chain of events they initiated in a menu-driven system. (It is also much easier to give detailed instructions and examples.)

2. Confusion over the tilde (~).

The tilde is a small symbol, easily overlooked on the screen or on paper (or mistaken for -), so students will sometimes omit it, or put it where it doesn't belong. As a visual aid, we recommend surrounding the ~ with a space on either side, even in 1-sided formulas.

A similar thing occurs with explicitly naming the data argument, which is not required for the **lattice** functions, but is for several other functions. Teaching the forms that work in all contexts is easier than teaching which contexts allow which forms.

3. Difficulty in setting up the R environment

This is all but eliminated when using an RStudio server, but in situations where instructors prefer a local R installation for each student, there are often a few issues involved in getting all students up and running. Installation of R and RStudio is straightforward, but one should make sure that students all have the latest version of each. To use the **mosaic** package, a number of additional packages must be installed. We recommend beginning with

```
update.packages()
```

or the equivalent operation from the RStudio "Packages" tab to make sure all packages currently on the system are up to date. In most cases,

```
install.packages("mosaic")
```

(again, this can also be done via the "Packages" tab in RStudio) will take care of the rest. But occasionally some package will not install correctly on a particular student's computer. Installing that package directly rather than as part of the dependencies of **mosaic** often solves this problem or at least provides a useful diagnostic regarding what the problem might be.

Impact

In 2016, the available CRAN logs indicate that **mosaic** was downloaded to approximately 96,000 unique IP addresses. Strong peaks over January and September suggest that much of this demand comes from university courses.

Efficiency Issues

For applications where speed is of utmost importance, the **mosaic** wrappers may not be the optimal approach. For the numerical summary functions, the **mosaic** versions cannot be faster than their counterparts in **base** or **stats** (because eventually they call the underlying functions) and may be noticeably slower in contexts where they are called many times. In particular, using the formula interface requires parsing the formula and creating a new object to contain the data described by the formula. On the other hand, for aggregated numerical summaries, the loss in performance may represent a small price to pay for the simplified syntax.

Similarly, using `do()` comes at a price, although here the increased computation time has more to do with the extra work involved in culling the objects and reformatting the results. The looping itself is as fast as using `replicate()` – indeed the underlying code is very similar – and can be faster when the **parallel** package is attached; even on a laptop with a single quad-core processor, the speed-up is noticeable.

Lattice vs. ggplot2

Early on, we chose to adopt **lattice** graphics because of its compatibility with the formula template. This provides a simple, consistent means of creating the plots our students need. One weakness of the **lattice** system is the difficulty of creating complex plots by overlaying multiple simpler layers. Beginners are not in a position to create the panel (and pre-panel) functions that **lattice** requires for this. Recently, we have begun work on a new package (**ggformula**, currently available via github) that provides a formula interface for creating **ggplot2** plots. This package could replace **lattice** for users who desire some of the features of **ggplot2** but want to keep a consistent formula interface. Initial use with our students suggests that this works at least as well as **lattice** and much better if plots with multiple layers and data sources are required.

Be selective

Over the years we have been developing the **mosaic** package, it has grown to the point that it now contains much more than a minimally sufficient set of commands for an introductory course. While we have attempted to give some sense of the scope of the package in this article, we advise instructors to use things selectively, keeping in mind their students and the goals for the course. What may represent just the right tool in one setting may be too much in another. Of course, the same advice holds for using functions from other packages as well. The instructor's temptation is often to do too much, forgetting the cognitive burden this can place on students. Less volume and more creativity will at times pull in opposite directions, and a skilled instructor must determine the appropriate balance for each setting.

Acknowledgments

Partial support for this work was provided by the National Science Foundation DUE 0920350 (Project MOSAIC). We thank Johanna Hardin, Colin Rundel, Xiaofei (Susan) Wang, and the reviewers for helpful comments and all of the users of the **mosaic** package who have provided feedback on their experience and offered suggestions for improvement.

Appendix: Additional features of the mosaic package

Table 2 lists some additional functions in the **mosaic** package not highlighted above.

Handling missing data

When there are missing values, the numerical summary functions in **base** and **stats** return results that may surprise new users.

Function	Uses
CIsim()	Demonstrate coverage rates of confidence intervals.
statTally()	Investigate test statistics and their empirical distributions.
panel.lmbands()	Add confidence and prediction bands to scatter plots.
ladd()	Simplified layering in lattice plots.
xchisq.test()	An extension to <code>chisq.test()</code> that prints a table including observed and expected counts, contribution to the chi-squared statistic and residuals.
zscore()	Convert a numeric vector into z-scores by subtracting the mean and dividing by the standard deviation.
col.mosaic()	A lattice theme with colors that work better in project images than the lattice defaults.
dot(), project(), vlength()	Functions from linear algebra.
ediff()	Like <code>diff()</code> , but the returned vector is padded with NAs so that the length is the same as the input vector.
SAD(), MAD()	all pairs sum and mean of absolute differences
rgeo()	randomly sample latitude, longitude pairs uniformly over the globe
googleMap()	show google maps in a browser. Together with <code>rgeo()</code> , this can be used to view maps of randomly selected points on the globe. See Stoudt et al. (2014) for an example of how this can be used for a classroom activity.

Table 2: Some additional functions in the **mosaic** package.

```
mean( ~ dayslink, data = HELPmiss)
```

```
## [1] NA
```

While there are workarounds using options to functions to drop values that are missing before performing the computation, these may be intimidating to new users.

```
mean( ~ dayslink, data = HELPmiss, na.rm = TRUE)
```

```
## [1] 257
```

We offer two other solutions to this situation. Our favorite is the `favstats()` function which computes a set of useful numerical summaries on the non-missing values and also reports the number of missing values.

```
favstats( ~ dayslink, data = HELPmiss)
```

```
##   min Q1 median  Q3 max mean  sd   n missing
##   2  75   363 365 456 257 151 447     23
```

The second solution is to change the default behavior of `na.rm` using `options()`. This will, of course, only affect the **mosaic** versions of these functions.

```
options(na.rm = TRUE)
mean( ~ dayslink, data = HELPmiss)
```

```
## [1] 257
```

```
with(HELPmiss, base::mean(dayslink))
```

```
## [1] NA
```

Users also have the option of changing the default for `na.rm` back if they like.

```
options(na.rm = NULL)
mean( ~ dayslink, data = HELPmiss)
```

```
## [1] NA
```

Inspecting a data frame

Summaries of all variables in a data frame can be obtained using `inspect()`. For quantitative variables, the results of `favstats()` are displayed. Other summaries are provided for categorical and time variables.

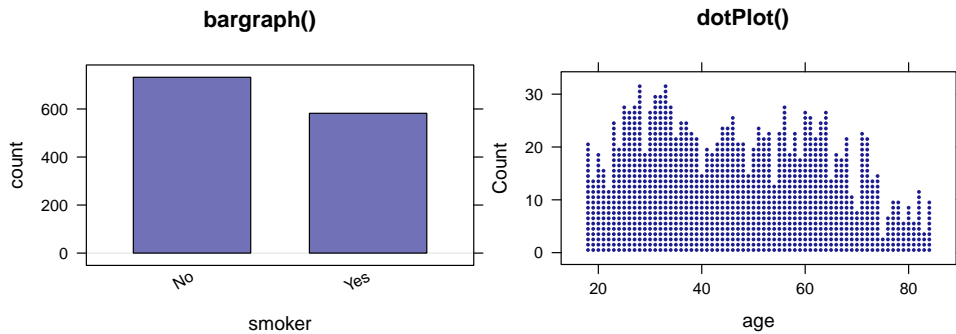
```
inspect(Births78)

##
## categorical variables:
## name class levels n missing distribution
## 1 wday ordered 7 365 0 Sun (14.5%), Mon (14.2%), Tues (14.2%) ...
##
## quantitative variables:
## name class min Q1 median Q3 max mean sd n missing
## 1 births integer 7135 8554 9218 9705 10711 9132 818 365 0
## 2 dayofyear integer 1 92 183 274 365 183 106 365 0
##
## time variables:
## name class first last min_diff max_diff n missing
## 1 date POSIXct 1978-01-01 1978-12-31 1 1 365 0
```

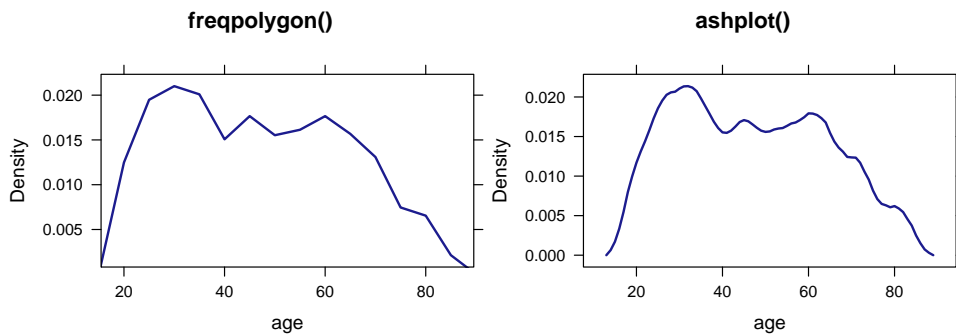
Additional high-level lattice plots

The **mosaic** package provides several new high-level **lattice** plots, including `bargraph()`, `dotPlot()`, `freqpolygon()`, `ashplot()`, `xqqmath()`, and `plotPoints()`.

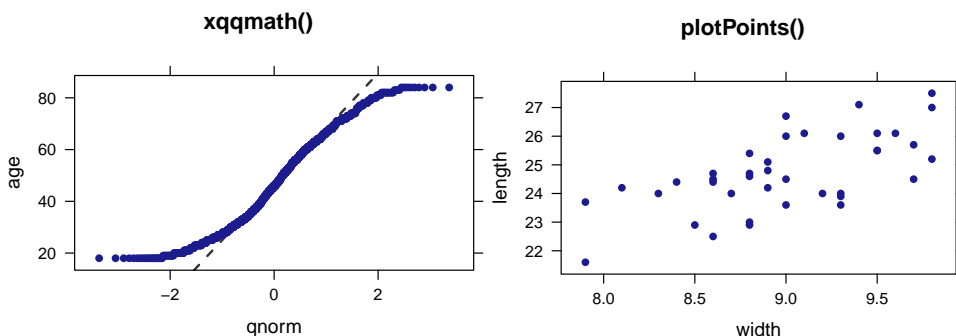
```
bargraph( ~ smoker, data = Whickham, main = "bargraph()")
dotPlot( ~ age, data = Whickham, width = 1, main = "dotPlot()")
```



```
freqpolygon( ~ age, data = Whickham, width = 5, main = "freqpolygon()")
ashplot( ~ age, data = Whickham, width = 5, main = "ashplot()")
```



```
xqqmath( ~ age, data = Whickham, main = "xqqmath()")
plotPoints(length ~ width, data = KidsFeet, main = "plotPoints()")
```

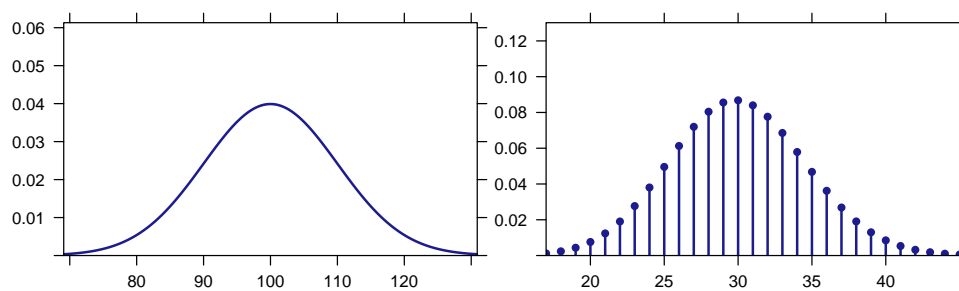


The `barplot()` function eliminates the need to first summarize the data before constructing a plot with `barplot()` and makes creating these plots from raw data simpler. The dot plots produced by `dotPlot()` are quite different from the Cleveland-style dot plots produced by `dotplot()`. The former are essentially histograms made of stacked dots and can be seen in many introductory statistics texts. They are also useful for producing plots from which students can quickly estimate p-values by counting dots in the tail of a randomization distribution. Frequency polygons and ASH plots (average shifted histograms) are less common, but share many features in common with density plots and are easier to explain to students. The main motivation for `plotPoints()` is the ability to use it to create additional layers on an existing plot with the option `add = TRUE`, otherwise `xyplot()` would suffice.

Visualizing distributions of random variables

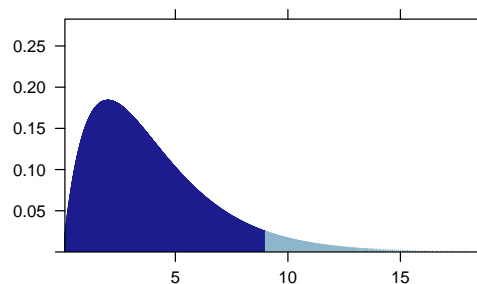
A number of functions make it simple to visualize random variables. The `plotDist()` function creates displays for any distribution for which standard `d-`, `p-`, and `q-` functions exist.

```
plotDist("norm", mean = 100, sd = 10)
plotDist("binom", size = 100, prob = 0.3)
```



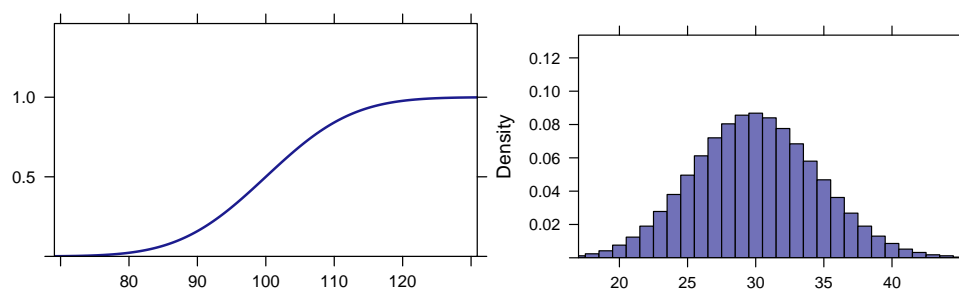
Tail probabilities can be highlighted using the `groups` argument in a way that is analogous to the lattice plots above.

```
plotDist("chisq", df = 4, groups = x > 9, type = "h")
```



Using the `kind` argument, we can obtain other kinds of plots, including cdfs and probability histograms.

```
plotDist("norm", mean = 100, sd = 10, kind = "cdf")
plotDist("binom", size = 100, prob = 0.3, kind = "histogram")
```



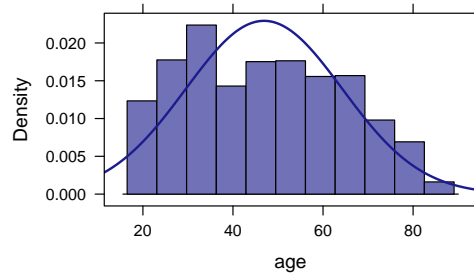
Any of these plots can be overlaid onto another plot using `add = TRUE`

```
favstats(~ age, data = Whickham)
```

```
## min Q1 median Q3 max mean sd n missing
## 18 32 46 61 84 46.9 17.4 1314 0

histogram(~ age, data = Whickham, main = 'histogram() with added plotDist()')
plotDist("norm", params = list(mean = 46.9, sd = 17.4), add = TRUE)
```

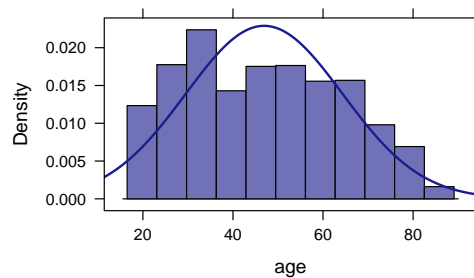
histogram() with added plotDist()



or by using additional features of the histogram() function provided in the **mosaic** package:

```
histogram(~ age, data = Whickham, fit = "normal",
          main = 'histogram() with fit = "normal"')
```

histogram() with fit = "normal"



Several other families of distributions can be added to a histogram in a similar way. The `fitdistr()` function from the **MASS** (Venables and Ripley, 2002) is used to estimate the parameters of the distribution.

For several distributions, we provide augmented versions of the distribution and quantile functions that assist students in understanding what values are returned by functions like `pnorm()` and `qnorm()`.

```
xpnorm(-2:2, main = "standard normal")

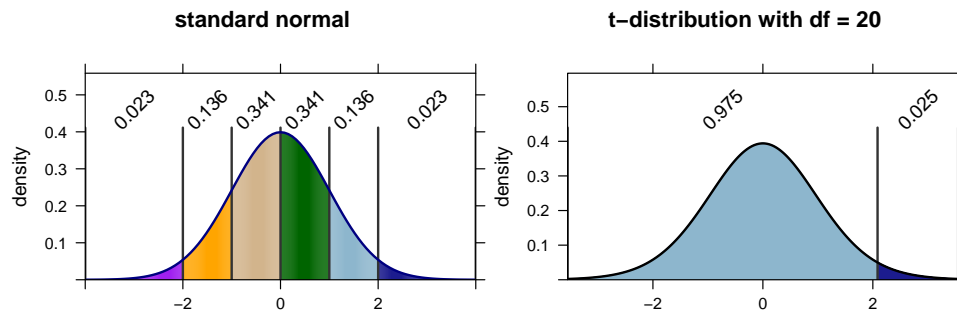
##
## If  $X \sim N(0, 1)$ , then
##
##  $P(X \leq -2) = P(Z \leq -2) = 0.02275$ 
##  $P(X \leq -1) = P(Z \leq -1) = 0.15866$ 
##  $P(X \leq 0) = P(Z \leq 0) = 0.50000$ 
##  $P(X \leq 1) = P(Z \leq 1) = 0.84134$ 
##  $P(X \leq 2) = P(Z \leq 2) = 0.97725$ 
##  $P(X > -2) = P(Z > -2) = 0.97725$ 
##  $P(X > -1) = P(Z > -1) = 0.84134$ 
##  $P(X > 0) = P(Z > 0) = 0.50000$ 
##  $P(X > 1) = P(Z > 1) = 0.15866$ 
##  $P(X > 2) = P(Z > 2) = 0.02275$ 

## [1] 0.0228 0.1587 0.5000 0.8413 0.9772

xqt(0.975, df = 20, main = "t-distribution with df = 20")

## 95%
## 0.000 0.597

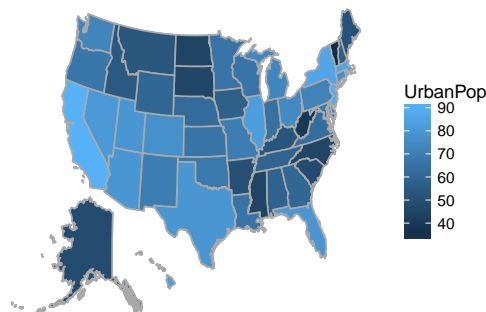
## [1] 2.09
```



Basic choropleth maps

The `mUSMap()` and `mWorldMap()` functions provide simple ways to construct choropleth maps of states in the US or countries in the world, and `makeMap()` allows users to provide their own map data.

```
USArrests <- USArrests %>% mutate(state = row.names(USArrests))
mUSMap(USArrests, key = "state", fill = "UrbanPop")
```



The `do()` function vs. the `replicate()` function

The usual alternative to `do()` is `replicate()`. For simple situations, `replicate()` can also be easy to use. Each of these stores its results in a vector rather than in a data frame but is otherwise very similar to the corresponding results using `do()`, although the first stores less information.

```
replicate(3, rflip(20))

## [1] 9 13 11

replicate(3, diffmean(age ~ smoker, data = resample(Whickham)))

## diffmean diffmean diffmean
## -2.04 -4.56 -5.33
```

Where `do()` really shines is for simulations based on models. The results returned by `do()` are stored in a data frame and include the components of the model most likely to be of interest.

```
do(3) * lm(shuffle(height) ~ sex + mother, data = Galton)

## Intercept sexM mother sigma r.squared F numdf dendf .row .index
## 1 64.1 0.0356 0.04086 3.59 0.000708 0.3168 2 895 1 1
## 2 66.4 0.0869 0.00543 3.59 0.000156 0.0699 2 895 1 2
## 3 62.2 0.0175 0.07175 3.58 0.002132 0.9562 2 895 1 3
```

Resampling from a linear model performs residual resampling:

```
Galton.mod <- lm(height ~ sex + mother, data = Galton)
do(3) * lm(height ~ sex + mother, data = resample(Galton.mod))

## Intercept sexM mother sigma r.squared F numdf dendf .row .index
## 1 41.2 5.01 0.358 2.44 0.534 513 2 895 1 1
## 2 41.1 5.26 0.357 2.40 0.563 577 2 895 1 2
## 3 41.7 5.07 0.351 2.37 0.553 554 2 895 1 3
```

In contrast, `replicate()` returns an object that is inscrutable and unusable for most beginners.

```
replicate(3, lm(shuffle(height) ~ sex + mother, data = Galton))
```

```
##           [,1]      [,2]      [,3]
## coefficients Numeric,3 Numeric,3 Numeric,3
## residuals   Numeric,898 Numeric,898 Numeric,898
## effects     Numeric,898 Numeric,898 Numeric,898
## rank        3         3         3
## fitted.values Numeric,898 Numeric,898 Numeric,898
## assign      Integer,3   Integer,3   Integer,3
## qr          List,5     List,5     List,5
## df.residual 895       895       895
## contrasts   List,1     List,1     List,1
## xlevels    List,1     List,1     List,1
## call       Expression Expression Expression
## terms     Expression Expression Expression
## model     List,3     List,3     List,3
```

With some additional work, this can be improved somewhat, although less information is being recorded and the matrix should probably be transposed (and perhaps converted to a data frame).

```
replicate(3, coef(lm(shuffle(height) ~ sex + mother, data = Galton)))
```

```
##           [,1]      [,2]      [,3]
## (Intercept) 67.774 70.4448 70.2286
## sexM       -0.226  0.0372 -0.0206
## mother     -0.014 -0.0578 -0.0539
```

Bibliography

- ASA GAISE College working group, R. Carver, M. Everson, J. Gabrosek, G. H. Rowell, N. J. Horton, R. Lock, M. Mocko, A. Rossman, P. Velleman, J. Witmer, and B. Wood. Guidelines for assessment and instruction in statistics education: College report, 2016. URL <http://www.amstat.org/education/gaise>. [p77]
- B. Baumer, M. Cetinkaya-Rundel, A. Bray, L. Loi, and N. J. Horton. R markdown: Integrating a reproducible analysis tool into introductory statistics. *Technology Innovations in Statistics Education*, 8 (1), 2014. URL <http://escholarship.org/uc/item/90b2f5xh>. [p92]
- R. D. De Veaux, P. F. Velleman, and D. E. Bock. *Intro Stats*. Always learning. Pearson Education, Limited, 2014. ISBN 9780321891358. URL <https://books.google.com/books?id=3mhRnwEACAAJ>. [p91]
- R. D. De Veaux, P. F. Velleman, and D. E. Bock. *Stats: Data and Models*. Pearson Education, 2015. ISBN 9780134175621. URL <https://books.google.com/books?id=0degBwAAQBAJ>. [p91]
- G. Golemund and H. Wickham. A cognitive interpretation of data analysis. *International Statistical Review*, 82(2):184–204, 2014. URL <http://EconPapers.repec.org/RePEc:bla:istatr:v:82:y:2014:i:2:p:184-204>. [p92]
- T. C. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. *The American Statistician*, 69(4):371–386, 2015. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4784504>. [p88, 91]
- N. J. Horton and J. S. Hardin. Teaching the next generation of statistics students to “think with data”: Special issue on statistics and the undergraduate curriculum. *The American Statistician*, 69(4):259–265, 2015. URL <http://amstat.tandfonline.com/doi/full/10.1080/00031305.2015.1094283>. [p77]
- N. J. Horton, B. S. Baumer, and H. Wickham. Setting the stage for data science: Integration of data management skills in introductory and second courses in statistics. *CHANCE*, 28(2):40–50, 2015. URL <http://chance.amstat.org/2015/04/setting-the-stage>. [p77]
- R. H. Lock, P. F. Lock, K. L. Morgan, E. F. Lock, and D. F. Lock. *Statistics: Unlocking the Power of Data*. John Wiley & Sons, 2013. [p91]

- D. S. Moore, G. P. McCabe, and B. A. Craig. *Introduction to the Practice of Statistics*. W. H. Freeman, 2014. ISBN 9781464133633. URL https://books.google.com/books?id=pX1_AwAAQBAJ. [p91]
- D. Nolan and D. T. Lang. Computing in the statistics curricula. *The American Statistician*, 64(2):97–107, 2010. URL <http://dx.doi.org/10.1198/tast.2010.09132>. [p77]
- R. Pruim. Teaching statistics with R. In *Joint Statistics Meetings Roundtable*, 2011. [p92]
- R. Pruim, D. T. Kaplan, and N. J. Horton. *mosaicData: Project MOSAIC (Mosaic-Web.org) Data Sets*. URL <https://github.com/ProjectMOSAIC/mosaicData>. R package version 0.14.0. [p80]
- R. Pruim, N. J. Horton, and D. T. Kaplan. Resources related to the mosaic package. Technical report, Project MOSAIC, 2016a. URL <https://cran.r-project.org/web/packages/mosaic/vignettes/mosaic-resources.html>. [p91]
- R. Pruim, D. T. Kaplan, and N. J. Horton. *Mosaic: Project MOSAIC Statistics and Mathematics Teaching Utilities*, 2016b. URL <https://github.com/ProjectMOSAIC/mosaic>. R package version 0.14.0. [p77]
- F. Ramsey and D. Schafer. *Statistical Sleuth: A Course in Methods of Data Analysis*. Brooks-Cole, 2013. [p91]
- J. Ridgway. Implications of the data revolution for statistics education. *International Statistical Review*, 84(3):327–557, 2016. [p77]
- D. Salsburg. *The Lady Tasting Tea: How Statistics Revolutionized Science in the Twentieth Century*. Holt Paperbacks, 2002. ISBN 0805071342. [p88]
- D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York, 2008. URL <http://lmdvr.r-forge.r-project.org>. ISBN 978-0-387-75968-5. [p78]
- S. Stoudt, Y. Cao, D. Udwin, and N. J. Horton. What percent of the continental US is within one mile of a road? *Statistics Education Web*, 2014. URL <http://www.amstat.org/education/STEW/pdfs/PercentWithinMileofRoad.pdf>. [p95]
- N. Tintle, B. Chance, G. W. Cobb, S. Roy, T. Swanson, and J. VanderStoep. Combating Anti-Statistical Thinking Using Simulation-Based Methods throughout the Undergraduate Curriculum. *The American Statistician*, 69(4):362–370, 2015. doi: 10.1080/00031305.2015.1081619. URL http://www.math.hope.edu/isi/presentations/white_paper_sim_inf_thru_curriculum.pdf. [p88]
- N. Tintle, B. L. Chance, G. W. Cobb, A. J. Rossman, S. Roy, T. Swanson, and J. VanderStoep. *Introduction to Statistical Investigations*. John Wiley & Sons, 2016. [p91]
- 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. [p98]
- X. Wang, C. Rush, and N. J. Horton. Data visualization on day one: Bringing big ideas into intro stats early and often. *Technology Innovations in Statistics Education*, in press, 2017. URL <https://arxiv.org/abs/1705.08544>. [p83]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. URL <http://had.co.nz/ggplot2/book>. [p86]
- H. Wickham and R. Francois. *Dplyr: A Grammar of Data Manipulation*, 2015. URL <https://github.com/hadley/dplyr>. R package version 0.5.0.9000. [p92]
- C. J. Wild, M. Pfannkuch, M. Regan, and N. J. Horton. Towards more accessible conceptions of statistical inference. *Journal of the Royal Statistical Society A*, 174 (part 2):247–295, 2011. [p81]

Randall Pruim
Calvin College
Department of Mathematics and Statistics
3201 Burton St SE
Grand Rapids, MI 49546
rpruim@calvin.edu

Daniel T Kaplan
Macalester College
Department of Mathematics and Computer Science

*1600 Grand Avenue
St. Paul, MN 55105 USA
dtkaplan@macalester.edu*

*Nicholas J Horton
Amherst College
Department of Mathematics and Statistics
PO Box 5000 AC #2239
Amherst, MA 01002-5000
nhorton@amherst.edu*

smoof: Single- and Multi-Objective Optimization Test Functions

by Jakob Bossek

Abstract Benchmarking algorithms for optimization problems usually is carried out by running the algorithms under consideration on a diverse set of benchmark or test functions. A vast variety of test functions was proposed by researchers and is being used for investigations in the literature. The **smoof** package implements a large set of test functions and test function generators for both the single- and multi-objective case in continuous optimization and provides functions to easily create own test functions. Moreover, the package offers some additional helper methods, which can be used in the context of optimization.

Introduction

The task of *global optimization* is to find the best solution $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbf{X}$ according to a set of objective functions $\mathcal{F} = \{f_1, \dots, f_m\}$. If input vectors as well as output values of the objective functions are real-valued, i. e., $f_i: \mathbb{R}^n \rightarrow \mathbb{R}$ the optimization problem is called *continuous*. Otherwise, i. e., if there is at least one non-continuous parameter, the problem is termed *mixed*. For $m = 1$, the optimization problem is termed *single-objective* and the goal is to minimize a single objective f , i. e.,

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}).$$

Clearly, talking about minimization problems is no restriction: we can maximize f by minimizing $-f$. Based on the structure of the search space, there may be multiple or even infinitely many global optima, i. e., $\mathbf{x}^* \in \mathbf{X}^* \subseteq \mathbf{X}$. We are faced with a *multi-objective* optimization problem if there are at least two objective functions. In this case as a rule no global optimum exists since the objectives are usually conflicting and there is just a partial order on the search space; for sure $(1, 4)^T \leq (3, 7)^T$ makes sense, but $(1, 4)^T$ and $(3, 2)^T$ are not comparable. In the field of multi-objective optimization we are thus interested in a set

$$PS = \{\mathbf{x} \in \mathbf{X} \mid \nexists \tilde{\mathbf{x}} \in \mathbf{X} : f(\tilde{\mathbf{x}}) \preceq f(\mathbf{x})\} \subseteq \mathbf{X}$$

of optimal trade-off solutions termed the *Pareto-optimal set*, where \preceq defines the *dominance relation*. A point $\mathbf{x} \in X$ dominates another point $\tilde{\mathbf{x}} \in X$, i. e., $\mathbf{x} \preceq \tilde{\mathbf{x}}$ if

$$\begin{aligned} &\forall i \in \{1, \dots, m\} : f_i(\mathbf{x}) \leq f_i(\tilde{\mathbf{x}}) \\ &\text{and } \exists i \in \{1, \dots, m\} : f_i(\mathbf{x}) < f_i(\tilde{\mathbf{x}}). \end{aligned}$$

Hence, all trade-off solutions $\mathbf{x}^* \in PS$ are *non-dominated*. The image of the Pareto-set $PF = f(PS) = (f_1(PS), \dots, f_m(PS))$ is the *Pareto-front* in the objective space. See Coello et al. (2006) for a thorough introduction to multi-objective optimization.

There exists a plethora of optimization algorithms for single-objective continuous optimization in R (see the CRAN Task View on [Optimization and Mathematical Programming](#) (Theussl and Borchers) for a growing list of available implementations and packages). Mullen (2014) gives a comprehensive review of continuous single-objective global optimization in R. In contrast there are just a few packages, e. g., **emoa** (Mersmann, 2012), **mco** (Mersmann, 2014), **ecr** (Bossek, 2017a), with methods suited to tackle continuous multi-objective optimization problems. These packages focus on evolutionary multi-objective algorithms (EMOA), which are very successful in approximating the Pareto-optimal set.

Benchmarking optimization algorithms

In order to investigate the performance of optimization algorithms or for comparing of different algorithmic optimization methods in both the single- and multi-objective case a commonly accepted approach is to test on a large set of artificial test or benchmark functions. Artificial test functions exhibit different characteristics that pose various difficulties for optimization algorithms, e. g., multimodal functions with more than one local optimum aim to test the algorithms' ability to escape from local optima. Scalable functions can be used to access the performance of an algorithm while increasing the dimensionality of the decision space. In the context of multi-objective problems the geometry of the Pareto-optimal front (convex, concave, ...) as well as the degree of multimodality are important

characteristics for potential benchmarking problems. An overview of single-objective test function characteristics can be found in [Jamil and Yang \(2013\)](#). A thorough discussion of multi-objective problem characteristics is given by [Huband et al. \(2006\)](#). [Kerschke and Dagefoerde \(2015\)](#) recently published an R package with methods suited to quantify/estimate characteristics of unknown optimization functions at hand. Since the optimization community mainly focuses on purely continuous optimization, benchmarking test sets lack functions with discrete or mixed parameter spaces.

Related work

Several packages make benchmark functions available in R. The packages `cec2005benchmark` ([Gonzalez-Fernandez and Soto, 2015](#)) and `cec2013` ([Zambrano-Bigiarini and Gonzalez-Fernandez, 2015](#)) are simple wrappers for the C implementations of the benchmark functions for the corresponding CEC 2005/2013 Special Session on Real-Parameter Optimization and thus very limited. The `globalOptTests` ([Mullen, 2014](#)) package interfaces 50 continuous single-objective functions. Finally the `soobench` ([Mersmann and Bischl, 2012](#)) package contains some single-objective benchmark functions and in addition several useful methods for visualization and logging.

Contribution

The package `smoof` ([Bossek, 2017b](#)) contains generators for a large and diverse set of both single-objective and multi-objective optimization test functions. Single-objective functions are taken from the comprehensive survey by [Jamil and Yang \(2013\)](#) and black-box optimization competitions ([Hansen et al., 2009](#); [Gonzalez-Fernandez and Soto, 2015](#)). Moreover, a flexible function generator introduced by [Wessing \(2015\)](#) is interfaced. Multi-objective test functions are taken from [Deb et al. \(2002\)](#); [Zitzler et al. \(2000\)](#) and [Zhang et al. \(2009\)](#). In the current version – version 1.4 in the moment of writing – there are 99 continuous test function generators available (72 single-objective, 24 multi-objective, and 3 function family generators). Discontinuous functions (2) and functions with mixed parameters spaces (1) are underrepresented at the moment. This is due to the optimization community mainly focusing on continuous functions with numeric-only parameter spaces as stated above. However, we plan to extend this category in upcoming releases.

Both single- and multi-objective `smoof` functions share a common and extensible interface, which allows to easily add new test functions. Finally, the package contains additional helper methods which facilitate logging in the context of optimization.

Installation

The `smoof` package is available on CRAN, the Comprehensive R Archive Network, in version 1.4. To download, install and load the current release, just type the code below in your current R session.

```
> install.packages("smoof")
> library(smoof)
```

If you are interested in toying around with new features take a look at the public repository at GitHub (<https://github.com/jakobbossek/smoof>). This is also the place to complain about problems and missing features / test functions; just drop some lines to the issue tracker.

Diving into the `smoof` package

In this section we first explain the internal structure of a test function in the `smoof` package. Later we take a look on how to create objective functions, the predefined function generators and visualization. Finally, we present additional helper methods which may facilitate optimization in R.

Anatomy of `smoof` functions

The functions `makeSingleObjectiveFunction` and `makeMultiObjectiveFunction` respectively can be used to create objective functions. Both functions return a regular R function with its characteristic properties appended in terms of attributes. The properties are listed and described in detail below.

name The function name. Mainly used for plots and console output.

id Optional short name. May be useful to index lists of functions.

description Optional description of the function. Default is the empty string.

- fn** The actual implementation of the function. This must be a function of a single argument x .
- has.simple.signature** Logical value indicating whether the function `fn` expects a simple vector of values or a named list. This parameter defaults to `TRUE` and should be set to `FALSE`, if the function depends on a mixed parameter space, i. e., there are both numeric and factor parameters.
- par.set** The set of function parameters of `fn`. **smoof** makes use of the [ParamHelpers](#) (Bischi et al., 2016) package to define parameters.
- noisy** Is the function noisy? Default is `FALSE`.
- minimize** Logical value(s) indicating which objectives are to be minimized (`TRUE`) or maximized (`FALSE`) respectively. For single objective functions a single logical value is expected. For multi-objective test functions a logical vector with `n.objectives` components must be provided. Default is to minimize all objectives.
- vectorized** Does the function accept a matrix of parameter values? Default is `FALSE`.
- constraint.fn** Optional function which returns a logical vector indicating which non-box-constraints are violated.
- tags** A character vector of tags. A tag is a kind of label describing a property of the test function, e.g., *multimodal* or *separable*. Call the `getAvailableTags` function for a list of possible tags and see (Jamil and Yang, 2013) for a description of these. By default, there are no tags associated with the test function.
- global.opt.params** If the global optimum is known, it can be passed here as a vector, matrix, list or `data.frame`.
- global.opt.value** The function value of the `global.opt.params` argument.
- n.objectives** The number of objectives.

Since there exists no global optimum in multi-objective optimization, the arguments `global.opt.params` and `global.opt.value` are exclusive to the single-objective function generator. Moreover, tagging is possible for the single-objective case only until now. In contrast, the property `n.objectives` is set to 1 internally for single-objective functions and is thus no parameter of `makeSingleObjectiveFunction`.

Creating smoof functions

The best way to describe how to create an objective function in **smoof** is via example. Assume we want to add the two-dimensional Sphere function

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}, \mathbf{x} \mapsto x_1^2 + x_2^2 \text{ with } x_1, x_2 \in [-10, 10]$$

to our set of test functions. The unique global optimum is located at $\mathbf{x}^* = (0, 0)^T$ with a function value of $f(\mathbf{x}^*) = 0$. The code below is sufficient to create the Sphere function with **smoof**.

```
> fn <- makeSingleObjectiveFunction(
>   name = "2D-Sphere",
>   fn = function(x) x[1]^2 + x[2]^2,
>   par.set = makeNumericParamSet(
>     len = 2L, id = "x",
>     lower = c(-10, -10), upper = c(10, 10),
>     vector = TRUE
>   ),
>   tags = "unimodal",
>   global.opt.param = c(0, 0),
>   global.opt.value = 0
> )
> print(fn)
Single-objective function
Name: 2D-Sphere
Description: no description
Tags:
Noisy: FALSE
Minimize: TRUE
Constraints: TRUE
Number of parameters: 2
      Type len Def          Constr Req Tunable Trafo
x numericvector  2  - -10,-10 to 10,10  -   TRUE   -
```

Global optimum objective value of 0.0000 at

```
x1 x2
1 0 0
```

Here we pass the mandatory arguments `name`, the actual function definition `fn` and a parameter set `par.set`. We state, that the function expects a single numeric vector parameter of length two where each component should satisfy the box constraints ($x_1, x_2 \in [-10, 10]$). Moreover we let the function know its own optimal parameters and the corresponding value via the optional arguments `global.opt.param` and `global.opt.value`. The remaining arguments fall back to their default values described above.

As another example we construct a mixed parameter space function with one numeric and one discrete parameter, where the latter can take the three values a, b and c respectively. The function is basically a shifted single-objective Sphere function, where the shift in the objective space depends on the discrete value. Since the function is not purely continuous, we need to pass the calling entity a named list to the function and thus `has.simple.signature` is set to `FALSE`.

```
> fn2 <- makeSingleObjectiveFunction(
>   name = "Shifted-Sphere",
>   fn = function(x) {
>     shift = which(x$disc == letters[1:3]) * 2
>     return(x$num^2 + shift)
>   },
>   par.set = makeParamSet(
>     makeNumericParam("num", lower = -5, upper = 5),
>     makeDiscreteParam("disc", values = letters[1:3])
>   ),
>   has.simple.signature = FALSE
> )
> print(fn2)
Single-objective function
Name: Shifted-Sphere
Description: no description
Tags:
Noisy: FALSE
Minimize: TRUE
Constraints: TRUE
Number of parameters: 2
      Type len Def  Constr Req Tunable Trafo
num  numeric  -  - -5 to 5  -   TRUE   -
disc discrete  -  - a,b,c  -   TRUE   -

> fn2(list(num = 3, disc = "c"))
[1] 15
```

Visualization

There are multiple methods for the visualization of 1D or 2D smooth functions. The generic `plot` method draws a contour plot or level plot or a combination of both. The following code produces the graphics depicted in Figure 1 (left).

```
> plot(fn, render.contours = TRUE, render.levels = TRUE)
```

Here the argument `render.levels` achieves the heatmap effect, whereas `render.contours` activates the contour lines. Moreover, numeric 2D functions can be visualized as a 3D graphics by means of the `plot3D` function (see Fig. 1 (right)).

```
> plot3D(fn, contour = TRUE)
```

If you prefer the visually appealing graphics of `ggplot2` (Wickham, 2009) you can make use of `autoplot`, which returns a `ggplot2` object. The returned `ggplot` object can be easily modified with additional geometric objects, statistical transformations and layers. For instance, let us visualize the mixed parameter function `fn2` which was introduced in the previous subsection. Here we activate `ggplot2` faceting via `use.facets = TRUE`, flip the default facet direction and adapt the limits of the objective axis by hand. Figure 2 shows the resulting plot.

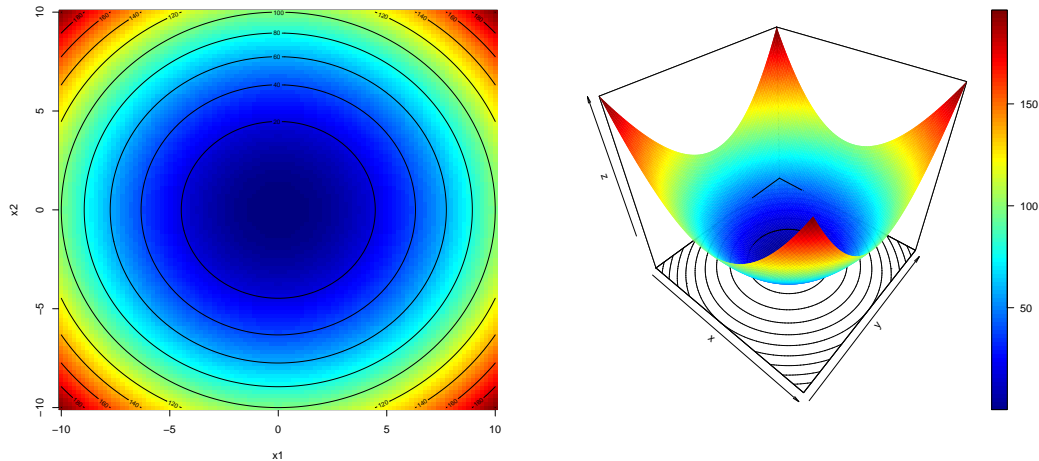


Figure 1: Contour plot (left) and 3D plot (right) of the two-dimensional Sphere function.

```
library(ggplot2)
pl <- autoplot(fn2, use.facets = TRUE) # basic call
pl + ylim(c(0, 35)) + facet_grid(. ~ disc) # (one column per discrete value)
```

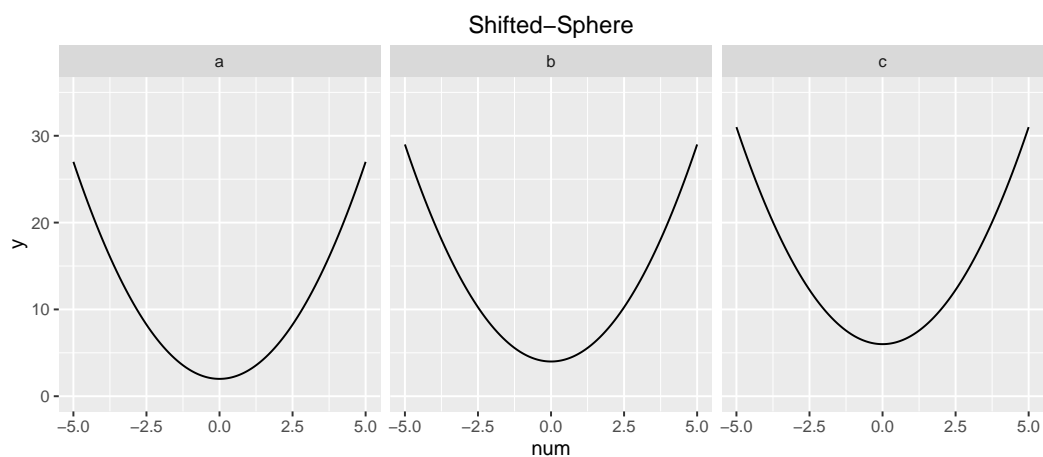


Figure 2: Plot of the mixed decisions space shifted Sphere function.

In particular, due to the possibility to subsequently modify the `ggplot` objects returned by the `autoplot` function it can be used effectively in other packages to, e. g., visualize an optimization process. For instance the `ecr` package makes extensive use of `smoof` functions and the `ggplot2` plots.

Getter methods

Accessing the describing attributes of a `smoof` function is essential and can be simply realized by `attr("attrName", fn)` or alternatively via a set of helper functions. The latter approach is highly recommended. By way of example the following listing shows just a few of the available helpers.

```
> getGlobalOptimum(fn)$param
  x1 x2
1  0  0
> getGlobalOptimum(fn)$value
[1] 0
> getGlobalOptimum(fn)$is.minimum
[1] TRUE
> getNumberOfParameters(fn)
[1] 2
```

```

> getNumberOfObjectives(fn)
[1] 1
> getLowerBoxConstraints(fn)
  x1  x2
-10 -10

```

Predefined test function generators

Extending **smoof** with custom objective functions is nice to have, but the main benefit in using this package is the large set of preimplemented functions typically used in the optimization literature. At the moment of writing there are in total 72 single objective functions and 24 multi-objective function generators available. Moreover there are interfaces to some more specialized benchmark sets and problem generators which will be mentioned in the next section.

Generators for single-objective test functions

To apply some optimization routines to say the Sphere function you do not need to define it by hand. Instead you can just call the corresponding *generator* function, which has the form `makeFUNFunction` where `FUN` may be replaced with one of the function names. Hence, the Sphere function can be generated by calling `makeSphereFunction(dimensions = 2L)`, where the integer `dimensions` argument defines the dimension of the search space for scalable objective functions, i. e., functions which are defined for arbitrary parameter space dimensions $n \geq 2$. All 72 single-objective functions with their associated tags are listed in Table 1. The tags are based on the test function survey in (Jamil and Yang, 2013). Six functions with very different landscapes are visualized in Figure 3.

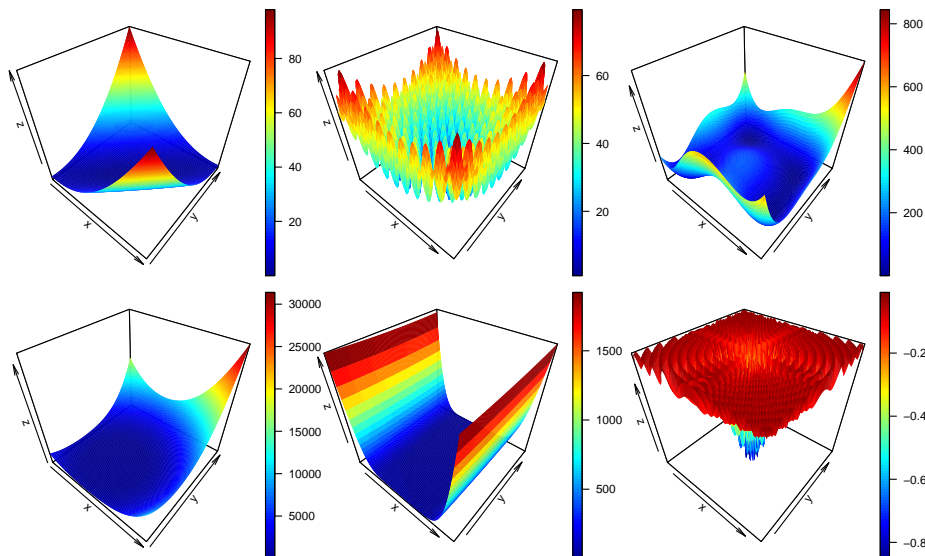


Figure 3: Two-dimensional test functions *Matyas* (top left), *Rastrigin* (top middle), *Himmelblau* (top right), *Zetfl* (bottom left), *Three Hump Camel* (bottom middle) and *Generalized Drop Wave* (bottom right).

Beside these functions there exist two additional single-objective generators, which interface special test function sets or function generators.

BBOB The 24 Black-Box Optimization Benchmark (BBOB) 2009 (Hansen et al., 2009) functions can be created with the `makeBBOBFunction(fid, iid, dimension)` generator, where `fid` $\in \{1, \dots, 24\}$ is the function identifier, `iid` is the instance identifier and `dimension` the familiar argument for specifying the parameter space dimension.

MPM2 The problem generator *multiple peaks model 2* (Wessing, 2015) is accessible via the function `makeMPM2Function`. This problem generator produces multimodal problem instances by combining several randomly distributed peaks (Wessing, 2015). The number of peaks can be set via the `n.peaks` argument. Further arguments are the problem dimension, an initial seed for the random numbers generator and the topology, which accepts the values 'random' or 'funnel' respectively. For details see the technical report of the multiple peaks model 2 (Wessing (2015)).

Function	Tags
Ackley	continuous, multimodal, differentiable, non-separable, scalable
Adjiman	continuous, differentiable, non-separable, non-scalable, multimodal
Alpine N. 1	continuous, non-differentiable, separable, scalable, multimodal
Alpine N. 2	continuous, differentiable, separable, scalable, multimodal
Aluffi-Pentini	continuous, differentiable, non-separable, non-scalable, unimodal
Bartels Conn	continuous, non-differentiable, non-separable, non-scalable, multimodal
Beale	continuous, differentiable, non-separable, non-scalable, unimodal
Bent-Cigar	continuous, differentiable, non-separable, scalable, unimodal
Bird	continuous, differentiable, non-separable, non-scalable, multimodal
BiSphere	multi-objective
Bohachevsky N. 1	continuous, differentiable, separable, scalable, multimodal
Booth	continuous, differentiable, non-separable, non-scalable, unimodal
BraninRCOS	continuous, differentiable, non-separable, non-scalable, multimodal
Brent	continuous, differentiable, non-separable, non-scalable, unimodal
Brown	continuous, differentiable, non-separable, scalable, unimodal
Bukin N. 2	continuous, differentiable, non-separable, non-scalable, multimodal
Bukin N. 4	continuous, non-differentiable, separable, non-scalable, multimodal
Bukin N. 6	continuous, non-differentiable, non-separable, non-scalable, multimodal
Carrom Table	continuous, differentiable, non-separable, non-scalable, multimodal
Chichinadze	continuous, differentiable, separable, non-scalable, multimodal
Chung Reynolds	unimodal, continuous, differentiable, scalable
Complex	continuous, differentiable, non-separable, non-scalable, multimodal
Cosine Mixture	discontinuous, non-differentiable, separable, scalable, multimodal
Cross-In-Tray	continuous, non-separable, non-scalable, multimodal
Cube	continuous, differentiable, non-separable, non-scalable, unimodal
Deckers-Aarts	continuous, differentiable, non-separable, non-scalable, multimodal
Deflected Corrugated Spring	continuous, differentiable, non-separable, scalable, multimodal
Dixon-Price	continuous, differentiable, non-separable, scalable, unimodal
Double-Sum	convex, unimodal, differentiable, separable, scalable, continuous
Eason	continuous, differentiable, separable, non-scalable, multimodal
Egg Crate	continuous, separable, non-scalable
Egg Holder	continuous, differentiable, non-separable, multimodal
El-Attar-Vidyasagar-Dutta	continuous, differentiable, non-separable, non-scalable, unimodal
Engvall	continuous, differentiable, non-separable, non-scalable, unimodal
Exponential	continuous, differentiable, non-separable, scalable
Freudenstein Roth	continuous, differentiable, non-separable, non-scalable, multimodal
Generalized Drop-Wave	multimodal, non-separable, continuous, differentiable, scalable
Giunta	continuous, differentiable, separable, multimodal
Goldstein-Price	continuous, differentiable, non-separable, non-scalable, multimodal
Griewank	continuous, differentiable, non-separable, scalable, multimodal
Hansen	continuous, differentiable, separable, non-scalable, multimodal
Himmelblau	continuous, differentiable, non-separable, non-scalable, multimodal
Holder Table N. 1	continuous, differentiable, separable, non-scalable, multimodal
Holder Table N. 2	continuous, differentiable, separable, non-scalable, multimodal
Hosaki	continuous, differentiable, non-separable, non-scalable, multimodal
Hyper-Ellipsoid	unimodal, convex, continuous, scalable
Jennrich-Sampson	continuous, differentiable, non-separable, non-scalable, unimodal
Judge	continuous, differentiable, non-separable, non-scalable, multimodal
Keane	continuous, differentiable, non-separable, non-scalable, multimodal
Kearfott	continuous, differentiable, non-separable, non-scalable, multimodal
Leon	continuous, differentiable, non-separable, non-scalable, unimodal
Matyas	continuous, differentiable, non-separable, non-scalable, unimodal
McCormick	continuous, differentiable, non-separable, non-scalable, multimodal
Michalewicz	continuous, multimodal, scalable
Periodic	continuous, differentiable, non-separable, non-scalable, multimodal
Double-Sum	continuous, differentiable, separable, scalable, unimodal
Price N. 1	continuous, non-differentiable, separable, non-scalable, multimodal
Price N. 2	continuous, differentiable, non-separable, non-scalable, multimodal
Price N. 4	continuous, differentiable, non-separable, non-scalable, multimodal
Rastrigin	multimodal, continuous, separable, scalable
Rosenbrock	continuous, differentiable, non-separable, scalable, multimodal
Schaffer N. 2	continuous, differentiable, non-separable, non-scalable, unimodal
Schaffer N. 4	continuous, differentiable, non-separable, non-scalable, unimodal
Schwefel	continuous, multimodal, scalable
Shubert	continuous, differentiable, non-scalable, multimodal
Six-Hump Camel Back	continuous, differentiable, non-separable, non-scalable, multimodal
Sphere	unimodal, separable, convex, continuous, differentiable, scalable
Styblinski-Tang	continuous, differentiable, non-separable, non-scalable, multimodal
Sum of Different Squares	unimodal, continuous, scalable
Swiler2014	discontinuous, mixed, multimodal
Three-Hump Camel	continuous, differentiable, non-separable, non-scalable, multimodal
Trecanni	continuous, differentiable, separable, non-scalable, unimodal
Zettl	continuous, differentiable, non-separable, non-scalable, unimodal

Table 1: All single objective functions currently available in **smoof** with their corresponding tags.

Generators for multi-objective test functions

Evolutionary algorithms play a crucial role in solving multi-objective optimization tasks. The relative performance of *multi-objective evolutionary algorithms* (MOEAs) is, as in the single-objective case, mainly studied experimentally by systematic comparison of performance indicators on test instances. In the past decades several test sets for multi-objective optimization were proposed mainly by the evolutionary computation community. The **smoof** package offers generators for the DTLZ function family by Deb et al. (Deb et al., 2002), the ZDT function family by Zitzler et al. (Zitzler et al., 2000) and the multi-objective optimization test instances UF1, ..., UF10 of the CEC 2009 special session and competition (Zhang et al., 2009).

The DTLZ generators are named `makeDTLZXFunction` with $X = 1, \dots, 7$. All DTLZ generators need the search space dimension n (argument dimensions) and the objective space dimension p (argument n.objectives) with $n \geq p$ to be passed. DTLZ4 may be passed an additional argument `alpha` with default value 100, which is recommended by Deb et al. (2002). The following lines of code generate the DTLZ2 function and visualize its Pareto-front by running the NSGA-II EMOA implemented in the **mco** package with a population size of 100 for 100 generations (see Figure 4).

```
> fn = makeDTLZ2Function(dimensions = 2L, n.objectives = 2L)
> visualizeParetoOptimalFront(fn, show.only.front = TRUE)
```

ZDT and UF functions can be generated in a similar manner by utilizing `makeZDTXFunction` with $X = 1, \dots, 5$ or `makeUFFunction`.

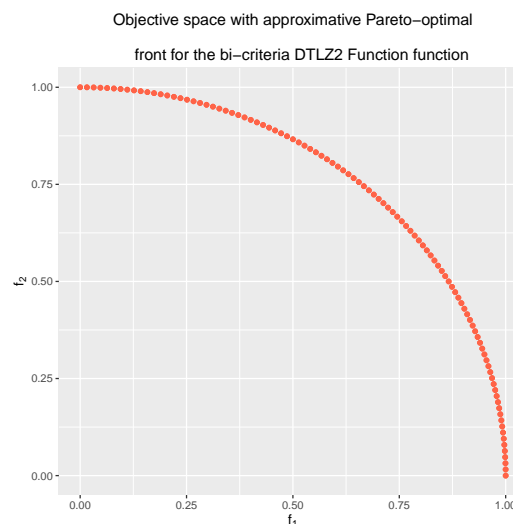


Figure 4: Visualization of the approximated Pareto-front for the DTLZ2 function with two-dimensional search and objective space.

Optimization helpers

In this section we present some additional helper methods which are available in **smoof**.

Filtering and building of test sets

In a benchmark study we most often need not just a single test function, but a set of test functions with certain properties. Say we want to benchmark an algorithm on all multimodal **smoof** functions. Instead of scouring the **smoof** documentation for suitable test functions we can make use of the `filterFunctionsByTags` helper function. This function has only a single mandatory argument, namely a character vector of tags. Hence, to get an overview of all multimodal functions we can write the following:

```
> fn.names <- filterFunctionsByTags(tags = "multimodal")
> head(fn.names)
[1] "Ackley"      "Adjiman"     "Alpine N. 1" "Alpine N. 2" "Bartels Conn"
[6] "Bird"
```

```
> print(length(fn.names))
[1] 46
```

The above shows there are 46 multimodal functions. The next step is to generate the actual **smoof** functions. We could do this by hand, but this would be tedious work. Instead we utilize the `makeFunctionsByName` helper function which comes in useful in combination with filtering. It can be passed a vector of generator names (like the ones returned by `filterFunctionsByTags`) and additional arguments which are passed down to the generators itself. E. g., to initialize all two-dimensional multimodal functions, we can apply the following function call.

```
> fns <- makeFunctionsByName(fn.names, dimensions = 2)
> all(sapply(fns, isSmoofFunction))
[1] TRUE
> print(length(fns))
[1] 46
> print(fns[[1L]])
Single-objective function
Name: 2-d Ackley Function
Description: no description
Tags: single-objective, continuous, multimodal, differentiable, non-separable, scalable
Noisy: FALSE
Minimize: TRUE
Constraints: TRUE
Number of parameters: 2
      Type len Def          Constr Req Tunable Trafo
x numericvector  2  - -32.8,-32.8 to 32.8,32.8  -  TRUE  -
Global optimum objective value of 0.0000 at
  x1 x2
1  0  0
```

Wrapping functions

The **smoof** package ships with some handy wrappers. These are functions which expect a **smoof** function and possibly some further arguments and return a *wrapped* **smoof** function, which behaves as the original and does some secret logging additionally. We can wrap a **smoof** function within a *counting wrapper* (function `addCountingWrapper`) to count the number of function evaluations. This is of particular interest, if we compare stochastic optimization algorithms and the implementations under consideration do not return the number of function evaluations carried out during the optimization process. Moreover, we might want to log each function value along the optimization process. This can be achieved by means of a *logging wrapper*. The corresponding function is `addLoggingWrapper`. By default it logs just the test function values (argument `logg.y` is `TRUE` by default). Optionally the logging of the decision space might be activated by setting the `logg.x` argument to `TRUE`. The following listing illustrates both wrappers by exemplary optimizing the Branin RCOS function with the Nelder-Mead Simplex algorithm.

```
> set.seed(123)
> fn <- makeBraninFunction()
> fn <- addCountingWrapper(fn)
> fn <- addLoggingWrapper(fn, logg.x = TRUE, logg.y = TRUE)
> par.set <- getParamSet(fn)
> lower <- getLower(par.set); upper = getUpper(par.set)
> res <- optim(c(0, 0), fn = fn, method = "Nelder-Mead")
> res$counts[1L] == getNumberOfEvaluations(fn)
[1] TRUE
> head(getLoggedValues(fn, compact = TRUE))
      x1  x2  y1
1 0.00 0.00 55.60
2 0.10 0.00 53.68
3 0.00 0.10 54.41
4 0.10 0.10 52.53
5 0.15 0.15 51.01
6 0.25 0.05 50.22
```

Conclusion and future work

Benchmarking optimization algorithms on a set of artificial test functions with well-known characteristics is an established means of evaluating performance in the optimization community. This article introduces the R package **smoof**, which contains a large collection of continuous test functions for the single-objective as well as the multi-objective case. Besides a set of helper functions is introduced which allows users to log in detail the progress of the optimization algorithm(s) studied. Future work will lay focus on implementing more continuous test functions, introducing test functions with mixed parameter spaces and provide reference Pareto-sets and Pareto-Fronts for the multi-objective functions. Furthermore the reduction of evaluation time by rewriting existing functions in C(++) is planned.

Jakob Bossek
 PhD Student
 Department of Information Systems
 University of Münster
 Germany
bossek@wi.uni-muenster.de

Bibliography

- B. Bischl, M. Lang, J. Bossek, D. Horn, J. Richter, and P. Kerschke. *ParamHelpers: Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning*, 2016. URL <https://github.com/berndbischl/ParamHelpers>. R package version 1.9. [p105]
- J. Bossek. *ecr: Evolutionary Computation in R*, 2017a. URL <https://github.com/jakobbossek/ecr2>. R package version 2.0.0. [p103]
- J. Bossek. *smoof: Single and Multi-Objective Optimization Test Functions*, 2017b. URL <https://github.com/jakobbossek/smoof>. R package version 1.5. [p104]
- C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag, Secaucus, NJ, USA, 2006. doi: 10.1007/978-0-387-36797-2. URL <https://doi.org/10.1007/978-0-387-36797-2>. [p103]
- K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 825–830, 2002. doi: 10.1109/cec.2002.1007032. URL <https://doi.org/10.1109/cec.2002.1007032>. [p104, 110]
- Y. Gonzalez-Fernandez and M. Soto. *cec2005benchmark: Benchmark for the CEC 2005 Special Session on Real-Parameter Optimization*, 2015. URL <http://CRAN.R-project.org/package=cec2005benchmark>. R package version 1.0.4. [p104]
- N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2009. [p104, 108]
- S. Huband, P. Hingston, L. Barone, and R. L. While. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Trans. Evolutionary Computation*, 10(5):477–506, 2006. doi: 10.1109/TEVC.2005.861417. URL <https://doi.org/10.1109/TEVC.2005.861417>. [p104]
- M. Jamil and X. Yang. A literature survey of benchmark functions for global optimisation problems. *IJMNO*, 4(2):150–194, 2013. doi: 10.1504/ijmmno.2013.055204. URL <https://doi.org/10.1504/ijmmno.2013.055204>. [p104, 105, 108]
- P. Kerschke and J. Dagefoerde. *flacco: Feature-Based Landscape Analysis of Continuous and Constraint Optimization Problems*, 2015. URL <http://CRAN.R-project.org/package=flacco>. R package version 1.1. [p104]
- O. Mersmann. *emoa: Evolutionary Multiobjective Optimization Algorithms*, 2012. URL <http://CRAN.R-project.org/package=emoa>. R package version 0.5-0. [p103]
- O. Mersmann. *mco: Multiple Criteria Optimization Algorithms and Related Functions*, 2014. URL <http://CRAN.R-project.org/package=mco>. R package version 1.0-15.1. [p103]
- O. Mersmann and B. Bischl. *soobench: Single Objective Optimization Benchmark Functions*, 2012. URL <http://CRAN.R-project.org/package=soobench>. R package version 1.0-73. [p104]

- K. M. Mullen. Continuous global optimization in R. *JOURNAL of Statistical Software*, 60(6):1–45, 2014. doi: 10.18637/jss.v060.i06. URL <https://doi.org/10.18637/jss.v060.i06>. [p103, 104]
- S. Theussl and H. W. Borchers. Cran task view: Optimization and mathematical programming. version 2015-11-17. <https://cran.r-project.org/web/views/Optimization.html>. [p103]
- S. Wessing. The multiple peaks model 2. Technical Report TR15-2-001, TU Dortmund, 2015. [p104, 108]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. doi: 10.1007/978-0-387-98141-3. URL <https://doi.org/10.1007/978-0-387-98141-3>. [p106]
- M. Zambrano-Bigiarini and Y. Gonzalez-Fernandez. *cec2013: Benchmark Functions for the Special Session and Competition on Real-Parameter Single Objective Optimization at CEC-2013*, 2015. URL <http://CRAN.R-project.org/package=cec2013>. R package version 0.1-5. [p104]
- Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, and W. Liu. Multiobjective optimization test instances for the cec 2009 special session and competition. Technical Report CES-487, School of Computer Science and Electronic Engineering, University of Essex, Colchester, 2009. [p104, 110]
- E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8:173–195, 2000. doi: 10.1162/106365600568202. URL <https://doi.org/10.1162/106365600568202>. [p104, 110]

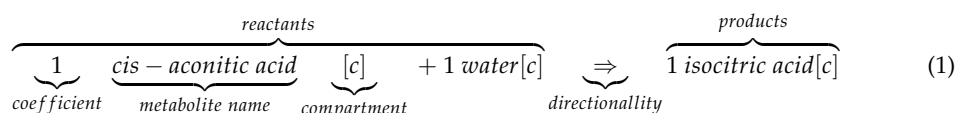
minval: An R package for MINimal VALidation of Stoichiometric Reactions

by Daniel Osorio, Janneth González and Andrés Pinzón

Abstract A genome-scale metabolic reconstruction is a compilation of all stoichiometric reactions that can describe the entire cellular metabolism of an organism, and they have become an indispensable tool for our understanding of biological phenomena, covering fields that range from systems biology to bioengineering. Interrogation of metabolic reconstructions are generally carried through Flux Balance Analysis, an optimization method in which the biological sense of the optimal solution is highly sensitive to thermodynamic unbalance caused by the presence of stoichiometric reactions whose compounds are not produced or consumed in any other reaction (orphan metabolites) and by mass unbalance. The **minval** package was designed as a tool to identify orphan metabolites and evaluate the mass and charge balance of stoichiometric reactions. The package also includes functions to characterize and write models in TSV and SBML formats, extract all reactants, products, metabolite names and compartments from a metabolic reconstruction.

Introduction

A chemical reaction is a process where a set of chemical compounds called *reactants* are transformed into others called *products* (Chen et al., 2013). The accepted way to represent a chemical reaction is called a *stoichiometric reaction*, where reactants are placed on the left and the products on the right separated by an arrow which indicates the direction of the reaction, as shown in equation 1 (Hendrickson, 1997). In biochemistry, a set of chemical reactions that transform a substrate into a product, after several chemical transformations is called a metabolic pathway (Lambert et al., 2011). The compilation of all stoichiometric reactions included in all metabolic pathways that can describe the entire cellular metabolism encoded in the genome of a particular organism is known as a *genome-scale metabolic reconstruction* (Park et al., 2009) and has become an indispensable tool for studying metabolism of biological entities at the systems level (Thiele and Palsson, 2010).



Reconstruction of genome-scale metabolic models starts with a compilation of all known stoichiometric reactions for a given organism, according to the presence of enzyme-coding genes in its genome. The stoichiometric reactions catalyzed by these enzymes are usually downloaded from specialized databases such as KEGG (Kanehisa, 2000), BioCyc (Caspi et al., 2014), Reactome (Croft et al., 2014), BRENDA (Chang et al., 2015) or SMPDB (Jewison et al., 2014). However, the downloaded stoichiometric reactions are not always mass-charge balanced and don't represent complete pathways as to construct a high-quality metabolic reconstruction (Thiele and Palsson, 2010; Gevorgyan et al., 2008). Therefore the identification and curation of these type of reactions is a time-consuming process which the researcher have to complete manually using available literature or experimental data (Lakshmanan et al., 2014).

Genome-scale metabolic reconstructions are usually interrogated through Flux Balance Analysis (FBA), an optimization method that allows us to understand the metabolic status of the cell, to improve the production capability of a desired product or make a rapid evaluation of cellular physiology at genomic-scale (Kim et al., 2008; Park et al., 2009). Nevertheless, FBA method is high sensitive to thermodynamic unbalance, so in order to increase the validity of a biological extrapolation (i.e. an optimal solution) from a FBA analysis it is mandatory to avoid this type of unbalancing in mass conservation through all model reactions (Reznik et al., 2013). Another drawback when determining the validity of a metabolic reconstruction is the presence of reactions with compounds that are not produced or consumed in any other reaction (dead ends), generally known as orphan metabolites (Park et al., 2009; Thiele and Palsson, 2010). The presence of this type of metabolites can be problematic since they lead to an artificial cellular accumulation of metabolism products which generates a bias in the biological conclusions. Tracking these metabolites is also a time-consuming process, which most of the time has to be performed manually or partially automatized by in-house scripting. Given that typical genome-scale metabolic reconstructions account for hundreds or thousands of biochemical reactions, the manual curation of these models is a task that can lead to both, the introduction of new errors and to overlook some others.

Two of the most popular implementations of FBA analysis are **COBRA** (Becker et al., 2007) and **RAVEN** (Agren et al., 2013) which operate as tools under the commercial MATLAB® environment. On the R environment side **sybil** (Gelius-Dietrich et al., 2013) and **abcdeFBA** (Gangadharan and Rohatgi, 2012) are the most common ones. **COBRA** and **RAVEN** include some functions for mass and charge balance (checkMassChargeBalance and getElementalBalance respectively). These functions identify mass unbalanced reactions, based in the chemical formula or the IUPAC International Chemical Identifier (InChI) supplied manually by the user for each metabolite included in the genome-scale metabolic reconstruction.

With the aim of minimizing the manual introduction of thousands of chemical formulas in a genome-scale reconstruction as well as to avoid the sometimes limiting use of licensed software, we have developed the **minval** package. The **minval** package includes twelve functions designed to characterize, check and deuplicate metabolic reconstructions before its interrogation through Flux Balance Analysis (FBA).

To show the potential use of the functions included into the **minval** package, a human-readable model composed by a set of 19 stoichiometric reactions that represent the glycolysis process was included. Glycolysis is the metabolic pathway that converts a molecule of glucose (C₆H₁₂O₆), into two molecules of pyruvate (CH₃COCOO⁻ + H⁺) through a sequence of ten enzyme-catalyzed reactions. Glycolysis occurs in most organisms in the cytosol of the cell and can be summarized as follows: 1 alpha-D-Glucose[c] + 2 NAD+[c] + 2 ADP[c] + 2 Orthophosphate[c] => 2 Pyruvate[c] + 2 NADH[c] + 2 H+[c] + 2 ATP[c] + 2 H2O[c]

Installation and functions

The **minval** package includes twelve functions and is available for download and installation from CRAN, the Comprehensive R Archive Network. To install and load it, just type:

```
> install.packages("minval")
> library(minval)
```

The **minval** package requires an R version 2.10 or higher. Development releases of the package are available in the GitHub repository <http://github.com/gibbslab/minval>.

Inputs and syntaxis

The functions included in **minval** package take as input a set of stoichiometric reactions where the metabolites should be separated by a plus symbol (+) between two blank spaces and may have just one stoichiometric coefficient before the name. The reactants should be separated from products by an arrow using the following symbol => for irreversible reactions and <=> for reversible reactions. The data can be loaded from traditional human-readable spreadsheets through other CRAN-available packages such as **gdata**, **readxl** or **xlsx**. To load the included glycolysis model just type:

```
> glycolysisFile <- system.file("extdata", "glycolysisModel.csv", package = "minval")
> glycolysisModel <- read.csv(file = glycolysisFile,
+                             sep = '\t',
+                             stringsAsFactors = FALSE)
```

Syntax validation

The first step for a metabolic reconstruction validation is to check the syntax of their stoichiometric reactions. The validateSyntax function validate the syntax (Equation 1) of all reactions in a metabolic reconstruction for several FBA implementations (i.e. **COBRA** and **RAVEN**) and returns a boolean value 'TRUE' if the syntax is correct. Syntax validation is a critical step due valid stoichiometric reactions are required to write models in TSV or SBML formats.

```
> validateSyntax(reactionList = glycolysisModel$REACTION)
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[16] TRUE TRUE TRUE TRUE
```

Metabolic models

Metabolic models include additional to the stoichiometric reactions also another information that allows model and interrogates them through FBA, the generally associated information is:

```
> colnames(glycolysisModel)
[1] "ID" "DESCRIPTION" "REACTION" "GPR" "LOWER.BOUND"
[6] "UPPER.BOUND" "OBJECTIVE"
```

Label	Description	Default Value
ID	A list of single character strings containing the reaction abbreviations, Entries in the field abbreviation are used as reaction ids, so they must be unique.	Mandatory
DESCRIPTION	A reaction description	Optional (the column can be empty)
REACTION	A set of stoichiometric reactions with the previously described characteristics.	Mandatory
GPR	A set of genes joined by boolean operators as AND or OR, rules may be nested by parenthesis. GPR rules represent the relationship between genes to synthesize the required enzyme or enzymes to catalyze the stoichiometric reaction.	Optional (the column can be empty)
LOWER.BOUND	A list of numeric values containing the lower bounds of the reaction rates. If not set, zero is used for an irreversible reaction and -1000 for a reversible reaction.	-1000 or 0
UPPER.BOUND	A list of numeric values containing the upper bounds of the reaction rates. If not set, 1000 is used by default.	1000
OBJECTIVE	A list of numeric values containing objective values (0 or 1) for each reaction	0 or 1

SBML files

The standard format to share and store biological processes such as metabolic models is the Systems Biology Markup Language (**SBML**) format. The **minval** package includes the `writeSBMLmod` function which is able to write models in SBML format as follows:

```
> writeSBMLmod(modelData = glycolysisModel,
+             modelID = "Glycolysis",
+             outputFile = "glycolysis.xml")
```

Metabolic models in SBML format can be readed through the `readSBMLmod` function of the **sybilSBML** R package:

```
> glycoModel <- sybilSBML::readSBMLmod("glycolysis.xml")
> glycoModel
model name: Glycolysis
number of compartments 2
c
b
number of reactions: 19
number of metabolites: 18
number of unique genes: 22
objective function: +1 R00200
```

After load the metabolic model, it can be interrogated through FBA using the `optimizeProb` function of the **sybil** R package. In this case, the reaction 'R00200' was set as the objective function. The 'R00200' reaction describes the production of pyruvate from phosphoenolpyruvate, an alpha-D-Glucose derivate.

```
> sybil::optimizeProb(glycoModel)
solver: glpkAPI
method: simplex
algorithm: fba
number of variables: 19
number of constraints: 18
return value of solver: solution process was successful
solution status: solution is optimal
value of objective function (fba): 6.000000
value of objective function (model): 6.000000
```

The interrogated glycolysis model estimates a production of six molecules of pyruvate by each alpha-D-Glucose molecule, probably due a mass unbalance in their stoichiometric reactions. FBA methods are sensitive to thermodynamic (mass-charge) unbalance, so in order to achieve a valid biological extrapolation is mandatory to avoid this type of unbalancing in all model reactions.

Mass-Charge balance validation

The second step for a metabolic reconstruction validation is to check the stoichiometric reactions mass-charge balance. In a balanced stoichiometric reaction according to the *Lomonosov-Lavoisier* law, the mass comprising the reactants should be the same mass present in the products. This process requires the use of a reference with chemical formulas, molecular weights and/or net charges for each metabolite included in the metabolic model.

Reference values for each metabolite can be manually provided or downloaded through the `downloadChEBI` function included into the **minval** package from the Chemical Entities of Biological Interest (ChEBI) database, a freely available dictionary of molecular entities focused on 'small' chemical compounds involved in biochemical reactions. To download the latest version of the ChEBI database just type:

```
> ChEBI <- downloadChEBI(release = "latest",
+                         woAssociations = TRUE)
```

The `checkBalance` function included into the **minval** package can test mass-charge balance using a user-given reference of formulas, masses or charges. The `checkBalance` function returns a boolean value 'TRUE' if stoichiometric reaction is balanced. For this example an user provided reference was used.

```
> # Loading reference
> chemicalData <- read.csv2(file = system.file("extdata", "chemData.csv",
+                                             package = "minval"))
> head(chemicalData, n= 5)
```

	NAME	FORMULA	MASS	CHARGE
1	H2O	H2O	18.0106	0
2	H+	H	1.0078	1
3	ATP	C10H16N5O13P3	506.9957	0
4	NAD+	C21H28N7O14P2	664.1169	1
5	3-Phospho-D-glyceroyl phosphate	C3H8O10P2	265.9593	0

```
> # Mass-Balance evaluation
> checkBalance(reactionList = glycolysisModel$REACTION,
+             referenceData = chemicalData,
+             ids = "NAME",
+             mFormula = "FORMULA")
[1] TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[13] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

As is shown above, the third stoichiometric reaction is mass-unbalanced. It can be corrected replacing manually the unbalanced reaction by a balanced one as follows:

```
> glycolysisModel$REACTION[3] <- "D-Glyceraldehyde 3-phosphate[c] + Orthophosphate[c] +
+ NAD+[c] <=> 3-Phospho-D-glyceroyl phosphate[c] + NADH[c] + H+[c]"
```

And mass-balance can be tested again, in this case using the molecular mass of each metabolite as reference:

```
> checkBalance(reactionList = glycolysisModel$REACTION,
+             referenceData = chemicalData,
+             ids = "NAME",
+             mWeight = "MASS")
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[16] TRUE TRUE TRUE TRUE
```

When all stoichiometric reactions are mass-balanced, then the model can be exported and loaded to be interrogated again:

```
> writeSBMLmod(modelData = glycolysisModel,
+             modelID = "GlycolysisBalanced",
+             outputFile = "glycolysisBalanced.xml")
> sybil::optimizeProb(sybilSBML::readSBMLmod("glycolysisBalanced.xml"))
```

```

solver:                glpkAPI
method:                simplex
algorithm:             fba
number of variables:   19
number of constraints: 18
return value of solver: solution process was successful
solution status:       solution is optimal
value of objective function (fba): 2.000000
value of objective function (model): 2.000000

```

As shown above, the correct mass-charge balance allows predicting in an accurate way the net yield of pyruvate from an alpha-D-glucose molecule through the glycolytic pathway using FBA analysis.

Characterize model

A metabolic reconstruction generally includes three types of reactions *compartmentalized*, *transport* and *exchange* reactions. The *compartmentalized reactions* are those in where all involved metabolites (all reactants and products) are assigned to the same compartment. e.g. $h[m] + nadph[m] + o2[m] + 25hvitd2[m] \Rightarrow h2o[m] + nadp[m] + 1a25dhvitd2[m]$. The *transport reactions* are those in where the involved metabolites are assigned to two or more compartments. e.g. $2 hco3[e] + na1[e] \Leftrightarrow 2 hco3[c] + na1[c]$, and finally, the *exchange reactions* are those used to import or release metabolites to the boundary. e.g. $acetone[e] \Leftrightarrow$. Characterize the stoichiometric reactions of a metabolic model is a required and time-consuming work. The **minval** package includes the `characterizeReactions` function to characterize the stoichiometric reactions and metabolites by type and compartment. This function counts the number of reactions, computes the relative frequency of each reaction type (transport, exchange and compartmentalized), computes the relative frequency of reactions by compartment, counts the number of unique metabolites and computes the relative frequency of metabolites by compartment. The `characterizeReactions` function returns all these information as a labeled list. To show its potential use, the RECON 2.04 Human Metabolic Reconstruction (Thiele et al., 2013) was included in a human-readable format. To load and characterize it just type:

```

> # Loading the Human Metabolic Reconstruction RECON 2.04
> RECON <- read.csv(system.file("extdata", "rRECON2.csv",
+                             package = "minval"))
> # Characterizing the stoichiometric reactions
> charRECON <- characterizeReactions(reactionList = RECON$REACTION)
> charRECON
$Reactions
[1] 7441

$rType

Compartmentalized reaction      Exchange reaction
                    55.825830                9.420777
Transport reaction
                    34.753393

$Reaction

      c      e      g      l      m      n      r      x
24.593469 1.760516 3.628545 2.983470 9.958339 1.666443 6.208843 5.026206

$Metabolites
[1] 5063

$Metabolites

      c      e      g      l      m      n      r      x
37.092633 12.680229 6.261110 5.964843 14.892356 3.258937 11.258147 8.591744

```

Computed values can be easily plotted as follows:

```

> # Combining two plots into one overall graph
> par(mfrow=c(1,2))
> # Plotting reactions by Type

```

```

> pie(x = charRECON$rType,
+     main = "Reactions by Type")
> # Plotting reactions by Compartment
> pie(x = charRECON$cReaction,
+     main = "Reactions by Compartment",
+     labels = compartmentNames)

```

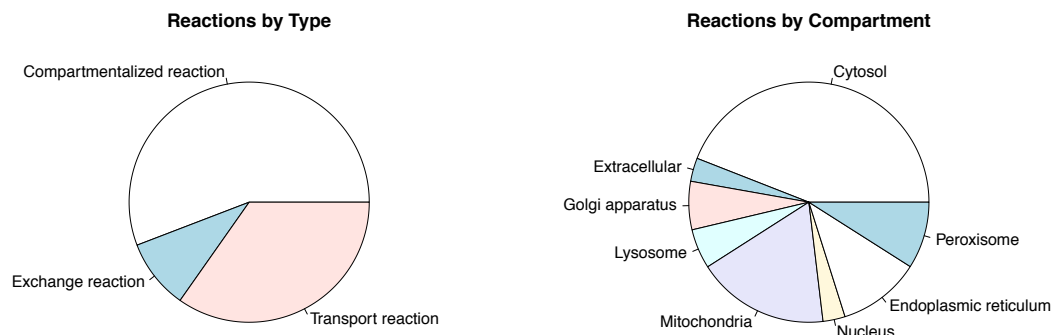


Figure 1: Distribution by type (left) and by compartments (right) of the reactions included into the RECON 2.04 Human Metabolic Reconstruction (Thiele et al., 2013).

Stoichiometric matrix

A metabolic reconstruction is often represented in a more compact form called the stoichiometry matrix (S). If a metabolic reconstruction has n reactions and m participating metabolites, then the stoichiometry matrix will have correspondingly m rows and n columns. Values in the stoichiometric matrix represent the metabolite coefficients in each reaction. To generate the stoichiometric matrix of a metabolic reconstruction just type:

```

> stoichiometricMatrix(reactionList = glycolysisModel$REACTION)

```

metabolites	reactions										
	R01	R02	R03	R04	R05	R06	R07	R08	R09	R10	
2-Phospho-D-glycerate[c]	-1	0	0	0	0	-1	0	0	0	0	
Phosphoenolpyruvate[c]	1	0	0	0	0	0	0	0	0	-1	
H2O[c]	1	0	0	0	0	0	0	0	0	0	
D-Glyceraldehyde 3-phosphate[c]	0	-1	-1	1	0	0	0	0	0	0	
Glycerone phosphate[c]	0	1	0	1	0	0	0	0	0	0	
Orthophosphate[c]	0	0	-1	0	0	0	0	0	0	0	
NAD+[c]	0	0	-1	0	0	0	0	0	0	0	
3-Phospho-D-glyceroyl phosphate[c]	0	0	1	0	1	0	0	0	0	0	
NADH[c]	0	0	1	0	0	0	0	0	0	0	
H+[c]	0	0	1	0	0	0	0	0	0	0	
beta-D-Fructose 1,6-bisphosphate[c]	0	0	0	-1	0	0	0	0	1	0	
ATP[c]	0	0	0	0	-1	0	-1	0	-1	1	
3-Phospho-D-glycerate[c]	0	0	0	0	-1	1	0	0	0	0	
ADP[c]	0	0	0	0	1	0	1	0	1	-1	
alpha-D-Glucose[c]	0	0	0	0	0	0	-1	0	0	0	
alpha-D-Glucose 6-phosphate[c]	0	0	0	0	0	0	1	-1	0	0	
beta-D-Fructose 6-phosphate[c]	0	0	0	0	0	0	0	1	-1	0	
Pyruvate[c]	0	0	0	0	0	0	0	0	0	1	

metabolites	reactions								
	R11	R12	R13	R14	R15	R16	R17	R18	R19
2-Phospho-D-glycerate[c]	0	0	0	0	0	0	0	0	0
Phosphoenolpyruvate[c]	0	0	0	0	0	0	0	0	0
H2O[c]	-1	0	0	0	0	0	0	0	0
D-Glyceraldehyde 3-phosphate[c]	0	0	0	0	0	0	0	0	0
Glycerone phosphate[c]	0	0	0	0	0	0	0	0	0
Orthophosphate[c]	0	0	0	-1	0	0	0	0	0
NAD+[c]	0	-1	0	0	0	0	0	0	0
3-Phospho-D-glyceroyl phosphate[c]	0	0	0	0	0	0	0	0	0
NADH[c]	0	0	-1	0	0	0	0	0	0

H+[c]	0	0	0	0	-1	0	0	0	0
beta-D-Fructose 1,6-bisphosphate[c]	0	0	0	0	0	0	0	0	0
ATP[c]	0	0	0	0	0	-1	0	0	0
3-Phospho-D-glycerate[c]	0	0	0	0	0	0	0	0	0
ADP[c]	0	0	0	0	0	0	0	0	-1
alpha-D-Glucose[c]	0	0	0	0	0	0	-1	0	0
alpha-D-Glucose 6-phosphate[c]	0	0	0	0	0	0	0	0	0
beta-D-Fructose 6-phosphate[c]	0	0	0	0	0	0	0	0	0
Pyruvate[c]	0	0	0	0	0	0	0	-1	0

Reactants and products

As described before, stoichiometric reactions represent the transformation of reactants into products in a chemical reaction. The reactants and products functions extract and return all reactants or products respectively in a stoichiometric reaction as a vector. If reaction is irreversible ('=>') then reactants and products are separated and returned afterward as follows:

```
> reactants(reactionList = "ADP[c] + Phosphoenolpyruvate[c] => ATP[c] + Pyruvate[c]")
[1] "ADP[c]"           "Phosphoenolpyruvate[c]"
> products(reactionList = "ADP[c] + Phosphoenolpyruvate[c] => ATP[c] + Pyruvate[c]")
[1] "ATP[c]"           "Pyruvate[c]"
```

In reversible cases ('<=>') all reactants at some point can act as products and *vice versa*, for that reason both functions return all reaction metabolites:

```
> reactants(reactionList = "H2O[c] + Urea-1-Carboxylate[c] <=> 2 CO2[c] + 2 NH3[c]")
[1] "H2O[c]"           "Urea-1-Carboxylate[c]" "CO2[c]"
[4] "NH3[c]"
> products(reactionList = "H2O[c] + Urea-1-Carboxylate[c] <=> 2 CO2[c] + 2 NH3[c]")
[1] "H2O[c]"           "Urea-1-Carboxylate[c]" "CO2[c]"
[4] "NH3[c]"
```

Metabolites

The metabolites function automatically identifies and lists all metabolites (with or without compartments) for a specific or a set of stoichiometric reactions. This list is usually required for programs that perform FBA analysis as an independent input spreadsheet. In this example we show how to extract all metabolites (reactants and products) included in a metabolic reconstruction with and without compartments.

```
> metabolites(reactionList = glycolysisModel$REACTION)
[1] "2-Phospho-D-glycerate[c]"           "Phosphoenolpyruvate[c]"
[3] "H2O[c]"                             "D-Glyceraldehyde 3-phosphate[c]"
[5] "Glycerone phosphate[c]"             "Orthophosphate[c]"
[7] "NAD+[c]"                            "3-Phospho-D-glyceroyl phosphate[c]"
[9] "NADH[c]"                            "H+[c]"
[11] "beta-D-Fructose 1,6-bisphosphate[c]" "ATP[c]"
[13] "3-Phospho-D-glycerate[c]"           "ADP[c]"
[15] "alpha-D-Glucose[c]"                 "alpha-D-Glucose 6-phosphate[c]"
[17] "beta-D-Fructose 6-phosphate[c]"     "Pyruvate[c]"
> metabolites(reactionList = glycolysisModel$REACTION, woCompartment = TRUE)
[1] "2-Phospho-D-glycerate"           "Phosphoenolpyruvate"
[3] "H2O"                             "D-Glyceraldehyde 3-phosphate"
[5] "Glycerone phosphate"             "Orthophosphate"
[7] "NAD+"                            "3-Phospho-D-glyceroyl phosphate"
[9] "NADH"                            "H+"
[11] "beta-D-Fructose 1,6-bisphosphate" "ATP"
[13] "3-Phospho-D-glycerate"           "ADP"
[15] "alpha-D-Glucose"                 "alpha-D-Glucose 6-phosphate"
[17] "beta-D-Fructose 6-phosphate"     "Pyruvate"
```

Orphan metabolites

Those compounds that are not produced or consumed in any other reaction are generally called orphan metabolites, they represent one of the main causes of mass unbalances in metabolic reconstructions

generating dead-ends without flux. The `orphanMetabolites` function extracts all orphan compounds included into a metabolic reconstruction.

```
> orphanMetabolites(reactionList = glycolysisModel$REACTION[noExchange])
[1] "alpha-D-Glucose[c]" "Pyruvate[c]"      "H+[c]"
[4] "H2O[c]"             "NAD+[c]"      "NADH[c]"
[7] "Orthophosphate[c]"
```

Due not all orphans are not consumed and not produced, the `orphanReactants` function, identifies compounds that are not produced internally by any other reaction and should be added to the reconstruction, for instance, as an input exchange reaction following the protocol proposed by [Thiele and Palsson \(2010\)](#).

```
> orphanReactants(reactionList = glycolysisModel$REACTION[noExchange])
[1] "alpha-D-Glucose[c]" "H+[c]"      "H2O[c]"
[4] "NAD+[c]"           "NADH[c]"    "Orthophosphate[c]"
```

By another side, the `orphanProducts` function, identifies compounds that are not consumed internally by any other reaction and should be added to the reconstruction, for instance, as an output exchange (sink) reaction.

```
> orphanProducts(reactionList = glycolysisModel$REACTION[noExchange])
[1] "Pyruvate[c]"      "H+[c]"      "H2O[c]"
[4] "NAD+[c]"          "NADH[c]"    "Orthophosphate[c]"
```

Compartments

As well as in eukaryotic cells, in which not all reactions occur in all compartments, stoichiometric reactions in a metabolic reconstruction can be labeled to be restricted to a single compartment during FBA, by the assignment of a compartment label after each metabolite name. Some FBA implementations require the reporting of all compartments included in the metabolic reconstruction as an independent section of the human-readable input file. In this example, we show how to extract all compartments for all reactions included in the RECON 2.04 Human Metabolic Reconstruction ([Thiele et al., 2013](#)).

```
> compartments(reactionList = RECON$REACTION)
[1] "c" "l" "m" "r" "e" "x" "n" "g"
```

TSV files

Additional to the SBML format, the TSV format is the default input of metabolic models for the `sybil` R package. The TSV format is composed of three text files, following a character-separated (tab by default) value format where each line contains one entry (stoichiometric reaction and associated info). The `writeTSVmod` function can write a metabolic model in a TSV format as follows:

```
> writeTSVmod(modelData = glycolysisModel,
+             modelID = "Glycolysis",
+             outputFile = "glycolysis")
```

Metabolic models in TSV format can be readed through the `readTSVmod` function included in the `sybil` package:

```
> sybil::readTSVmod(prefix = "glycolysis",quoteChar = "\\")
model name:          Glycolysis
number of compartments 1
                    [c]
number of reactions: 19
number of metabolites: 18
number of unique genes: 22
objective function:  +1 R00200
```

Summary

We introduced the `minval` package to check the syntax validity, evaluate the mass-charge balance and extract all orphan metabolites of a set of stoichiometric reactions. Together, this steps represent the minimal validation that should be performed in a genome-scale metabolic reconstruction. Functions to characterize and export metabolic models in SBML and TSV formats as well as to extract all

reactants, products, metabolite names and compartments for a set of stoichiometric reactions were also introduced. Moreover, we also show in a step by step fashion, how this minimal evaluation process of mass balance can avoid an overestimation of the net yield of pyruvate from an alpha-D-glucose molecule when using an unbalanced model of the glycolysis pathway.

Acknowledgements

DO and JG were supported by the Pontificia Universidad Javeriana (Grant ID 5619, 6235, 6371, 6375). We thank to the anonymous reviewers for their helpful comments and suggestions to improve the package.

Bibliography

- R. Agren, L. Liu, S. Shoaie, W. Vongsangnak, I. Nookaew, and J. Nielsen. The RAVEN Toolbox and Its Use for Generating a Genome-Scale Metabolic Model for *Penicillium Chrysogenum*. *PLoS Computational Biology*, 9(3), 2013. URL <https://doi.org/10.1371/journal.pcbi.1002980>. [p115]
- S. A. Becker, A. M. Feist, M. L. Mo, G. Hannum, B. Ø. Palsson, and M. J. Herrgard. Quantitative Prediction of Cellular Metabolism with Constraint-Based Models: The COBRA Toolbox. *Nature protocols*, 2(3):727–38, 2007. URL <https://doi.org/10.1038/nprot.2007.99>. [p115]
- R. Caspi, T. Altman, R. Billington, K. Dreher, H. Foerster, C. A. Fulcher, T. A. Holland, I. M. Kessler, A. Kothari, A. Kubo, M. Krummenacker, M. Latendresse, L. A. Mueller, Q. Ong, S. Paley, P. Subhraveti, D. S. Weaver, D. Weerasinghe, P. Zhang, and P. D. Karp. The MetaCyc Database of Metabolic Pathways and Enzymes and the BioCyc Collection of Pathway/Genome Databases. *Nucleic Acids Research*, 42(D1):D459–D471, 2014. URL <https://doi.org/10.1093/nar/gkr1014>. [p114]
- A. Chang, I. Schomburg, S. Placzek, L. Jeske, M. Ulbrich, M. Xiao, C. W. Sensen, and D. Schomburg. BRENDA in 2015: Exciting Developments in Its 25th Year of Existence. *Nucleic Acids Research*, 43(D1):D439–D446, 2015. URL <https://doi.org/10.1093/nar/gku1068>. [p114]
- W. L. Chen, D. Z. Chen, and K. T. Taylor. Automatic Reaction Mapping and Reaction Center Detection. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(6):560–593, 2013. URL <https://doi.org/10.1002/wcms.1140>. [p114]
- D. Croft, A. F. Mundo, R. Haw, M. Milacic, J. Weiser, G. Wu, M. Caudy, P. Garapati, M. Gillespie, M. R. Kamdar, B. Jassal, S. Jupe, L. Matthews, B. May, S. Palatnik, K. Rothfels, V. Shamovsky, H. Song, M. Williams, E. Birney, H. Hermjakob, L. Stein, and P. D'Eustachio. The Reactome Pathway Knowledgebase. *Nucleic Acids Research*, 42(D1):D472–D477, 2014. URL <https://doi.org/10.1093/nar/gkv1351>. [p114]
- A. Gangadharan and N. Rohatgi. *ABCDE FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package*, 2012. URL <https://CRAN.R-project.org/package=abcdeFBA>. R package version 0.4. [p115]
- G. Gelius-Dietrich, C. J. Fritze-meier, A. A. Desouki, and M. J. Lercher. Sybil – efficient constraint-based modelling in r. *BMC Systems Biology*, 7(1):125, 2013. ISSN 1752-0509. URL <https://doi.org/10.1186/1752-0509-7-125>. [p115]
- A. Gevorgyan, M. G. Poolman, and D. a. Fell. Detection of Stoichiometric Inconsistencies in Biomolecular Models. *Bioinformatics*, 24(19):2245–2251, 2008. URL <https://doi.org/10.1093/bioinformatics/btn425>. [p114]
- J. B. Hendrickson. Comprehensive System for Classification and Nomenclature of Organic Reactions. *Journal of Chemical Information and Computer Sciences*, 37(97):850–852, 1997. URL <https://doi.org/10.1021/ci970040v>. [p114]
- T. Jewison, Y. Su, F. M. Disfany, Y. Liang, C. Knox, A. Maciejewski, J. Poelzer, J. Huynh, Y. Zhou, D. Arndt, Y. Djoumbou, Y. Liu, L. Deng, A. C. Guo, B. Han, A. Pon, M. Wilson, S. Rafatnia, P. Liu, and D. S. Wishart. SMPDB 2.0: Big Improvements to the Small Molecule Pathway Database. *Nucleic Acids Research*, 42(D1):D478–D484, 2014. URL <https://doi.org/10.1093/nar/gkt1067>. [p114]
- M. Kanehisa. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 28(1):27–30, 2000. URL <https://doi.org/10.1093/nar/28.1.27>. [p114]

- H. U. Kim, T. Y. Kim, and S. Y. Lee. Metabolic Flux Analysis and Metabolic Engineering of Microorganisms. *Mol. BioSyst.*, 4(2):113–120, 2008. URL <https://doi.org/10.1039/b712395g>. [p114]
- M. Lakshmanan, G. Koh, B. K. S. Chung, and D.-Y. Lee. Software Applications for Flux Balance Analysis. *Briefings in Bioinformatics*, 15(1):108–122, 2014. URL <https://doi.org/10.1093/bib/bbs069>. [p114]
- A. Lambert, J. Dubois, and R. Bourqui. Pathway Preserving Representation of Metabolic Networks. *Computer Graphics Forum*, 30(3):1021–1030, 2011. URL <https://doi.org/10.1111/j.1467-8659.2011.01951.x>. [p114]
- J. M. Park, T. Y. Kim, and S. Y. Lee. Constraints-Based Genome-Scale Metabolic Simulation for Systems Metabolic Engineering. *Biotechnology Advances*, 27(6):979–988, 2009. URL <https://doi.org/10.1016/j.biotechadv.2009.05.019>. [p114]
- E. Reznik, P. Mehta, and D. Segrè. Flux Imbalance Analysis and the Sensitivity of Cellular Growth to Changes in Metabolite Pools. *PLoS Computational Biology*, 9(8):e1003195, 2013. URL <https://doi.org/10.1371/journal.pcbi.1003195>. [p114]
- I. Thiele and B. Ø. Palsson. A protocol for Generating a High-Quality Genome-Scale Metabolic Reconstruction. *Nature Protocols*, 5(1):93–121, 2010. URL <https://doi.org/10.1038/nprot.2009.203>. [p114, 121]
- I. Thiele, N. Swainston, R. M. Fleming, A. Hoppe, S. Sahoo, M. K. Aurich, H. Haraldsdottir, M. L. Mo, O. Rolfsson, M. D. Stobbe, and others. A community-driven global reconstruction of human metabolism. *Nature biotechnology*, 31(5):419–425, 2013. URL <https://doi.org/10.1038/nbt.2488>. [p118, 119, 121]

Daniel Osorio

Grupo de Investigación en Bioinformática y Biología de Sistemas

Instituto de Genética, Universidad Nacional de Colombia

Bogotá

Colombia

ORCID: 0000-0003-4424-8422

dcosorioh@unal.edu.co

Janneth González

Grupo de Investigación en Bioquímica Experimental y Computacional

Facultad de Ciencias, Pontificia Universidad Javeriana

Bogotá

Colombia

ORCID: 0000-0003-2009-3374

janneth.gonzalez@javeriana.edu.co

Andrés Pinzón

Grupo de Investigación en Bioinformática y Biología de Sistemas

Instituto de Genética, Universidad Nacional de Colombia

Bogotá

Colombia

ampinzonv@unal.edu.co

Working with Daily Climate Model Output Data in R and the `futureheatwaves` Package

by G. Brooke Anderson, Colin Eason, and Elizabeth A. Barnes

Abstract Research on climate change impacts can require extensive processing of climate model output, especially when using ensemble techniques to incorporate output from multiple climate models and multiple simulations of each model. This processing can be particularly extensive when identifying and characterizing multi-day extreme events like heat waves and frost day spells, as these must be processed from model output with daily time steps. Further, climate model output is in a format and follows standards that may be unfamiliar to most R users. Here, we provide an overview of working with daily climate model output data in R. We then present the `futureheatwaves` package, which we developed to ease the process of identifying, characterizing, and exploring multi-day extreme events in climate model output. This package can input a directory of climate model output files, identify all extreme events using customizable event definitions, and summarize the output using user-specified functions.

Introduction

Research on climate change impacts can require extensive processing of climate model output data. This is not only because output files for a single climate model can be large, but also because of the rising popularity of ensemble techniques (Cubasch et al., 2013) in which, to better characterize uncertainty in projections, impacts are assessed for multiple climate models, multiple simulations of each climate model, and multiple climate experiments. Such ensemble techniques help characterize uncertainty in projections of regional climate change over the next century due to three distinct sources: (1) internal climate variability, i.e., climate noise, (2) climate model uncertainty, i.e. the same forcing can produce a different response in different models and (3) scenario uncertainty, i.e., uncertainty in future climate forcings (e.g. Hawkins and Sutton (2009)).

A key source of data for ensemble techniques is the Coupled Model Intercomparison Project, Phase 5 (CMIP5; Taylor et al. (2012)). This project brought together major climate modeling groups around the world to simulate the same future radiative forcing scenarios, but with their own models. This created an ensemble of state-of-the-art climate model projections that allows researchers to study projections and their uncertainties. Most of these modeling groups additionally performed more than one simulation for each scenario and model (i.e. multiple ensemble members), perturbing the initial conditions by a very tiny amount to quantify uncertainties due to internal climate variability.

We begin this article with an overview of CMIP5 climate model output data for R users, focusing on output with a daily time step. We outline where data from CMIP5 can be obtained as well as how to work with the file format (netCDF) from R. We overview some R packages that can be useful when working with this data, as well as aspects of the data (e.g., non-standard calendars) of which users should be aware when working with daily climate model output in R.

After this overview, we present the `futureheatwaves` package, which we created to aid in identifying and characterizing any type of multi-day extreme event from daily climate model output (Table 1). The impacts of multi-day extreme events must be assessed using output in daily time step, unlike other climate impacts that can be assessed using climate model output at monthly, seasonal, or yearly time steps. Further, extreme events are identified based on conditions that are rare for a certain location (e.g., 98th percentile of local temperature distribution for identifying heat waves) (Cubasch et al., 2013). In this case, the event definition must be determined at each study location from climate model output before events can be identified. Finally, it is often of interest to create summaries of multiple characteristics of these extreme events. For example, one may be interested in determining whether the frequency or characteristics (e.g., length, intensity) of heat waves or warm spells will change under certain climate change scenarios (Cubasch et al., 2013).

The `futureheatwaves` package handles these challenges and can be used to identify and characterize a variety of multi-day extreme events across different ensemble members of one or more climate models. It also provides some functionality specifically useful in identifying and characterizing heat waves. Quantification of the impacts of heat waves on human health suffers from additional sources of uncertainty beyond those inherent in projections of regional changes in surface temperature. These include: (1) uncertainty in the definition of a heat wave itself and (2) uncertainty in the ability of communities to adapt to changing temperatures (the adaptation scenario). This package therefore

Design goals of the futureheatwaves package	
1	Make processing of large ensembles of climate simulations more practical for researchers exploring the potential impacts of heat waves and other multi-day extreme events.
2	Speed up processing time by incorporating C++ in event identification.
3	Keep track of the names of climate models and number of ensemble members processed for each.
4	Not only identify, but also characterize, all extreme events within each climate simulation, to allow the exploration of patterns in these characteristics across different simulations and also to allow the use of more complex impacts models, including models that incorporate event characteristics (e.g., event length, event intensity). For example, this package allows the user to apply a health-effects model where risk of mortality is not the same for every heat wave, but rather is modified by heat wave length, intensity, or other measured characteristics.
5	Give users extensive power in customizing the process, including allowing custom extreme event definitions.
6	Allow users to easily explore the extreme events identified within all climate simulations.
7	Create output that is in a “tidy” data format, allowing it to work well with ggplot2 for visualization.

Table 1: Design goals for the **futureheatwaves** package.

allows the user to create and use a custom extreme event definition to identify events in the climate model output, as well as providing options to explore different scenarios of adaptation to heat.

An overview of climate model output for R users

CMIP5 climate model output data

For climate impact studies, a main source of data is the Coupled Model Intercomparison Project, which is currently in its fifth phase (CMIP5). Over 20 climate modeling groups created one or more climate models which, for this project, were run using standardized scenarios (Taylor et al., 2012). The resulting output is uniform across modeling groups and has a consistent structure, which allows comparison of simulations from different models (Flato et al., 2013). CMIP5 climate model output is archived at a number of different time steps (e.g., daily, monthly, seasonal, yearly) (Taylor and Doutriaux, 2010), and some variables are reported at multiple levels in the ocean or atmosphere (e.g., ocean temperature is reported at different ocean depths). Here, we will focus on data with a daily time step for variables reported at a single level (e.g., near-surface air temperature).

Each modeling group ran simulations for CMIP5 under several experiments, with experiments varying in terms of radiative forcing scenarios through the use of different scenarios of time-varying model inputs (greenhouse gas emissions or concentrations, land use changes, etc.) (Taylor et al., 2012; Flato et al., 2013). Experiments include historical experiments (run using radiative forcing consistent with observed and reconstructed data for 1850–2005), pre-industrial control experiments, and experiments of future scenarios of radiative forcing over the 21st century or longer (e.g., RCP4.5, RCP8.5) (Taylor et al., 2012). Some modeling groups created ensembles of output for a specific model and experiment, in which they ran the experiment multiple times with very small changes to the initial conditions.

The CMIP5 climate model output data are distributed across data nodes at different climate modeling centers (Taylor et al., 2012), but can be accessed centrally at the World Climate Research Programme CMIP5 data portal at <https://pcmdi.llnl.gov/search/cmip5/>. Users must register before downloading data, and some data are restricted to non-commercial use. There is a separate file for each combination of climate model, experiment, modeling realm (e.g., atmosphere, ocean), variable, time step, and ensemble member (Taylor et al., 2012; Taylor and Doutriaux, 2010). For finer time scales, the output is further split across multiple files for specific year ranges (e.g., 5 years of output for each file) (Taylor and Doutriaux, 2010). Each file’s name includes the output variable, climate model, experiment, and ensemble member for the simulation (Taylor and Doutriaux, 2010).

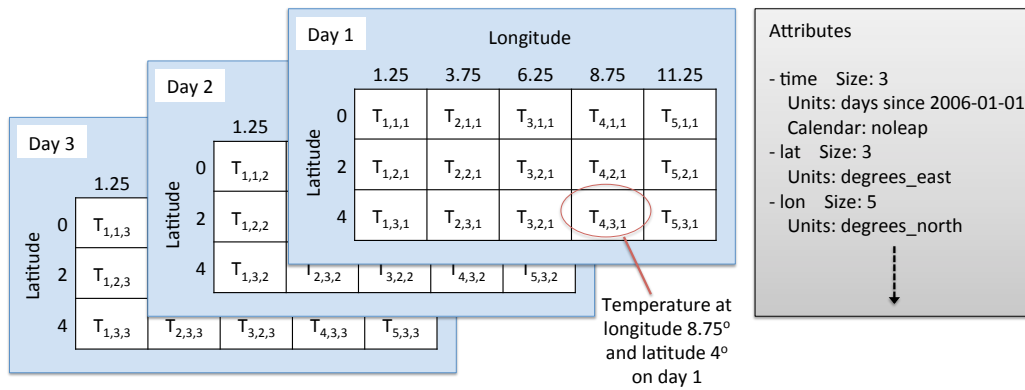


Figure 1: Example structure of a netCDF climate model output file for a variable reported at a single level, like near-surface air temperature. Data are stored in a three-dimensional array, with measurements at each time step and climate grid location. These data are typically indexed in the netCDF file by longitude, latitude, and time, in that order. For example, if the near-surface air temperature in the example netCDF shown here is read into an R object called `tas`, you can access the value for the first day at the fourth longitude and third latitude with `tas[4, 3, 1]`. In addition to the output variable (temperature in this example), vectors with the ordered values of each dimension (longitude, latitude, and time) can also be read in from the netCDF file, as well as attribute data (e.g., units for variables, the calendar used for time).

CMIP5 files can be searched and downloaded through a point-and-click web interface. They can also be downloaded in bulk to computers with Unix or Mac operating systems using the `wget` file downloading utility. Appropriate `wget` scripts can be created either through the World Climate Research Programme CMIP5 data portal or through the Earth System Grid Federation’s Search RESTful API. Tips on efficiently searching and downloading the data, including through the use of `wget` scripts and the search API, are available as user tutorials through the website of the University of Colorado Boulder’s Earth System CoG (e.g., <https://www.earthsystemcog.org/projects/cog/doc/wget> for a tutorial on downloading files using `wget`).

CMIP5 files are saved in Network Common Data Format (netCDF), a binary file format that allows storage of data representing a regular array. For climate model output at a single level (e.g., near-surface air temperature), the data is a 3-dimensional array, with dimensions representing time and two coordinates of location (e.g., latitude and longitude). Figure 1 provides a sketch of the structure of netCDF files for single-level climate model output.

Each data point in the netCDF array gives the modeled value of the variable (e.g., surface temperature) for a single time point and location. Global climate models generate output at regularly-spaced time steps, typically at regularly-spaced grid points around the world. The latitude and longitude spacing of grid points vary by climate model, but are typically 1–2 degrees for atmospheric variables in CMIP5 models (Flato et al., 2013). For CMIP5 climate model output, the location units are in degrees east and degrees north for longitude and latitude, respectively. For daily output files, the time unit is in days since a specified origin date-time (e.g., days since 1850-01-01 00:00:00) (Taylor and Doutriaux, 2010).

All CMIP5 output files are required to include certain metadata (or “attributes”) (Taylor and Doutriaux, 2010), including the experiment, forcing agents input to the model to create the simulation, time step, institution and institutional contact information, climate model, and modeling realm (Taylor and Doutriaux, 2010). The metadata also must include units for all of the dimension variables (e.g., longitude, latitude, time). The netCDF format allows one to access metadata and variables describing the dimensions of the data without reading the full file into memory.

To find out more about the CMIP climate model output data, Taylor et al. (2012) and Meehl et al. (2007) are excellent resources.

Working with climate model output in R

When working with daily climate model output data, challenges to R users include: (1) the file format, (2) use of non-Gregorian calendars, and (3) large file sizes. This section explains these challenges and offers some strategies for dealing with them.

CMIP5 data are available in the netCDF file format. Free specialty software exists to work with

Modeled temperature on a day in July 2075

GFDL-ESM2G model, RCP8.5 experiment, r1i1p1 ensemble member

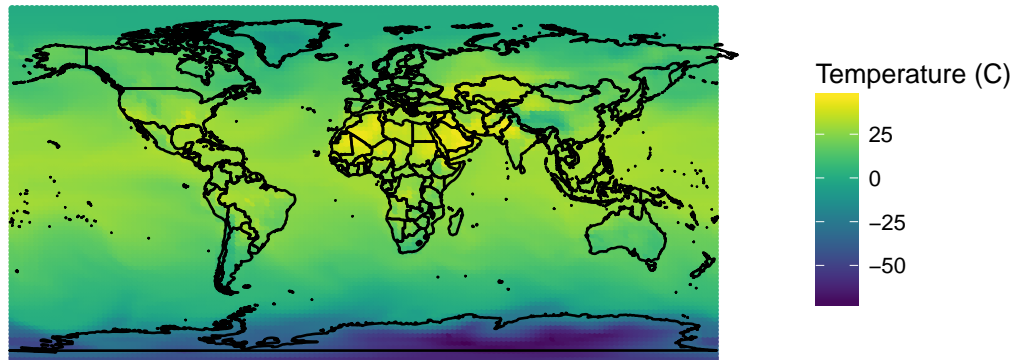


Figure 2: Example of mapping near-surface air temperature data worldwide for a single day of climate model output data. This map uses data from the Geophysical Fluid Dynamics Laboratory’s Earth System Model 2G, r1i1p1 ensemble member, on a single day in the summer of 2075. Full code for recreating the map is available in the “starting_from_netcdf” vignette of the **futureheatwaves** package.

climate model output files in this format, including a collection of command line tools developed by the Max Planck Institute called Climate Model Operators (CDO) (Schulzweida, 2017) and an interpreted language developed by the National Center for Atmospheric Research (NCAR) called the NCAR Command Language (NCL) (UCAR/NCAR Computational and Information Systems Laboratory, 2017). Although such software can be used to quickly process netCDF climate model output files, they require learning a new language syntax, and so for R users may not be worth the computational speed gain compared to alternative solutions that can be scripted in the R language. While base R import functions do not exist for netCDF files, there are a few R package extensions that allow R users to work with the netCDF file format used for CMIP5 files directly from R. Older packages include **ncdf** and **ncvar**, but these do not work with the newer netCDF version 4 released in 2008 and are no longer available through CRAN. More recent packages, including **ncdf4** (Pierce, 2015) and **RNetCDF** (Michna and Woods, 2013, 2016), work with both version 4 and netCDF’s older version 3. While climate model output data for CMIP5 are required to conform with the earlier version (version 3) (Taylor and Doutriaux, 2010), it is safer to write code using functions that can be used with either version, in case future phases of CMIP do not require files to conform with netCDF version 3.

You can do a number of things with netCDF files in R using these packages. For example, **ncdf4**’s `nc_open` function can be used to open a connection to a netCDF file; the object returned by the function includes the file’s attribute data. Once a file connection is open, variables can be read in using the `ncvar_get` function. For example, the variables defining the dimensions of the sketched netCDF file in Figure 1 could be read into R with `ncvar_get` with the `varid` parameter set to “lat”, “lon”, or “time”.

The climate output variable (e.g., near-surface air temperature) can similarly be read in using `ncvar_get`. In this case, the `varid` parameter should be set using the appropriate CMIP5 variable name (e.g., “tas” for near-surface air temperature); these variable names can be found in the CMIP requested output tables (Taylor and Doutriaux, 2010). If only a subset of the full file is needed, the dimensional time and location data can be used to identify the location of the needed data in the netCDF array and this information can then be used to read a portion of data into memory (e.g., with the `nc.get.var.subset.by.axes` function in **ncdf4.helpers**). Once the user is done reading in data from the file, the connection to the netCDF should be closed (e.g., with the `nc_close` function from **ncdf4**).

A second challenge when working with climate model output data in R is that some climate models output to non-Gregorian calendars. Since the late 1500s, Western dates have been set using the Gregorian calendar, which has 365.2425-day years. Some climate models, however, are run using different calendars, including the Julian calendar (365.25-day years), a calendar where there are no leap years (365-day years), a calendar where every year is a leap year (366-day years), and a calendar of twelve 30-day months (360-day years) (Eaton et al., 2011). With these non-Gregorian calendars, R’s base functions for converting a vector to a Date class based on the number of days since an origin date (`as.Date`, `as.POSIXct`) do not return the desired values.

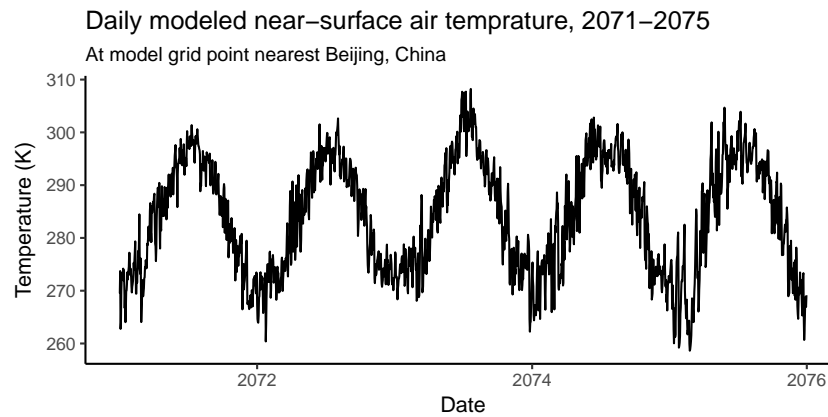


Figure 3: Example of plotting a time series of temperature simulations between 2071 and 2075 from CMIP5 daily climate model output data for the model grid cell point closest to Beijing, China. This plot uses data from the Geophysical Fluid Dynamics Laboratory’s Earth System Model 2G, r1i1p1 ensemble member. Full code for recreating the map is available in the “starting_from_netcdf” vignette of the `futureheatwaves` package.

Two R packages provide help with non-Gregorian calendars: `PCICt` (Bronaugh and Drepper, 2013) and `ncdf4.helpers` (Bronaugh, 2014). The `nc.get.time.series` function in `ncdf4.helpers` pulls and uses metadata on the calendar stored in the CMIP5 netCDF file’s attributes to convert the “time” variable in the file to an object of the `PCICt` class. This class is defined in the `PCICt` package and provides date-like functionality for 360- and 365-day calendars (Bronaugh and Drepper, 2013). However, while these functions will help with handling most CMIP5 files, the CMIP5 standards allows use of other calendars which may not be successfully handled by these functions, so it is important to assess whether the time variable range in the `PCICt` object correctly matches the expected date ranges for a file when processing CMIP5 data in R. While most CMIP5 climate models use the same calendar for all experiments, a few do not; a full table of the calendars used for each climate model and experiment, pulled from netCDF metadata, is available at https://www.earthsystemcog.org/projects/cog/faq_data.

Finally, the size of CMIP5 files can make them difficult to work with in R. CMIP5 climate model output files can be as large as several gigabytes. The size of the files can therefore be large enough that it may make more sense to work with smaller chunks of the data in R, rather than reading all data into memory and working with the data all at once (Todd-Brown and Bond-Lamberty, 2016). This problem aggregates when working with multiple climate models and more than one ensemble member for each of those climate models.

In addition to these general packages for working with netCDF files, there are several R packages specifically for working with climate model output data, including `RCMIP5` (Todd-Brown and Bond-Lamberty, 2016) and `wux` (Mendlik et al., 2016). However, these packages are more useful for working with data output at time steps of a month or higher and have limited utility with the daily climate model output data required for studies of multi-day extreme events.

The `RCMIP5` package includes functions to read in CMIP5 data from netCDF files, scan a directory of CMIP5 files and determine models with continuous available data, create objects of a special `cmip5data` class to work with CMIP5 data within R, and parse the file names for all files in a directory to extract information within the file name. For this package, most functions only work with monthly or less frequent data (Todd-Brown and Bond-Lamberty, 2016). While the `loadCMIP5` function does successfully load daily data as a `cmip5data` object, most of the methods for this object type do not do anything meaningful for daily data. The package’s `getFileInfo` function, however, will work with CMIP5 files of any time step; this function identifies all CMIP5 files in a directory and creates a data frame with information parsed from the file name. The `get.split.filename.cmip5` function in the `ncdf4.helper` package similarly can be used to parse information contained in CMIP5 file names (Bronaugh, 2014).

The `wux` package (Mendlik et al., 2016) includes functions that allow the user to download CMIP5 output at a monthly time step directly from R with the `CMIP5fromESGF` function. The package then uses the `model2wux` function to read climate model output netCDF files and convert it to “WUX” data frames, which can be used by other functions in the package. While this function can input climate model output with daily time steps (the “what.timesteps” element of the `modelinput` list input must be set to “daily”), the function aggregates this data to a monthly or less frequent (e.g., seasonal)

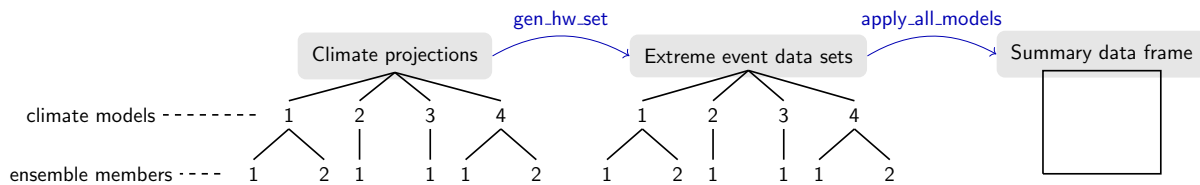


Figure 4: Overview of the functionality of the **futureheatwaves** package. The package takes a directory with climate projection files (left), for one or more climate models, with one or more ensemble members for each climate model (this example figure shows four climate models with one or two ensemble members each). The `gen_hw_set` function processes these files to create a data frame for each ensemble member, identifying and characterizing all multi-day extreme events (e.g., heat waves) in the time series projection for that ensemble member. The `apply_all_models` function allows users to explore these extreme events by applying user-created functions across all the extreme event data frames, creating a summary data frame with results.

aggregation when creating the WUX data frame. Therefore, while this package provides very useful functionality for working with averaged output of daily climate model output data or with output data at a larger time step, it cannot easily be used to identify and characterize multi-day extreme events like heat waves.

The functions and packages described in this section can be used with CMIP5 netCDF files to do things in R like map near-surface air temperatures from a single climate model on a specific day (Figure 2) or pull a time series of daily near-surface air temperature simulations at a specific climate model grid point (Figure 3). The **futureheatwaves**’ “Starting from netCDF files” vignette (https://cran.r-project.org/web/packages/futureheatwaves/vignettes/starting_from_netcdf.html) provides all code required to create these figures, as well as more details and code examples on working with CMIP5 netCDF files in R.

The futureheatwaves package

Motivation

We created the **futureheatwaves** package to aid in identifying, characterizing, and exploring multi-day extreme events in daily climate model output data. While most of the discrete tasks involved in identifying and characterizing multi-day extreme events are fairly straightforward, the full process can be code-intensive, especially for multi-city studies, studies that test sensitivity to how an event is defined, or studies that incorporate different scenarios of adaptation in the case of events defined using a threshold relative to community climate. Our aim in developing this package was therefore to make the full process of identifying and characterizing these extreme events much more convenient and so facilitate the use of multi-model, multi-ensemble member analyses in climate impact studies conducted by non-climate scientists.

How the package works

Figure 4 gives an overview of the two primary functions of the **futureheatwaves** package. First, the `gen_hw_set` function processes a directory of climate projection files that are stored locally on the user’s computer (Figure 4, “Climate projections”), to generate a list of all extreme events in each projection, as well as over a dozen characteristics of each identified extreme event (Table 2). This package starts from files rather than R objects to avoid loading data from all climate model ensembles at once; instead, the function loads, processes, and saves output for a single climate model ensemble member at a time. The extreme events are identified and characterized at one or more study locations (e.g., cities), which the user specifies in an input file. The extreme events identified for each ensemble member are output as separate files in a directory specified by the user (Figure 4, “Extreme events datasets”).

Once the user creates these data frames of location-specific extreme events, the `apply_all_models` function can be used to apply custom functions across all the extreme event data frames. This functionality allows users to create summaries of extreme events across all climate models and ensemble members (Figure 4, right). The function can be used to generate summary statistics (e.g., determine average heat wave length or total frost days) or to apply more complex functions (e.g., apply epidemiologic effect estimates across the heat waves to generate health impact estimates).

When using this package, CMIP5 climate model output data require some pre-processing. Data will need to be saved in a specific format, with files stored in a specific directory structure. Full details of the required file and directory structure are provided in the package's main vignette (<https://cran.r-project.org/web/packages/futureheatwaves/vignettes/futureheatwaves.html>), while tips and an R script for conducting this processing starting from CMIP5 netCDF files are given in the "Starting from netCDF files" vignette.

This package can be used for study locations worldwide. The "Starting from netCDF files" package vignette provides an example of using this package to identify and explore future heat waves in several Chinese cities.

Example data

We have included data files in the package to serve as example files so that users can try this package before applying it to their own directory of climate projection files. These example data come from two climate models that are a part of CMIP5: (1) the model of the Beijing Climate Center, China Meteorological Administration (BCC) (Xin et al., 2013) and (2) the National Center for Atmospheric Research's (NCAR's) Community Climate System Model, version 4 (CCSM4) (Gent et al., 2011). We include one ensemble member from BCC (r1i1p1) and two from CCSM (r1i1p1 and r2i1p1). Once the **futureheatwaves** package is installed and loaded, the user can find the location of these files on his or her computer using R's `system.file` function.

To ensure that the size of this example data is reasonably small, we have only included projection data for grid points from these climate models that are near five U.S. east coast cities: New York, NY; Philadelphia, PA; Newark, NJ; Baltimore, MD; and Providence, RI. Further, to keep the file sizes reasonably small, the historical projections range over the years 1990 to 1999, while the future projections are limited to 2060 to 2079. Users' applications of this package will likely use directories with many more climate model ensemble members and more locations; however, the operation of the package is the same for this smaller example application, as it would be for a much larger application.

Basic example of using futureheatwaves

Once climate model output files are set up, as specified in the "futureheatwaves" package vignette, the package can process them to identify and characterize heat waves in each ensemble member's projection for each location using the `gen_hw_set` function. For example, to process the example climate model output data included with the package, the user can run:

```
library(futureheatwaves)
projection_dir_location <- system.file("extdata/cmip5",
                                       package = "futureheatwaves")
city_file_location <- system.file("extdata/cities.csv",
                                  package = "futureheatwaves")

gen_hw_set(out = "example_results",
           dataFolder = projection_dir_location ,
           dataDirectories = list("historical" = c(1990, 1999),
                                 "rcp85" = c(2060, 2079)),
           citycsv = city_file_location,
           coordinateFileNames = "latitude_longitude_NorthAmerica_12mo.csv",
           tasFileNames = "tas_NorthAmerica_12mo.csv",
           timeFileNames = "time_NorthAmerica_12mo.csv")
```

This code first identifies and saves as objects the path names on the user's computer of the example climate projections directory (`projection_dir_location`) and the file of study locations (`city_file_location`). The `gen_hw_set` function processes the example input and creates a new directory, 'example_results', with files of identified and characterized heat waves, in the user's current working directory. In this example code, the processing is done using default values for the event definition, adaptation scenario, etc. How and why to customize these choices are explained later in the text. Function arguments (e.g., `dataDirectories`, `tasFileNames`) are used to specify the format of the data and the directory structure.

Once the function has completed running, results will be written locally to the directory specified by the `out` argument of `gen_hw_set`. This directory will include files with some basic information about the climate models and the closest grid points of each climate model to each location, as well as a directory with files of identified and classified extreme events for each ensemble member, including

Column name	Description of characteristic
mean.var	Average daily value of the variable across all days in the extreme event, in the units in which the variable is expressed in input files (e.g., average daily mean temperature during the heat wave in degrees Kelvin)
max.var	Highest daily value of the variable across all days in the extreme event, in the units in which the variable is expressed in input files
min.var	Lowest daily value of the variable across all days in the extreme event, in the units in which the variable is expressed in input files
length	Number of days in the event
start.date	Date of the first day of the event
end.date	Date of the last day of the event
start.doy	Day of the year of the first day of the event (1 = Jan. 1, etc.)
start.month	Month in which the event started (1 = January)
days.above.abs.thresh.1	Number of days in the event above a specified absolute threshold (default is the number of days in the event above 80°F / 26.7°C, but this and the following three absolute thresholds can be changed with the absolute_thresholds argument in gen_hw_set)
days.above.abs.thresh.2	Number of days in the event above a specified absolute threshold (default is the number of days in the event above 85°F / 29.4°C)
days.above.abs.thresh.3	Number of days in the event above a specified absolute threshold (default is the number of days in the event above 90°F / 32.3°C)
days.above.abs.thresh.4	Number of days in the event above a specified absolute threshold (default is the number of days in the event above 95°F / 35.0°C)
days.above.99th	Number of days in the event above the 99 th percentile of the variable for the location, using the period specified with the referenceBoundaries argument in gen_hw_set as a reference for determining these percentiles
days.above.99.5th	Number of days in the event above the 99.5 th percentile of the variable for the location, using the period specified with the referenceBoundaries argument in gen_hw_set as a reference for determining these percentiles
first.in.year	Whether the event was the first to occur in its calendar year in the location
mean.var.quantile	The percentile of the average variable value during the event compared to the location's year-round distribution of the variable, based on the variable distribution for the location during the period specified by the referenceBoundaries argument in gen_hw_set
max.var.quantile	The percentile of the maximum variable value during the event compared to the location's year-round distribution of the variable, based on the variable distribution for the location during the period specified by the referenceBoundaries argument in gen_hw_set
min.var.quantile	The percentile of the minimum variable value during the event compared to the location's year-round distribution of the variable, based on the variable distribution for the location during the period specified by the referenceBoundaries argument in gen_hw_set
mean.seasonal.var	The location's average seasonal value of the variable (by default, season is set to May–September, but this can be changed with the seasonal_months argument in gen_hw_set), based on the variable values for the location during the years specified by the referenceBoundaries argument in gen_hw_set
mean.yearround.var	The location's average year-round value of the variable, based on the variable values for the location during the years specified by the referenceBoundaries argument in gen_hw_set

Table 2: Extreme event characteristics measured by the `gen_hw_set` function in the `futureheatwaves` package. The left column gives the name of each variable's column in the extreme event data frames created by the `gen_hw_set` function. When characterizing extreme events below a threshold, like cold spells, appropriate alternatives are given for some columns (e.g., `days.below.abs.thresh.1`, `days.below.1st`).

all characteristics in Table 2. See the package's vignettes for more details on the content and structure of this output.

Once users have created a directory of characterized event files for each ensemble member ("Extreme event data sets", Figure 4), they can explore the results using the `apply_all_models` function. This function allows the user to apply custom R functions across all extreme event data frames created by the `gen_hw_sets` call. The user can apply any R function that follows certain standards in accepting input and returning output. Full details on these standards are given in the main package vignette.

As an example, if the user wanted to calculate the average temperature of the heat waves identified for each ensemble member in the output generated by the code above, he or she could write a simple function:

```
average_mean_temp <- function(hw_datafr){
  out <- mean(hw_datafr$mean.var)
  return(out)
}
```

This function can then be applied across all extreme event data sets output by `gen_hw_set` using the `apply_all_models` function. For example, to apply this function to all the example output results that come with the package, the user could run:

```
out <- system.file("extdata/example_results", package = "futureheatwaves")
apply_all_models(out = out, FUN = average_mean_temp)

#>  model ensemble  value
#> 1  bcc1          1 302.3745
#> 2  ccsm          1 302.4458
#> 3  ccsm          2 302.3428
```

This output gives the results (value column) of running the custom function for each ensemble member of each climate model. Note that the location of the directory with the heat wave data frames must be specified using the `out` argument when calling `apply_all_models`. Typically, this will be the directory path for the directory specified with the `out` argument in `gen_hw_set`.

Location-specific results can be generated using the `city_specific` argument in `apply_all_models`:

```
apply_all_models(out = out, FUN = average_mean_temp, city_specific = TRUE)

#>  model ensemble city  value
#> 1  bcc1          1 balt 305.1816
#> 2  bcc1          1 nwk  300.3367
#> 3  bcc1          1 ny   300.3367
#> 4  bcc1          1 phil 305.1816
#> 5  bcc1          1 prov 298.0402
#> 6  ccsm          1 balt 303.1277
#> 7  ccsm          1 nwk  302.4053
#> 8  ccsm          1 ny   302.4053
#> 9  ccsm          1 phil 302.3425
#> 10 ccsm          1 prov 301.8895
#> 11 ccsm          2 balt 302.9373
#> 12 ccsm          2 nwk  302.2748
#> 13 ccsm          2 ny   302.2748
#> 14 ccsm          2 phil 302.2858
#> 15 ccsm          2 prov 301.9520
```

The same process can be used to create a number of other summaries of the identified extreme events. For example, it could be used to determine average length of extreme events or estimate how much earlier in the year events are expected to start across an ensemble of climate model simulations. The functionality can also be used for more complex analysis of extreme event files. For example, it can be used to apply epidemiological models of heat wave to estimate excess heat-related mortality under different future scenarios; an example of this application is provided in the main `futureheatwaves` vignette. The output from `apply_all_models` is structured as "tidy" data (Wickham, 2014), allowing it to be used easily with the graphing package `ggplot2` (Wickham, 2009) and other packages in the tidyverse.

Customizing the extreme event definition

By default, the package identifies extreme events in climate model output data using a specific definition for heat waves that has been used in some epidemiological and climate impact research (e.g., [Anderson and Bell \(2009\)](#)):

A *heat wave* is two or more days at or above a city-specific threshold temperature, with the threshold determined as the 98th percentile of year-round temperature in the city during some reference period (by default, 1990–1999).

However, this is not the only accepted heat wave definition in the scientific literature. A variety of different heat wave definitions have been used to identify heat waves in a time series of temperature data ([Smith et al., 2013](#); [Kent et al., 2014](#); [Chen et al., 2015](#); [Anderson and Bell, 2009](#)), and the choice of heat wave definitions can influence both projected heat wave trends ([Smith et al., 2013](#)) and estimates of health risks during events ([Chen et al., 2015](#); [Kent et al., 2014](#); [Anderson and Bell, 2009](#)). Further, other types of extreme events will be defined differently than heat waves (for example, frost day spells may be defined as one or more days with temperature at or below 32°F / 0°C).

Therefore, this package allows the user to extensively customize the definition used to identify extreme events. Users can write a custom R function with either a different heat wave definition (see [Smith et al. \(2013\)](#) and [Kent et al. \(2014\)](#) for listings of some of the definitions used in scientific studies) or with a definition appropriate for a different type of extreme event (e.g., one or more days at or below 32°F / 0°C for frost day spells). For heat wave identification, researchers might want to use a different event definition because, for example, it matches the definition used by local health officials to declare heat wave warnings or, in the case of health impact assessments, to match with a definition used in an epidemiological study. For studies of other extreme events, a heat wave definition likely will not be applicable and so a customized definition is necessary.

Three components of the extreme event definition can be easily customized in the `gen_hw_set` function call, without creating a new R function to use to identify heat waves. First, many extreme event definitions are based on conditions that are rare in the study location ([Cubasch et al., 2013](#)), but definitions may vary in how rare conditions must be. For example, some of the different definitions used to identify heat waves vary only in the percentile temperature used for a threshold (e.g., one definition is ≥ 2 days at or above the 98th percentile temperature at a location while another is ≥ 2 days at or above the 99th percentile temperature; [Kent et al. \(2014\)](#); [Smith et al. \(2013\)](#)). Therefore, the **futureheatwaves** package allows users to change the percentile of the variable of interest required for an extreme event using the `probThreshold` option in `gen_hw_set`. Other heat wave definitions vary only in the number of consecutive days that must be over the threshold for a period to qualify as an extreme event (e.g., one definition is ≥ 2 days at or above the 98th percentile temperature at a location while another is ≥ 4 days at or above the 98th percentile temperature; [Anderson and Bell \(2009\)](#)). Therefore, the package allows the user to change the number of days used in the heat wave definition using the `numDays` argument in the `gen_hw_set` function. Combined, these two customization choices allow the user to identify heat waves using many of the heat wave definitions used in previous climate and health research— for example, 9 of 16 heat wave definitions outlined in [Kent et al. \(2014\)](#) could be fit using different combinations of these two options for specifying threshold percentile and number of days. Third, some extreme events like cold waves and frost day spells are defined as a certain number of days below, rather than above, a threshold. While the default is to identify events by searching for days above a threshold, this behavior can be changed with the `above_threshold = FALSE` argument in the `gen_hw_set` function.

Beyond these simpler options, the customization of the event definition is even more extensive as one has the option of writing and using a custom R function to identify extreme events. This functionality allows the user to use definitions that either require a number of days above or below an absolute threshold (e.g., maximum temperature of $\geq 95^\circ\text{F}$ for ≥ 1 day [Kent et al. \(2014\)](#); [Tan et al. \(2007\)](#); minimum temperature $\leq 0^\circ\text{C}$ for ≥ 1 day for frost day spells) or that require a combination of thresholds to be met (e.g., maximum daily temperature above a lower threshold every day of the heat wave and above a higher threshold for a certain number of days within the heat wave; [Kent et al. \(2014\)](#); [Peng et al. \(2011\)](#)). To use a customized event definition, the user must write and load an R function that implements the definition. This custom function is passed to the `gen_hw_set` function using the `IDheatwavesFunction` argument. To work correctly, this custom function must allow only specific inputs and generate only specific outputs; details about the required structure are provided in the main **futureheatwaves** package vignette. To increase processing speed when identifying extreme events, we coded parts of the default event definition function in C++ and synced it with R using the **Rcpp** package ([Eddelbuettel and Francois, 2011](#)). Users should consider a similar strategy for custom heat wave definitions, especially when processing a large number of climate projection files.

Exploring sensitivity of results to adaptation

Extreme events tend to be identified based on conditions that are rare for a specific location using location-specific relative thresholds. These thresholds are often defined for climate impact studies based on a variable's distribution at that location in present-day or historical data. However, for some extreme events, impacts are associated with how rare the conditions during the event are compared to current norms in the location (Anderson and Bell, 2009), which suggests some capacity for adaptation to heat and raises the question of whether extreme events should be defined using a percentile threshold based on present-day variable distributions or based on distributions in the time period being projected. Therefore, it can be interesting to explore trends in extreme events under climate change if extreme events are identified based on variable distributions during the projection period or another future period. The `futureheatwaves` package allows users to specify the time period to use when determining a location-specific relative threshold for an event definition using the `thresholdBoundaries` argument in the function `gen_hw_set`. This feature allows users to explore how sensitive projections of impacts are to this choice of the time period to use when determining relative variable measures, including thresholds used for percentile-based event definitions.

Similarly, some of the event characteristics (e.g., `mean.temp.quantile`, Table 2) are also calculated by the package based on relative temperature, providing measures of how the value of the variable of interest during an extreme event compares to the typical distribution of that variable at that location (e.g., the `mean.var.quantile`, `min.var.quantile`, and `max.var.quantile` characteristics, Table 2) or how long conditions of a certain rarity persisted during the event (e.g., the `days.above.99th` and `days.above.99.5th` characteristics, Table 2). These characteristics are measured for each of the extreme events identified by the `gen_hw_set` function by taking the absolute value of the variable during the event (e.g., average temperature during the heat wave is 90°F, 32.2°C) and comparing it to the location's typical variable distribution. This process generates relative measures of how intense the event is compared to what is normal in that location (e.g., 90°F, 32.2°C is in the 99th percentile of year-round temperatures in the location).

These relative event characteristics will vary depending on whether you calculate them based on a location's present-day variable distribution or on the location's variable distribution in the future, since the distributions of many relevant variables (e.g., temperature, precipitation) are expected to change in many locations with climate change. The package therefore allows the user to specify date ranges of the temperature distributions to be used in calculating these relative temperature metrics in each location, which can be done using the `referenceBoundaries` option of `gen_hw_set`.

Mapping grid points

Finally, it can be useful to explore the location of the climate model grid point used to pull climate model output for each study location with a given climate model. Therefore, the package has a function called `map_grid_leaflet` that plots the locations of grid points used for each location from each climate model. This function is built using the `htmlWidget` `leaflet` package (Cheng and Xie, 2016). The following code illustrates the use of this function with the example data to create Figure 5, which plots the grid points used in the example data from the BCC climate model in the example data:

```
out <- system.file("extdata/example_results", package = "futureheatwaves")
map_grid_leaflet(plot_model = "bcc1", out = out)
```

This interactive map can be panned and zoomed to explore the locations of climate model grid points used to represent each study location. This mapping function works for study locations worldwide.

Extensions

While this package was created to be used for research on extreme events in climate change projections, it can be used more broadly. For example, there are other episodes like wildfires and air pollution where it may be interesting to identify extended periods of high exposures in projection time series. The `futureheatwaves` package is not exclusive to CMIP5 model output data, and so could be applied to gridded air pollution model output to explore these exposures.

Future directions for working with climate model output in R

Research that assesses the potential impacts of climate change is critical in informing current policy choices, and R is an important tool for many researchers performing such assessments. While the `futureheatwaves` package described here takes steps to facilitate the assessment of impacts related to

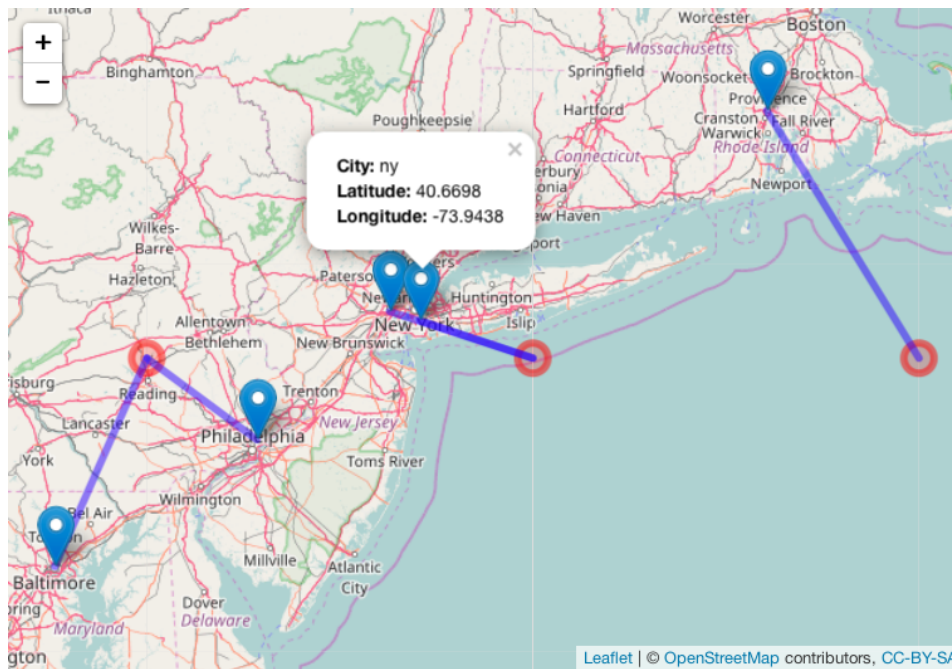


Figure 5: Snapshot of an interactive map created using the `map_grid_leaflet` showing the locations of study cities and their matching climate model grid points for the BCC climate model example data included with `futureheatwaves`. The lines on the map connect each climate model grid point to the study location(s) for which that grid point was used. The interactive maps include pop-ups with city identifiers; one is shown open in this snapshot as an example. From this map, you can see that the climate model grid point closest to New York City for this climate model is over the Atlantic Ocean.

sustained, multi-day events, a number of challenges remain in working with climate model output data in R, and future R software development offers the potential to further address the challenges of working with this data.

One important step in future development of R software to work with climate model output could be the development of R wrappers for some of the existing command line tools available through the Climate Data Operators (CDO) software (Schulzweida, 2017). Libraries already exist for Python and Ruby that allow the functionality of CDO tools to be used within these scripting languages (available from the Max Planck Institute at <https://code.zmaw.de/projects/cdo/wiki/Cdo%7Brbpy%7D>). While one R package (`ncdf4.helpers`; Bronaugh (2014)) already provides R wrappers for a few CDO operators, such functionality could be extended through future R software to capture more of the full functionality of the CDO toolkit.

Another important path for development could be through approaches that allow researchers to take advantage of the statistical tools offered by R while maintaining large climate model output files in a netCDF format. For example, Goncalves and coauthors recently described a “round table” approach of connecting as-needed data access from netCDF climate data files through to functionality available in R and CDO through the intermediary of a MonetDB database system (Goncalves et al., 2015). In other topical areas, R programmers are also improving the efficiency of working with data in large netCDF files through approaches that avoid loading all data in-memory. For example, the Bioconductor package `ncdfFlow` enables R users to conduct analyses of hundreds of large flow cytometry output files through the creation and use of an `ncdfFlowSet` class that stores the data in an HDF5 format rather than in-memory (Jiang et al., 2017; Finak et al., 2014). Such an approach could be promising for future R software development for working with large climate model files.

Acknowledgements

This work was supported by grants from the National Institute of Environmental Health Sciences (R00ES022631), the National Science Foundation (1331399), and the Colorado State University Vice President for Research. Thank you to early testers of the package: Julia Bromberek, Wande Benka-Coker, Josh Ferreri, Ryan Gan, Molly Gutilla, Mike Lyons, Casey Quinn, Rachel Severson, and Meilin Yan.

Bibliography

- G. B. Anderson and M. L. Bell. Weather-related mortality: How heat, cold, and heat waves affect mortality in the United States. *Epidemiology*, 20(2):205–213, 2009. URL <https://doi.org/10.1097/ede.0b013e318190ee08>. [p133, 134]
- D. Bronaugh. *Ncdf4.helpers: Helper Functions for Use with the Ncdf4 Package*, 2014. URL <https://CRAN.R-project.org/package=ncdf4.helpers>. R package version 0.3.3. [p128, 135]
- D. Bronaugh and U. Drepper. *PCICt: Implementation of POSIXct Work-Alike for 365 and 360 Day Calendars.*, 2013. URL <https://CRAN.R-project.org/package=PCICt>. R package version 0.5.4. [p128]
- K. Chen, J. Bi, J. Chen, X. Chen, L. Huang, and L. Zhou. Influence of heat wave definitions to the added effect of heat waves on daily mortality in Nanjing, China. *Science of the Total Environment*, 506-507:18–25, 2015. URL <https://doi.org/10.1016/j.scitotenv.2014.10.092>. [p133]
- 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. [p134]
- U. Cubasch, D. Wuebbles, D. Chen, M. C. Facchini, D. Frame, N. Mahowald, and J.-G. Winther. Introduction. In: *Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change. Climate Change 2013*, 5:119–158, 2013. [p124, 133]
- B. Eaton, J. Gregory, B. Drach, K. Taylor, S. Hankin, J. Caron, R. Signell, P. Bentley, G. Rappa, H. Höck, and others. *NetCDF Climate and Forecast (CF) Metadata Conventions Version 1.6*, 2011. URL <http://cfconventions.org/Data/cf-conventions/cf-conventions-1.6/build/cf-conventions.html>. [p127]
- D. Eddebuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL <https://doi.org/10.18637/jss.v040.i08>. [p133]
- G. Finak, J. Frelinger, W. Jiang, E. W. Newell, J. Ramey, M. M. Davis, S. A. Kalams, S. C. De Rosa, and R. Gottardo. OpenCyto: An open source infrastructure for scalable, robust, reproducible, and automated, end-to-end flow cytometry data analysis. *PLoS Comput Biol*, 10(8):e1003806, 2014. [p135]
- G. Flato, J. Marotzke, B. Abiodun, P. Braconnot, S. C. Chou, W. J. Collins, P. Cox, F. Driouech, S. Emori, V. Eyring, and others. Evaluation of Climate Models. In: *Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change. Climate Change 2013*, 5:741–866, 2013. [p125, 126]
- P. R. Gent, G. Danabasoglu, L. J. Donner, and others. The Community Climate System Model Version 4. *Journal of Climate*, 24(19):4973–4991, 2011. URL <https://doi.org/10.1175/2011jcli4083.1>. [p130]
- R. Goncalves, M. Ivanova, F. Alvanaki, J. Maassen, K. Kyzirakos, O. Martinez-Rubi, and H. Mühleisen. A round table for multi-disciplinary research on geospatial and climate data. In *e-Science (e-Science), 2015 IEEE 11th International Conference on*, pages 165–170. IEEE, 2015. [p135]
- E. Hawkins and R. Sutton. The potential to narrow uncertainty in regional climate predictions. *Bulletin of the American Meteorological Society*, 90(8):1095–1107, 2009. URL <https://doi.org/10.1175/2009bams2607.1>. [p124]
- M. Jiang, G. Finak, and N. Gopalakrishnan. *ncdfFlow: A Package That Provides HDF5 Based Storage for Flow Cytometry Data*, 2017. R package version 2.22.0. [p135]
- S. T. Kent, L. A. McClure, B. F. Zaitchik, T. T. Smith, and J. M. Gohlke. Heat waves and health outcomes in Alabama (USA): The importance of heat wave definition. *Environmental Health Perspectives*, 122(2):151–158, 2014. URL <https://doi.org/10.1289/ehp.1307262>. [p133]
- G. A. Meehl, C. Covey, T. Delworth, M. Latif, B. McAvaney, J. F. B. Mitchell, R. J. Stouffer, and K. E. Taylor. The WCRP CMIP3 Multimodel Dataset: A new era in climate change research. *Bulletin of the American Meteorological Society*, pages 1383–1394, 2007. [p126]
- T. Mendlik, G. Heinrich, and A. Leuprecht. *Wux: Wegener Center Climate Uncertainty Explorer*, 2016. URL <https://CRAN.R-project.org/package=wux>. R package version 2.2.1. [p128]
- P. Michna and M. Woods. RNetCDF—a package for reading and writing netcdf datasets. *The R Journal*, 5:29–36, 2013. [p127]

- P. Michna and M. Woods. *RNetCDF: Interface to NetCDF Datasets*, 2016. URL <https://CRAN.R-project.org/package=RNetCDF>. R package version 1.8.2. [p127]
- R. D. Peng, J. F. Bobb, C. Tebaldi, L. McDaniel, M. L. Bell, and F. Dominici. Toward a quantitative estimate of future heat wave mortality under global climate change. *Environmental Health Perspectives*, 119(5):701–706, 2011. URL <https://doi.org/10.1289/ehp.1002430>. [p133]
- D. Pierce. *Ncdf4: Interface to Unidata netCDF (Version 4 or Earlier) Format Data Files*, 2015. URL <https://CRAN.R-project.org/package=ncdf4>. R package version 1.15.0. [p127]
- U. Schulzweida. *CDO User's Guide*, 2017. URL <https://code.zmaw.de/projects/cdo/embedded/cdo.pdf>. Climate Data Operators, Version 1.8.1. [p127, 135]
- T. T. Smith, B. F. Zaitchik, and J. M. Gohlke. Heat waves in the United States: Definitions, patterns and trends. *Climatic Change*, 118(3-4):811–825, 2013. URL <https://doi.org/10.1007/s10584-012-0659-2>. [p133]
- J. Tan, Y. Zheng, G. Song, L. S. Kalkstein, A. J. Kalkstein, and X. Tang. Heat wave impacts on mortality in Shanghai, 1998 and 2003. *International Journal of Biometeorology*, 51(3):193–200, 2007. URL <https://doi.org/10.1007/s00484-006-0058-3>. [p133]
- K. E. Taylor and C. Doutriaux. *CMIP5 Model Output Requirements: File Contents and Format, Data Structure and Metadata*, 2010. URL http://cmip-pcmdi.llnl.gov/cmip5/docs/CMIP5_output_metadata_requirements.pdf. [p125, 126, 127]
- K. E. Taylor, R. J. Stouffer, and G. A. Meehl. An overview of CMIP5 and the experiment design. *Bulletin of the American Meteorological Society*, 93(4):485–498, 2012. URL <https://doi.org/10.1175/bams-d-11-00094.1>. [p124, 125, 126]
- K. Todd-Brown and B. Bond-Lamberty. *RCMIP5: Tools for Manipulating and Summarizing CMIP5 Data*, 2016. URL <https://cran.r-project.org/web/packages/RCMIP5/>. R package version 1.2.0. [p128]
- UCAR/NCAR Computational and Information Systems Laboratory. *The NCAR Command Language (NCL)*, 2017. URL <https://doi.org/10.5065/d6wd3xh5>. Version 6.4.0. [p127]
- H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. [p132]
- H. Wickham. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. doi: 10.18637/jss.v059.i10. URL <https://www.jstatsoft.org/v059/i10>. [p132]
- X.-G. Xin, T.-W. Wu, and Z. Jie. Introduction of CMIP5 experiments carried out with the climate system models of Beijing Climate Center. *Advances in Climate Change Research*, 4(1):41–49, 2013. URL <https://doi.org/10.3724/sp.j.1248.2013.041>. [p130]

G. Brooke Anderson
Colorado State University
Department of Environmental & Radiological Health Sciences
1681 Campus Delivery
Fort Collins, Colorado 80523
brooke.anderson@colostate.edu

Colin Eason
Colorado State University
Department of Computer Science
1873 Campus Delivery
Fort Collins, Colorado 80523
aimisce@gmail.com

Elizabeth A. Barnes
Colorado State University
Department of Atmospheric Science
1371 Campus Delivery
Fort Collins, CO 80523
eabarnes@atmos.colostate.edu

alineR: an R Package for Optimizing Feature-Weighted Alignments and Linguistic Distances

by Sean S. Downey, Guowei Sun and Peter Norquest

Abstract Linguistic distance measurements are commonly used in anthropology and biology when quantitative and statistical comparisons between words are needed. This is common, for example, when analyzing linguistic and genetic data. Such comparisons can provide insight into historical population patterns and evolutionary processes. However, the most commonly used linguistic distances are derived from edit distances, which do not weight phonetic features that may, for example, represent smaller-scale patterns in linguistic evolution. Thus, computational methods for calculating feature-weighted linguistic distances are needed for linguistic, biological, and evolutionary applications; additionally, the linguistic distances presented here are generic and may have broader applications in fields such as text mining and search, as well as applications in psycholinguistics and morphology. To facilitate this research, we are making available an open-source R software package that performs feature-weighted linguistic distance calculations. The package also includes a supervised learning methodology that uses a genetic algorithm and manually determined alignments to estimate 13 linguistic parameters including feature weights and a skip penalty. Here we present the package and use it to demonstrate the supervised learning methodology by estimating the optimal linguistic parameters for both simulated data and for a sample of Austronesian languages. Our results show that the methodology can estimate these parameters for both simulated and real language data, that optimizing feature weights improves alignment accuracy by approximately 29%, and that optimization significantly affects the resulting distance measurements. Availability: **alineR** is available on CRAN.

Introduction

Human speech patterns change through time in response to both cultural and demographic processes of speech communities such as migration and social contact. Analyzing differences among languages can provide insight into historical patterns and general processes of biological and cultural evolution (Pagel, 2012). Linguistic distances based on the comparison of two words are often used when quantitative analyses are required. For example, numerous studies make language/gene comparisons on continental and regional scales (Sokal, 1988; Barbujani and Sokal, 1990; Cavalli-Sforza et al., 1992; Smouse and Long, 1992; Chen et al., 1995; Cavalli-Sforza, 1991; Cox, 2003; Hunley and Long, 2005; Diamond and Bellwood, 2003; Nettle and Harriss, 2003), and also at smaller geographical scales (Lansing et al., 2007; Downey et al., 2008; Tumonggor et al., 2014; Cox and Lahr, 2006). In addition, edit distances are used in text mining, for example in the extraction of news content (Qiujun, 2010), and in biological applications such as extracting mutation data from the literature (Horn et al., 2004).

The use of evolutionary linguistics in anthropology suggests that further development of quantitative methods are necessary in order to identify new patterns in language families, to identify controversial or undiscovered language families, and to address outstanding problems in human prehistory (Croft, 2008). Research in computational phonology has developed several quantitative metrics for measuring linguistic distances between pairs of words. Algorithms for quantifying the distance between cognate pairs (words with a shared meaning) include measuring phonetic sequence distance based on types of segments (Covington, 1998), or the feature scores of phonemes (Somers, 1998). However, the most common approach is the Levenshtein distance – also called the ‘edit distance’ – which is defined as the minimum total number of additions, deletions, and substitutions of symbols necessary to transform one word to the other (Levenshtein, 1966). Various mathematical refinements to the Levenshtein distance have been proposed (Wichmann et al., 2010; Petroni and Serva, 2010), including an approach that uses empirically determined log-odds (Fine and Jaeger, 2013). The Levenshtein distance is parsimonious and robust and it has been found to correlate with perceptions of dialectal distances (Gooskens and Heeringa, 2004); however, feature-based alignment approaches have been found to be a complementary approach to calculating linguistic distances (Kondrak, 2000).

The ALINE algorithm

ALINE is an automatic phonetic sequence alignment algorithm that determines the similarity of two words (Kondrak, 2000). It uses dynamic programming to calculate the optimal similarity score

of candidate alignments. In addition to binary character comparisons, insertions, and deletions, the algorithm uses phonetic information to determine the resulting optimal score. A set of feature weighting parameters and a skip penalty are used to determine individual similarity scores for each phonetic feature in the words being measured; thus, the optimal phonetic sequence alignment depends on the values of the feature weight parameters, and the resulting similarity scores are sensitive to the selection of these values.

Similarity scores can range from $[0, \infty]$ and are strongly influenced by word length. To facilitate integration with biological and evolutionary research we previously defined the ALINE Distance as,

$$\text{ALINE}_{Dist} = 1 - \frac{2s}{s_1 + s_2} \quad (1)$$

where s is the similarity score and $s_{1,2}$ are similarity scores for each word's self-comparison (Downey et al., 2008). This equation results in a finite value $[0, 1]$ that can be easily compared, for example, to common population differentiation statistics such as the fixation index (Fst). For this reason, our package by default returns ALINE_{Dist} , but can optionally return the similarity score. Because similarity scores and ALINE distances are expected to be sensitive to feature weights, the package parameterizes the values used by the ALINE algorithm so they can be easily modified within the R environment.

We provide **alineR** as an open-source package for the R statistical computing language that facilitates calculation of linguistic distances using the ALINE algorithm. The original ALINE algorithm is provided as an executable (<http://webdocs.cs.ualberta.ca/~kondrak/>) so the default parameters cannot be modified. An open-source python version called PyAline (Huff, 2010) (<http://pyaline.sourceforge.net>) allows these values to be modified; however, parameter estimation was not a focus. And while the R base command `adist()` and several packages can calculate Levenshtein distances (see `stringdist()` in **stringdist** (van der Loo, 2014), `levenshteinDist()` in **RecordLinkage** (Borg and Sariyar, 2016), and `stringDist()` in **Biostrings** (Pagès et al., 2017)), to the best of our knowledge, this is the first time ALINE distances can be calculated directly from an R function.

An important new feature of **alineR** is to provide a way to estimate the feature-weight parameters and skip penalty. Below we analyze how changing these values affects the resulting alignments and distance measurements. We present a supervised learning methodology that uses manual alignment determinations and a genetic algorithm (GA) to estimate the optimal feature weights for any paired word data. First, we use a simulation analysis and determine that the GA can correctly estimate known feature weights for simulated data. Second, we show that a supervised learning methodology can successfully estimate optimal (unknown) linguistic parameters for a data set consisting of Austronesian word lists from Sumba in Eastern Indonesia. Third, we show that optimizing feature weights improves alignment accuracy using manual determinations as a baseline. Finally, we show how estimating feature-weights and skip penalties affects the resulting distance calculations.

Parameterizing the ALINE algorithm

The ALINE algorithm is a phonetic sequence alignment algorithm based on multivalued features. The program runs quickly because it uses dynamic programming and it is written in C++. Twelve (12) features are considered in calculating the phonetic similarity score: syllabic, voice, lateral, high, manner, long, place, nasal, aspirated, back, retroflex, and round. In addition, there is a skip penalty. Weighting values for each of these parameters are used to choose the optimal string alignments as well as the resulting similarity score. However, in the publicly available version of ALINE the default values were compiled into the original program so they could not be modified. Our **alineR** package includes a modified version of the original ALINE code that interfaces directly with R.

Overview of article

In the next section we provide a how-to guide for calculating ALINE distances and similarity scores with **alineR**. We present simple instructions for basic alignment operations and for users who want to calculate linguistic distances using this alternative to the Levenshtein distance, the instructions in this section may be sufficient. We then describe the genetic algorithm and illustrate with simple examples how to use it with supervised-learning to optimize ALINE's feature-weight parameters. Next, we show the results from a simulation experiment that validates that the GA can recover a set of 'unknown' feature weighting parameters. We then present a proof-of-concept case study in which we use the GA to determine the optimal feature-weighting values for a sample of languages from Eastern Indonesia. This includes a description of the training dataset and the work flow necessary to reproduce the analysis. We perform a statistical analysis of the effect the supervised learning and GA optimization process has on the resulting linguistic distance measurements. Finally, we perform a bootstrap analysis to determine whether the results are stable. We close by briefly discussing some

possible applications of **alineR** in psycholinguistics and morphology.

Basic alignment and distance calculations with **alineR**

Calculating ALINE distances minimally requires two scalar words using UTF-8 file encoding. Word lists should be encoded in the International Phonetic Alphabet to maximize the use of phonetic feature information (Association, 1999). If IPA is not available, the ASCII subset can also be used.

First, load the **alineR** package in the standard way.

```
library('alineR')
```

Consider the following example for calculating the phonetic distance between two related words meaning 'to float' from Borneo: [kərampu] and [ləməntuŋ]. The most basic use of **alineR** entails passing the two words as the first and second parameters to the `aline()` function. The standardized $ALINE_{Dist}$ is returned.

```
aline("kərampu", "ləməntuŋ")
[1] 0.4864198
```

More typically, word lists will be passed as two vectors that will be analyzed such that $ALINE_{Dist}$ will be calculated for matching elements of each vector. For example, here we pass two vectors representing two cognate pairs ("stone" and "to float") for two related languages. Note that the phonetic difference between *u* and *ʊ* does not yield a quantitative difference in the resulting ALINE score.

```
aline(c("batu", "kərampuʊ"), c("batu", "ləməntuŋ") )
[1] 0.0000000 0.4864198
```

The `aline()` function has several parameters that provide additional functionality.

```
aline( w1, w2, sim = FALSE, m1 = NULL, m2 = NULL, mark = FALSE, alignment = FALSE, ...)
```

The first and second elements, `w1` and `w2`, are required and they are used for passing the two word vectors to be compared as shown above. All additional parameters are optional and provide additional functionality, which will be illustrated in more detail below. These include the following: `sim = TRUE` returns the similarity score rather than the ALINE Distance; `m1` and `m2` allow user-defined feature mappings; setting `mark = TRUE` will mark invalid characters with an @ character to assist in data checking; setting `alignment = TRUE` will return the IPA word pairs vertically arranged so that the aligned characters can be delimited with vertical bars (|). Additionally, feature weighting parameters can also be passed to the internal `raw.alignment()` function using `...`. In the next sections we explain some more advanced uses of **alineR** which require using these optional parameters.

A typical use in historical linguistics is to calculate a matrix of language-distance comparisons among multiple languages. Given the numerous ways that language data can be stored, the need for data consistency, and the difficulty of providing comprehensive error-handling, we do not provide a built-in function for multiple-language comparisons in **alineR**. However, data processing in R is relatively straight-forward. Here we illustrate one possible approach that can be used as a starting point for more complicated analyses. This example processes three word lists that each include three glosses. The results are combined into a distance matrix composed of average ALINE Distance scores.

```
# multiple language comparisons
word.lists <- rbind(c("baqa", NA, "anax"), c("haqa", "dodo", "anar"),
                  c("abut", "suli", "oan"))
glosses <- colnames(word.lists) <- c("akar", "alir_me", "anak")
languages <- rownames(word.lists) <- c("language.1", "language.2", "language.3")
word.lists

           akar    alir_me  anak
language.1 "baqa"  NA      "anax"
language.2 "haqa"  "dodo"  "anar"
language.3 "abut"  "suli"  "oan"

# dim empty matrices: a (ALINE scores), and n (a counter)
n <- matrix(0, nrow = length(languages), ncol = length(languages),
            dimnames = list(languages, languages))
a <- n
```

```

# nested loops for calculating the mean ALINE Distances for multiple languages and glosses
for(i in 1:length(glosses)){ # loop glosses
  for(j in 1:length(languages)){ # outer language loop
    for(k in 1:length(languages)){ # inner language loop
      if(j >= k){
        x <- word.lists[j, i] # first word to compare
        y <- word.lists[k, i] # second word to compare
        if( !is.na(x) & !is.na(y) ) { # skip if missing data
          a[j, k] <- a[j, k] + aline(x, y) # ALINE Distance
          n[j, k] <- n[j, k] + 1 # increment counter
        }
      }
    }
  }
}

```

as.dist(a / n) # distance matrix composed of mean ALINE Distances

```

          language.1 language.2
language.2 0.3500000
language.3 0.3869697 0.5479798

```

alineR uses a custom ASCII encoding scheme to identify features. Valid encodings include lowercase letters from *a* to *z*, and the uppercase modifiers shown in Table 1 are used to indicate features.

Feature	Code
dental	D
palato-alveolar	V
retroflex	X
palatal	P
spirant	S
nasal	N
aspirated	A
long	H
front	F
central	C

Table 1: ALINE features

The full list of *IPA::feature* mappings, including ASCII values, is stored in a data frame included in the package. It can be seen using the `show.map()` function. For example, row 2 indicates that Latin Capital B with the Unicode value of 66 will be encoded as a spirant “b”, “bS”, with the two ASCII values, 98 and 83.

```

show.map()
      IPA Aline U.Val  A.Val
1
2      B  bS   66    98 83
3      O  oF   79   111 70
4      a   a   97     97
5      b   b   98     98
...
102     N 8319     78

```

The `encode.ALINE()` function can be used to see the ASCII character encoding of an IPA character string or vector.

```

encode.ALINE("diŋŋira", "dinnira")

      diŋŋira  dinnira
"digNgNira"  "dinnira"

```

It is not required for basic usage to manually encode words because this function is called internally. However, if an unknown IPA character is used in an alignment, under the default behavior a warning is issued, the character is dropped, and the remaining characters are aligned as usual. When this occurs, the `encode.ALINE()` command may be used with `mark = TRUE` to substitute all unknown characters with the `@` symbol. In the following example, `ɣ` is an unknown feature type.

```
encode.ALINE(c("ũmlatθ", "ɣmlatθ"), mark = TRUE)
Invalid character: ɣ in ɣmlatθ
```

```
      ũmlatθ      ɣmlatθ
"uNmlattS" "@mlattS"
```

In such cases, it is possible to eliminate the system warnings by providing these characters with feature mappings. In addition, it is possible to over-ride existing mappings given by `show.map()`. Both can be done using the `m1` and `m2` parameters of `aline()`. For example, the following substitutes all instances of "ɣ" with "o".

```
aline(w1 = c("ũmlatθ", "dinnira"), w2 = c("ɣmlatθ", "dinnira"), m1 = "ɣ", m2 = "o")
[1] 0.07647059 0.10810811
```

By default, `aline()` returns a vector of ALINE distances indexed by the position in the input vectors. However, additional information about the alignments can also be returned, including the optimal alignment and the similarity score, as shown here.

```
aline("watu", "dat", alignment = TRUE, sim = TRUE)
      pair1
w1      watu
w2      dat
scores      50
a1  | - w a t | u
a2  | d - a t | -
```

In this example, setting the optional parameter `alignment = TRUE` will change the output format to a data frame in which each column represents a word-pair comparison. In this example, only one pair of words is compared. Rows 1 and 2 in this data frame contain word 1 ("w1"), and word 2 ("w2"). The third element will contain either the ALINE distance (if `sim = FALSE`) or the similarity score (if `sim = TRUE`). Rows 4-5 contain the optimal alignment of word 1 and word 2. If three pairs of words are compared, the data frame would consist of three columns and five rows. We adopted this output format as a convenience so that the alignments in rows 4 and 5 could be easily examined directly in the R command output window.

In determining the optimal alignment, the ALINE algorithm associates feature values, or weights, with particular phonemes based on phonologically similar features. Note that each element of the resulting optimally aligned vector maps corresponding elements within the vertical bars. So when aligning *watu* and *dat*, the phonetic similarity of *at* yielded the highest similarity score, 50. In making these calculations, the algorithm calculates weighted feature values for each pair of features, including skips, to determine the optimal similarity score. A vector of the individual similarity scores for the phoneme pairs can be extracted using the `ALINE.segments()` function:

```
aline("watu", "dat", sim = TRUE) # returns similarity score for comparison
[1] 50
align <- raw.alignment(c("watu", "dat"))
cat(align[[3]], align[[4]], sep = "\n")
| - w a t | u
| d - a t | -
s <- ALINE.segments(align)
s
[1] 0 0 15 35
sum(s)
[1] 50
```

One of the key features of **alineR** is the ability to easily change the default feature weight values used to determine the optimal alignment and resulting ALINE distances. Here for example, reducing the weight given to Place from the default of 40 to 10 reduces the values of non-zero distances.

```
aline(c("batu", "kərampuʊ"), c("batu", "ləməntuŋ"))
[1] 0.0000000 0.4864198
```

```
aline(c("batu", "kərampuʊ"), c("batu", "ləməntuŋ"), Place = 10)
[1] 0.0000000 0.4567901
```

In examples introduced below, it will be more convenient to store the feature weights in a vector. In these instances, the vector can be passed as an argument to `aline()` by constructing a named list and using `do.call()`:

```
opts <- c(61, 92, 51, 26, 54, 38, 20, 40, 31, 38, 66, 72, 60) # feature weights
names(opts) <- names(formals(raw.alignment)[-1]) # add feature names
```

```
args <- c(list(w1 = c("batu", "kərampuʊ"), w2 = c("batu", "ləməntuŋ")), opts)
do.call("aline", args)
[1] 0.0000000 0.5604938
```

Optimal parameter estimation using a genetic algorithm for **alineR**

The 13 parameters used by ALINE creates a high-dimensional search space, and a grid search would therefore be computationally inefficient. Instead, we employ a genetic algorithm (Back et al., 1997) to determine the optimal feature weights for a given word list. The general approach is to mimic biological evolution and natural selection by using a performance function that iteratively evaluates alignments generated with candidate parameters. We define the performance function as the total number of “correct” alignments, where correctness is determined manually by a linguist trained in phonetic analysis.

An overview of the algorithm is provided in Figure 1. At startup, the total population size (N) and number of iterations are defined for each run of the genetic algorithm. (1) N vectors are initialized by sampling uniformly between $[0, 100]$ for each of the linguistic parameters. (2) The performance of each vector in the current (initial or iterated) population is evaluated using the performance function. (3) Each parameter vector in the population is ranked according to performance and the top $.25N$ are retained. (4) A new $0.25N$ vectors are initialized into the current generation. (5) Crossover 1 – the retained vectors (from step 3) are recombined using a crossover procedure to create $.25N$ in the current generation. (6) Crossover 2 – the retained vectors (from step 3) are recombined with the new random vectors (from step 4) using a crossover procedure to create $.25N$ new vectors. (7) The four subpopulations from steps 3-6 are combined. (8) Check if the maximum number of iterations has been reached. If not, the algorithm returns to step 2, but otherwise continues to step 9. (9) Aggregate all parameter vectors and estimate medians and variances from the top ranked $.25N$. Both crossover procedures involve selecting pairs of vectors, selecting a sequence of adjacent elements within each vector, and exchanging them between vectors. Crossover 1 samples $.25N$ individuals from n_s (with replacement) and pairs them for the crossover procedure. Crossover 2 samples $.125N$ individuals from n_s and pairs them with an equal number from n_r . Each GA run generates a vector of optimized values for each linguistic feature, and when convergence is achieved the median of each distribution is used to parameterize the ALINE algorithm for the training data. Convergence is determined using visual diagnostic plots. To account for the possibility that a single GA run incompletely explores the parameter space, we run multiple iterations and consolidate the results of all runs. In practice, we minimize the computation time by parallelizing this process, as we will demonstrate below. We note, however, that the GA we provide is only one approach to solving this high-dimensional search problem. There are any number of supervised and unsupervised optimization routines that could potentially be used to estimate the linguistic features available in **alineR**.

Supervised learning procedure using the genetic algorithm

We developed a supervised learning procedure for estimating the optimal feature weights using the GA. It is based on expert alignment determinations that are selected from a list of possible alignments for a given set of word lists. Commands in **alineR** are provided to implement this procedure. First, to create a training dataset we require a set of cognates and we use simulation to sample linguistic parameters from a uniform distribution to determine a list of possible alignments for each cognate pair. The simulation excludes the skip penalty (which creates numerous alternative alignments),

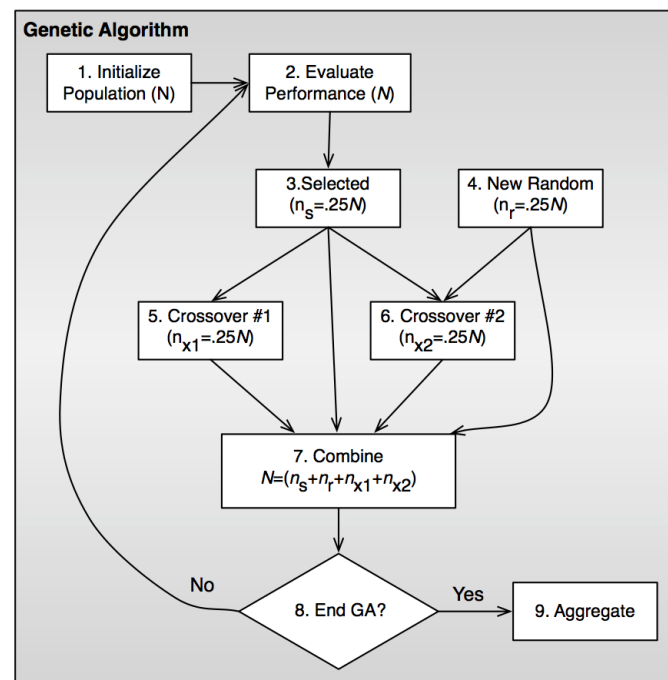


Figure 1: A flowchart showing how the linguistic parameters for ALINE are estimated using a genetic algorithm

and it eliminates cognate pairs with only a single alignment. We then quantify the number of possible alignments for each cognate pair. First, to prepare the cognate lists for the manual alignment determinations we will create some example data.

```

training_wordlists <- rbind(c("hamate", "kanabu", "delu"), c("pameti", "pena?o", "telu"))
training_wordlists
  [,1] [,2] [,3]
[1,] "hamate" "kanabu" "delu"
[2,] "pameti" "pena?o" "telu"

```

The next step is to perform the simulation process described above to generate a list of unique alignments. In the following series of commands, `generate_training()` will return an R object that will be used internally during the optimization process. However, specifying the optional `table = TRUE` parameter will export a file named 'candidate_alignments.csv' to the working directory that can be used to create a spreadsheet for manually identifying the best alignments.

```

training_set <- generate_training(raw.data = training_wordlists, search.size = 10,
                                table = TRUE)

```

Typically, each cognate pair can generate 3-6 unique alignments. But long words contain more phonemes and therefore comparisons between long words will result in a greater number of possible alignments, and in rare cases some cognate pairs can generate 17 or more. When this procedure is used, the list of possible alignments is then provided as a spreadsheet to a trained linguist for manual evaluation (Table 2). Minimally, this requires identifying the "best" of the provided alignments based on the rules of phonology and knowledge of the languages. The linguist's decisions are then provided to the GA as a vector indicating the numeric index of the "best" alignment. These are subsequently used in the optimization process. The following configuration initializes 200 populations and instructs the GA to run 50 iterations and return a list of feature weights. These parameters are designed to run quickly for demonstration purposes, and therefore the following optimization examples do not converge. More realistic parameters may run slowly.

```

linguist_determinations <- c(2, 1, 2)
optimal_set <- optimize.features(set = training_set, ranking = linguist_determinations,
                                num = 200, step = 50, replication = 3)
optimal_set
[1] 69 41 47 12 40 65 71 43 48 20 54 76 51

```


Alignment No.	Cognate pair 1	Cognate pair 2	Cognate pair 3
	hamate pameti	kanabu penaʔo	delu telu
1	- h a m - a t - e p - a m e - t i -	- - k a n a - b u - p e - - n a ʔ - - o	- d e l u t - e l u
2	- h a m - a t e p - a m e - t i	k - a n a - b u - p e - n a ʔ - - o	d e l u t e l u
3	h a m - a t - e p a m e - t i -	k - a n a - b u p e - n a ʔ - o	
4	h a m - a t e p a m e - t i	k - a n a b - u p e - n a ʔ o -	
Linguist's Choices	2	1	2

Table 2: Manual alignment determination worksheet

By default, `optimize.features()` returns a vector containing the optimized feature weights, but when `list = TRUE` the returned object can be used with the `features.plot()` function to generate a multi-panel plot showing the results of the optimization process (see Figure 3).

```
optimal_set <- optimize.features(set = training_set, ranking = linguist_determinations,
  num = 200, step = 50, replication = 3, list = TRUE)
features.plot(optimal_set) # not shown, but see Figure 3.
```

As noted above, the GA can be computationally intensive and may require replicates. Therefore it may be necessary to perform multiple runs. The following code illustrates this using a single processing core.

```
reps <- 4
MultiOptResult <- matrix(NA, nrow = reps, ncol = 13)
for (i in 1:reps){
  MultiOptResult[i,] <- optimize.features(set = training_set,
    ranking = linguist_determinations, num = 200, step = 50, replication = 3)
}
round(apply(MultiOptResult, 2, FUN = median)) # optimized feature weights
[1] 53 20 32 22 63 70 52 68 47 71 42 47 70
```

Here we show how to parallelize feature optimization using the `doMC` package (Revolution Analytics and Steve Weston, 2015).

```
# ...replicate using parallelization (OSX/linux)
library(doMC)
registerDoMC(cores = 4)
reps <- 4
MultiOptResult <- foreach(i = 1:reps, .combine = rbind) %dopar% {
  optimize.features(set = training_set, ranking = linguist_determinations, num = 200,
    step = 50, replication = 3)
}
opts <- round(apply(MultiOptResult, 2, FUN = median)) # optimized feature weights
names(opts) <- names(formals(raw.alignment)[-1])
opts
[1] 61 92 51 26 54 38 20 40 31 38 66 72 60
```

Finally, we illustrate how to pass the results from the optimization process in `opts` to `aline()`, and the effects on the resulting ALINE distances.

```
list1 <- c("batu", "kəɾampuʊ")
list2 <- c("batu", "ləməntuŋ")

aline(list1, list2)
```

```
[1] 0.0000000 0.4864198 # with default feature weights

args <- c(list(w1 = list1, w2 = list2), opts) # construct nested list for do.call()
do.call("aline", args)
[1] 0.0000000 0.5506173 # optimized Aline distances
```

Results

The **alineR** package redistributes a modified version of the original C++ ALINE code, and we maintained the original feature weight values originally used by Kondrak as defaults. Therefore, when the `aline()` function is used without providing optional feature weight parameters, the default performance should be equivalent with Kondrak's original algorithm. In this section we present the results of analyses we conducted to validate and test our GA optimization process. We do this by making an explicit comparison of the default performance of the ALINE algorithm provided in **alineR** (which is equivalent to the original C++ version of ALINE) to its performance using optimized parameters.

Genetic algorithm validation

First, we use a simulation experiment to validate that our algorithm converges under ideal conditions. We randomly simulate linguistic parameters and generate alignments for our data set that we designated as "correct" and then used the GA to estimate these values. The results are shown in Figure 2A. Under these conditions, the GA quickly converges and returns linguistic parameter values that correctly align all the cognate pairs in the training set. Figure 2B also shows results from a proof-of-concept trial of the supervised learning methodology, which we described in the next section.

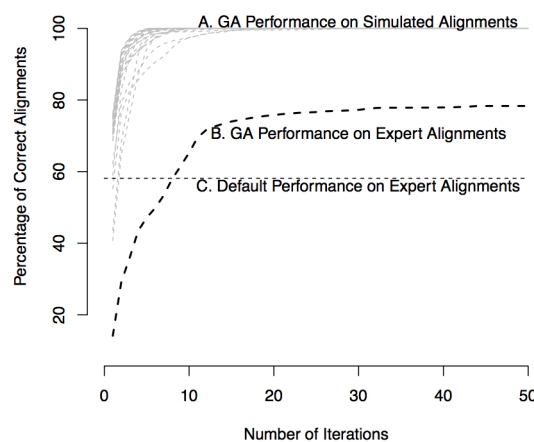


Figure 2: Genetic algorithm convergence plots showing (A) GA convergence using 100 simulated feature parameter sets. In all cases, the GA discovers parameters that could replicate the optimal alignments determined by the simulated parameter values; (B) GA performance estimating unknown parameter values using alignments determined manually by a trained linguist; (C) the alignment performance of the default parameter values when compared to manual determinations. In all cases the GA was run with a population of 100 and for 100 iterations to ensure convergence; only time steps 1-50 are plotted here.

Supervised learning with Indonesian languages

The overall goal of the optimization methodology is to determine whether our procedure can effectively estimate linguistic alignment parameters using real word lists with meaningful phonetic patterns. The reason this is important is because the frequency of linguistic features is expected to vary among languages or groups of languages so the optimal weighting parameters may also vary (Kondrak, 2003). As a proof-of-concept, we analyze word lists that were collected on the island of Sumba in

Eastern Indonesia (Lansing et al., 2007). The data set includes 11,515 unique words organized in a matrix consisting of 56 word lists and 439 cognates. Even though this sample includes a relatively small number of languages, there are approximately 12,000 unique pairwise cognate comparisons. To generate a training data set, we used a stratified framework to sample 203 cognate pairs from the database that are broadly representative of local phonemic complexity and submitted this data set to the linguist for manual alignment determinations. Strata were defined using the proportion of cognate pairs with each number of alignments (1-5) throughout the complete database. A historical linguist (Norquest) determined the “best” alignment for a sample of cognate pairs by manually examining each word pair and choosing from the list of candidate alignments. These determinations were made qualitatively, based on detailed knowledge from our previous study of the historical relationships among these languages, including sound changes, drift, and inheritance. We then analyzed the performance of the default ALINE parameters for identifying the best possible alignment and compared it to the performance of optimized parameters determined by the genetic algorithm and the supervised learning methodology outlined above. The results of the supervised learning procedure are shown in Figure 3.

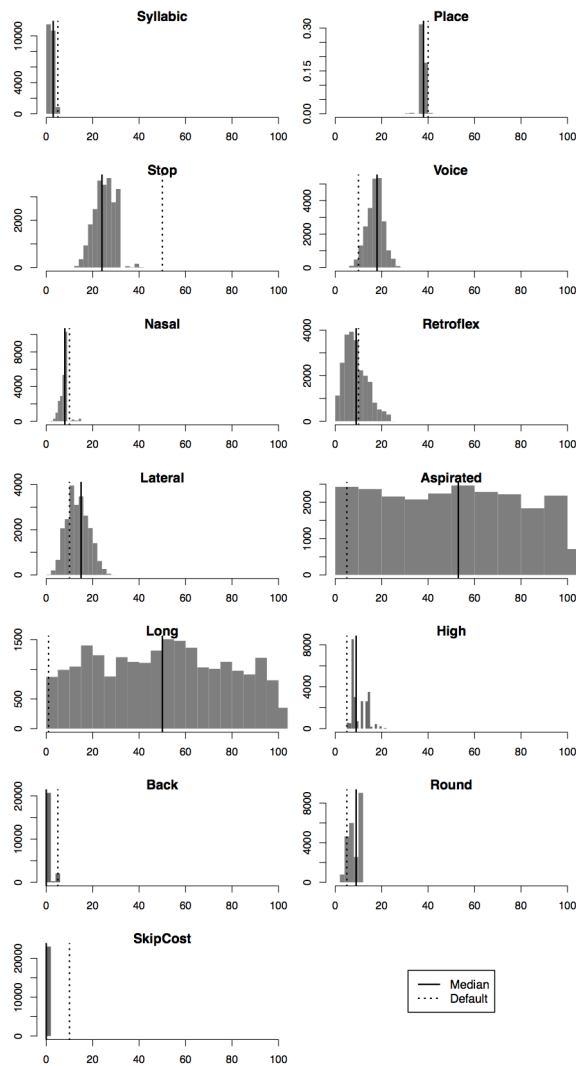


Figure 3: Distribution of optimized feature weights

Alignment accuracy using optimized and default weights

We analyze the ability of the GA to estimate unknown parameters using the alignments manually selected by the linguist. Figure 2 shows that GA performance converges at a lower level when analyzing these alignments. However, suboptimal performance is expected because the manual coding process necessitates trade-offs between optimal features for aligning any given cognate pair

and the optimal values for the entire training set. Nevertheless, we find that the optimized parameters performed better than the default parameters. Since the ALINE algorithm encodes linguistic rules, it is expected to achieve a certain number of correct alignments regardless of parameter weights. We find that this is the case: the median number of correct alignments using simulated random parameters is 19, while the default parameters achieve 118 correct alignments, and the optimized parameters achieve 158. Thus, our results show that in comparison to randomly selected feature weights, the optimal feature weights are approximately 29% more accurate than the defaults.

Statistical analysis of optimized and default weights

Next, we test whether the difference between the parameter estimates from the GA and the default values are statistically significant. The population of optimized candidate parameter values from the GA is used to determine empirical distributions for each linguistic parameter and to calculate the probability for each. The default values, median estimated values, and probabilities are reported in Table 3. We find 9/13 features differed from the default values. We include $\alpha \leq .10$ because of the size of the training sample – a larger data set is expected to return even more significant results. Regardless of sampling, it is expected that not all features would be significant in a given language family because certain features are more important than others for classification tasks (Kondrak, 2003). In two cases (aspirated, long), the estimated parameters differ significantly from the defaults based on the empirical distributions, but in these cases the algorithm is not sensitive to these parameters, most likely because those features are not prominently represented in the training data set.

Parameter	Default	Optimized	Pr($X < > x$)
Syllabic	5	3	*** 0.0022
Place	40	38	** 0.0108
Stop	50	26	*** $< 1.0^{-16}$
Voice	10	18	* 0.0506
Nasal	10	8	* 0.0626
Retroflex	10	9	0.6972
Lateral	10	14	0.4710
Aspirated	5	52	* 0.0946
Long	1	52	** 0.0194
High	5	9	*** 0.0004
Back	5	0	0.1580
Round	5	9	0.1762
Skip	10	0	*** $< 1.0^{-16}$

Sig. levels: $\alpha \leq .01$ (***); $\alpha \leq .05$ (**); $\alpha \leq .1$ (*)

Table 3: Summary statistics for a population of optimized parameters

Cross-validation

To determine whether the resulting optimized parameters can correctly align reserved data, we use leave-one-out cross-validation. We iteratively optimize parameters using 202/203 of the cognate pairs from the training data such that each cognate pair is reserved once. After each iteration, the optimized parameters are used to align the reserved pair and a success is recorded when the automated alignment matches the linguist-determined alignment. In this binary classification routine the default parameters successfully predicted 118/203 (58%) of the manual determinations and the optimized parameters predicted 151/203 (75%).

Statistical analysis of distances generated with optimized and default weights

Finally, to determine whether optimizing parameters affects the linguistic distance measurements, we analyze the distribution of the pairwise differences between the distances calculated by the optimized and default parameters. A Wilcoxon signed rank test for paired samples finds that the median optimized distance is 0.0096 greater than the default median and that this difference is statistically significant ($M_o = 0.1896, M_d = 0.1800, V = 9247, p < 2.612 \times 10^{-07}$). We also note that there are instances when the difference between the default and optimized distances is less than 0. This suggests that optimizing linguistic parameters may have nonlinear effects on the resulting distances.

Future research: potential synchronic uses for *alineR*

Although our work has been used primarily for diachronic applications (the generation of phylogenetic trees), it has potential synchronic applications as well.¹ One potential use involves the computation of what are known as phonological neighbors – words which are similar to a target word but differ from that target by a single phoneme. The total number of words fitting this definition can be described as a phonological neighborhood. A word's neighborhood size can affect the way we understand and use it. When we hear a word, everything that sounds like that word becomes slightly more accessible in memory. It therefore takes listeners longer to determine what a word is when that word has several neighbors (Luce and Pisoni, 1998). Take, for example, the phonological neighborhood of *cat*, where each word in the neighborhood differs from *cat* by a single phoneme (either consonant or vowel):

cat cab, cad, can, cam, cap, cart, cash, catch, caught, coat, cot, court, cut, bat, chat, fat, hat, kit, kite, mat, pat, rat, sat, that

With a large neighborhood such as the one above, it is more difficult to eliminate the words that were *not* said; *cot* or *cap* for example, rather than *cat*. A word like *green*, which has fewer neighbors than *cat*, is less confusable, and thus requires less work to identify:

green grain, gran, greed, grief, grease, Greece, Greek, greet, grin, grown, groan

While phonological neighbors can be viewed in terms of differing segments, a finer-grained analysis may also break these segments down into phonological features. For example, *cad* differs from *cat* by one feature: voicing in the third segment. On the other hand, *cab* differs from *cat* in both voicing and place of articulation of the final segment. Likewise, *grain* differs from *green* by one feature in the vowel (height), but *groan* differs from *green* by three features (height, backness, and rounding).

As a feature-driven algorithm, *ALINE* could allow the members of a phonological neighborhood to be mapped in a quantitative way based on phonological distance. The resulting neighborhood maps can then be used in psycholinguistic experiments, testing the hypothesis that phonological distance between neighbors at the feature level correlates with both access times and production values.

Another way in which *ALINE* could be used synchronically involves certain kinds of speech errors and neologisms. Strictly phonological errors include metathesis, sound-exchange errors, and spoonerisms. These are all instances in which the linear order of a pair of phonemes is reversed, as in the examples below:

Metathesis pus pocket > pos pocket

Sound-exchange error night life > knife light

Spoonerism light a fire > fight a liar

Other forms of phonological errors include additions and deletions:

Addition optimal > moptimal

Deletion unanimity > unanimity

Still other forms include anticipation and perseveration, where a phoneme (or sequence of phonemes) in one word influences the articulation of another word in the same phrase:

Anticipation reading list > leading list

Perseveration black boxes > black bloxes

With *ALINE*'s feature-driven alignment capabilities, and given an adequate corpus of errors, generalizations could be quickly drawn about both the position of errors within the word as well as the frequency with which various phonemes participate in unitary instances as well as in pairs.

A final example, which occurs in both speech errors and intentional neologisms, is what are known as blends:

channel x *tunnel* > *chunnel*

breakfast x *lunch* > *brunch*

The *ALINE* algorithm could foster some unique insights in these cases. In unmarked blends such as *chunnel* and *brunch*, the initial part of the first donor word (the onset of the initial syllable in these cases) replaces the same in the second donor word. On the other hand, in the unintentional blend *perple*, an entire syllable is copied from the first donor word:

¹We would like to thank an anonymous reviewer for suggesting the inclusion of this section.

person x people > perple

At first it is not clear why this example differs from the ones given above. However, one fact which stands out is that the initial consonant [p] is common to both of the input forms as well as the output form (a fact which would be reflected in the output score of an ALINE comparison). Since both donor words have the same initial consonant, if the same rule was applied here which operated in the *chunnel* and *brunch* examples, the hypothetical blend would be indistinguishable from the second donor word:

person x people > people

A more ambiguous case is that of *motel*, where the blend contains two phonemes (a vowel and a consonant) which are shared between the donor words:

motor x hotel > motel

An ALINE analysis would indicate this ambiguity via the output score which correlates with the shared material overlapping between the two donor words.

ALINE could therefore be used to quickly analyze and categorize different kinds of blends (particularly in the case of a large corpus), with an eye toward answering questions such as what determines the size of the constituents of a blend (how many segments or syllables are copied from the donor words to the blend), what determines the type of constituents in a blend (i.e. are they syllable onsets, full syllables, and so on), and what role segmental overlap plays in blend formation. While an analysis of these phenomena lies outside the scope of this paper, given the appropriate type and size of corpi, we consider the examples above to be fruitful prospects for future research in phonology and psycholinguistics.

Conclusion

In conclusion, **alineR** is a new R package for calculating and optimizing feature-weighted linguistic distances. It is now available on CRAN. Supplemental materials accompanying this paper include an R script ('downey.R') for the commands and analyses included above. The linguistic data from Sumba used for training the GA is in the train object in 'downey.Rdata'. We suggest that optimized feature-weighted linguistic distances are an important complement to other linguistic distances such as the edit or Levenshtein distance. In addition to calculating the ALINE distance and returning relevant alignment information and similarity scores, the package provides a genetic algorithm that can be used to estimate linguistic parameters using supervised learning. It may be particularly useful for bioinformatic applications in anthropology and historical linguistics or in comparisons with well-resolved quantitative distance measurements (e.g., Fst). As such, it has the potential to help advance our understanding of the evolutionary relationships between languages and genetics. Not only can this help uncover historical demographic patterns, but coevolutionary analyses using the ALINE Distance may provide insight into general processes of biological and cultural evolution.

Acknowledgment

The authors wish to express their appreciation to Murray Cox for his input on early drafts of this paper and for his guidance submitting **alineR** to CRAN. Steve Lansing provided access to the Sumbanese language data set, which he collected during his previous projects. We also thank Tanmoy Bhattacharya, Aakrosh Ratan, and the other participants of the Santa Fe Institute Workshop, "Co-evolution of Social Structure and Communication in Different Genotypes," May 26 - 29, 2015 for critical and constructive input.

Funding

This work was supported by the National Science Foundation [Grant number SBS-1030031 to P.N. and S.D.], and the University of Maryland.

Bibliography

- I. P. Association. *Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet*. Cambridge University Press, 1999. [p140]
- T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997. [p143]

- G. Barbujani and R. R. Sokal. Zones of sharp genetic change in europe are also linguistic boundaries. *Proceedings of the National Academy of Sciences*, 87(5):1816–1819, 1990. URL <https://doi.org/10.1073/pnas.87.5.1816>. [p138]
- A. Borg and M. Sariyar. *RecordLinkage: Record Linkage in R*, 2016. URL <https://CRAN.R-project.org/package=RecordLinkage>. R package version 0.4-10. [p139]
- L. L. Cavalli-Sforza. Genes, peoples and languages. *Scientific American*, 265(5):104–110, 1991. URL <https://doi.org/10.1038/scientificamerican1191-104>. [p138]
- L. L. Cavalli-Sforza, E. Minch, and J. Mountain. Coevolution of genes and languages revisited. *Proceedings of the National Academy of Sciences*, 89(12):5620–5624, 1992. URL <https://doi.org/10.1073/pnas.89.12.5620>. [p138]
- J. Chen, R. R. Sokal, and M. Ruhlen. Worldwide analysis of genetic and linguistic relationships of human populations. *Human Biology*, pages 595–612, 1995. URL <https://doi.org/10.3378/027.084.0506>. [p138]
- M. A. Covington. Alignment of multiple languages for historical comparison. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 275–279. Association for Computational Linguistics, 1998. [p138]
- M. P. Cox. *Genetic Patterning at Austronesian Contact Zones*. PhD thesis, University of Otago, 2003. [p138]
- M. P. Cox and M. M. Lahr. Y-chromosome diversity is inversely associated with language affiliation in paired austronesian-and papuan-speaking communities from solomon islands. *American Journal of Human Biology*, 18(1):35–50, 2006. URL <https://doi.org/10.1002/ajhb.20459>. [p138]
- W. Croft. Evolutionary linguistics. *Annual Review of Anthropology*, 37(1):219, 2008. URL <https://doi.org/10.1146/annurev.anthro.37.081407.085156>. [p138]
- J. Diamond and P. Bellwood. Farmers and their languages: The first expansions. *Science*, 300(5619):597–603, 2003. ISSN 0036-8075. URL <https://doi.org/10.1126/science.1078208>. [p138]
- S. S. Downey, B. Hallmark, M. P. Cox, P. Norquest, and J. S. Lansing. Computational feature-sensitive reconstruction of language relationships: Developing the ALINE distance for comparative historical linguistic reconstruction. *Journal of Quantitative Linguistics*, 15(4):340–369, 2008. URL <https://doi.org/10.1080/09296170802326681>. [p138, 139]
- A. B. Fine and T. F. Jaeger. Evidence for implicit learning in syntactic comprehension. *Cognitive Science*, 37(3):578–591, 2013. URL <https://doi.org/10.1111/cogs.12022>. [p138]
- C. Gooskens and W. Heeringa. Perceptive evaluation of levenshtein dialect distance measurements using norwegian dialect data. *Language Variation and Change*, 16:189–208, 2004. URL <https://doi.org/10.1017/s0954394504163023>. [p138]
- F. Horn, A. L. Lau, and F. E. Cohen. Automated extraction of mutation data from the literature: Application of mutext to g protein-coupled receptors and nuclear hormone receptors. *Bioinformatics*, 20(4):557–568, 2004. URL <https://doi.org/10.1093/bioinformatics/btg449>. [p138]
- P. A. Huff. *PyAline: Automatically growing language family trees using the ALINE distance*. Master’s thesis, Brigham Young University-Provo, 2010. [p139]
- K. Hunley and J. C. Long. Gene flow across linguistic boundaries in native North American populations. *Proceedings of the National Academy of Sciences of the United States of America*, 102(5):1312–1317, 2005. URL <https://doi.org/10.1073/pnas.0409301102>. [p138]
- G. Kondrak. A new algorithm for the alignment of phonetic sequences. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, pages 288–295. Association for Computational Linguistics, 2000. [p138]
- G. Kondrak. Phonetic alignment and similarity. *Computers and the Humanities*, 37(3):273–291, 2003. URL <https://doi.org/10.1023/A:1025071200644>. [p146, 148]
- J. S. Lansing, M. P. Cox, S. S. Downey, B. M. Gabler, B. Hallmark, T. M. Karafet, P. Norquest, J. W. Schoenfelder, H. Sudoyo, J. C. Watkins, and others. Coevolution of languages and genes on the island of sumba, eastern indonesia. *Proceedings of the National Academy of Sciences*, 104(41):16022–16026, 2007. URL <https://doi.org/10.1073/pnas.0704451104>. [p138, 147]

- V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966. [p138]
- P. A. Luce and D. B. Pisoni. Recognizing spoken words: The neighborhood activation model. *Ear and Hearing*, 19(1):1, 1998. URL <https://doi.org/10.1097/00003446-199802000-00001>. [p149]
- D. Nettle and L. Harriss. Genetic and linguistic affinities between human populations in Eurasia and West Africa. *Human Biology*, pages 331–344, 2003. URL <https://doi.org/10.1353/hub.2003.0048>. [p138]
- M. Pagel. *Wired for Culture: Origins of the Human Social Mind*. WW Norton & Company, 2012. [p138]
- H. Pagès, P. Aboyou, R. Gentleman, and S. DebRoy. *Biostrings: String Objects Representing Biological Sequences, and Matching Algorithms*, 2017. URL <http://bioconductor.org/packages/Biostrings>. R package version 2.44.0. [p139]
- F. Petroni and M. Serva. Measures of lexical distance between languages. *Physica A: Statistical Mechanics and its Applications*, 389(11):2280–2283, 2010. URL <https://doi.org/10.1016/j.physa.2010.02.004>. [p138]
- L. QiuJun. Extraction of news content for text mining based on edit distance. *Journal of Computational Information Systems*, 6(11):3761–3777, 2010. [p138]
- Revolution Analytics and Steve Weston. *doMC: Foreach Parallel Adaptor for 'parallel'*, 2015. URL <https://CRAN.R-project.org/package=doMC>. R package version 1.3.4. [p145]
- P. E. Smouse and J. C. Long. Matrix correlation analysis in anthropology and genetics. *American Journal of Physical Anthropology*, 35(S15):187–213, 1992. URL <https://doi.org/10.1002/ajpa.1330350608>. [p138]
- R. R. Sokal. Genetic, geographic, and linguistic distances in Europe. *Proceedings of the National Academy of Sciences*, 85(5):1722–1726, 1988. URL <https://doi.org/10.1073/pnas.85.5.1722>. [p138]
- H. L. Somers. Similarity metrics for aligning children’s articulation data. In *Proceedings of the 17th International Conference on Computational Linguistics-Volume 2*, pages 1227–1232. Association for Computational Linguistics, 1998. [p138]
- M. K. Tumonggor, T. M. Karafet, S. Downey, J. S. Lansing, P. Norquest, H. Sudoyo, M. F. Hammer, and M. P. Cox. Isolation, contact and social behavior shaped genetic diversity in West Timor. *Journal of Human Genetics*, 59(9):494–503, 2014. URL <https://doi.org/10.1038/jhg.2014.62>. [p138]
- M. P. van der Loo. The stringdist package for approximate string matching. *The R Journal*, 6(1):111–122, 2014. URL <https://journal.r-project.org/archive/2014/RJ-2014-011/index.html>. [p139]
- S. Wichmann, E. W. Holman, D. Bakker, and C. H. Brown. Evaluating linguistic distance measures. *Physica A: Statistical Mechanics and its Applications*, 389(17):3632–3639, 2010. URL <https://doi.org/10.1016/j.physa.2010.05.011>. [p138]

Sean S. Downey
Department of Anthropology, University of Maryland
4302 Chapel Lane, College Park, MD 20742
United States
sdowney2@umd.edu

Guowei Sun
Department of Mathematics, University of Maryland
176 Campus Drive, College Park, MD 20742
United States
gwsun@umd.edu

Peter Norquest
Department of Anthropology, University of Arizona
1009 E. South Campus Drive, Tucson, AZ 85721
United States
norquesp@email.arizona.edu

Implementing a Metapopulation Bass Diffusion Model using the R Package deSolve

by Jim Duggan

Abstract Diffusion is a fundamental process in physical, biological, social and economic settings. Consumer products often go viral, with sales driven by the word of mouth effect, as their adoption spreads through a population. The classic diffusion model used for product adoption is the Bass diffusion model, and this divides a population into two groups of people: potential adopters who are likely to adopt a product, and adopters who have purchased the product, and influence others to adopt. The Bass diffusion model is normally captured in an aggregate form, where no significant consumer differences are modeled. This paper extends the Bass model to capture a spatial perspective, using metapopulation equations from the field of infectious disease modeling. The paper’s focus is on simulation of deterministic models by solving ordinary differential equations, and does not encompass parameter estimation. The metapopulation model is implemented in R using the **deSolve** package, and shows the potential of using the R framework to implement large-scale integral equation models, with applications in the field of marketing and consumer behaviour.

Introduction

Diffusion is a fundamental process in physical, biological, social and economic settings (Rahmandad and Sterman, 2008). In the business arena, consumer products frequently go viral by the word of mouth effect between consumers. In this scenario, products experience rapid sales growth, which eventually slows as the number of potential adopters decline. A classic dynamic model to capture these growth processes is the Bass model (Bass, 1969), which is based on the assumption that the timing of purchases is related to the number of previous buyers. This model provided good predictions of the sales peak, and the timing of the peak when compared to historical data. Providing model-based estimates of peak timing can assist with production capacity planning and sales distribution strategies, for example, to ensure that enough product items are available in time to match future demand. A representation of the Bass model is shown in Figure 1, and this captures the scenario whereby newer technologies are not immediately adopted by all potential buyers, and a diffusion process is set into motion (Norton and Bass, 1987). This version of the Bass model is informed by standard epidemiological diffusion models (Vynnycky and White, 2010), and so the term *force of persuasion* is analogous to the epidemiological term *force of infection*.

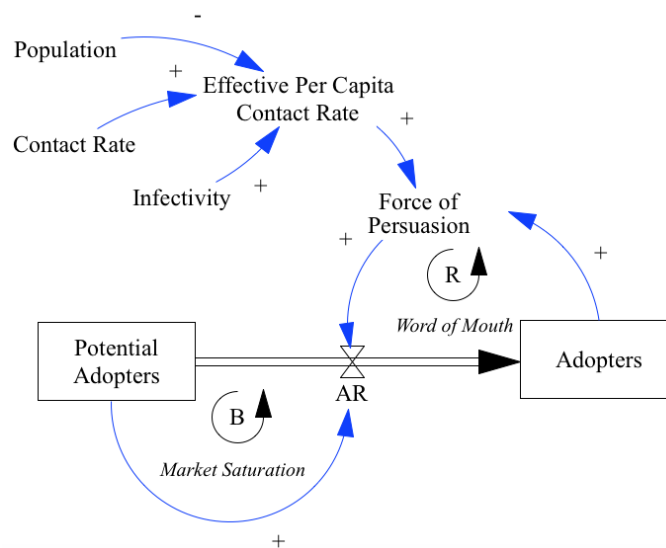


Figure 1: Market diffusion model, aggregate version. *AR* represents the adoption rate (derivative), and + and - refer to positive and negative association respectively. The symbol *B* represents a negative feedback loop (balancing), while *R* indicates a positive feedback loop (reinforcing).

The modeling methodology used in this paper is system dynamics (Sterman, 2000; Duggan, 2016), a method for modeling social systems, which focuses on how feedback structures (i.e. causal loops) impact overall system behavior. Feedback is a defining element of system dynamics (Lane, 2006), where a feedback loop is a chain of circular causal links represented in a model. There are two types of feedback which impact a variable: *negative feedback* which opposes the direction of change in a loop variable (for example, a thermostat controller that regulates heat in a room), whereas *positive feedback* amplifies a variable's value, as in the case of the *contagion effect* in infectious disease spread. At a technical level, system dynamics models are solved using integration, where integrals are referred to as stocks, and derivatives are modeled as flows. System dynamics diagrams are intuitive, as they present a dynamic system in terms of the stocks (containers), and flows (valves), which control the rate at which stocks fill and drain over time.

Bass diffusion model

The Bass diffusion model has two integrals: *Potential Adopters* (1), which model individuals in the population that have yet to acquire a product, and *Adopters* (2), which represent people who have purchased a product, and are in a position to influence others to initiate a purchase. For this paper, the initial values of these stocks are 99,999 and 1 respectively, giving a total population size of 100,000. These initial values will also be maintained across the metapopulation model, which provides a useful basis for comparing the outputs of both models.

$$\text{Potential Adopters} = \int_0^t -AR \, dt \quad (1)$$

$$\text{Adopters} = \int_0^t AR \, dt \quad (2)$$

The flow equation AR (8) captures a number of concepts. First, there is the idea of the effective per capita contact rate, known by the parameter β (3), which is based on:

- The contact rate (4) between members of the population, which is an estimate of how often people interact.
- The infectivity (5) of these interactions, which is a way of modeling how likely it is for an individual to be convinced to purchase, based on an interaction with an adopter.
- The total population (6), which is the denominator of the equation.

$$\beta = \frac{\text{ContactRate} \times \text{Infectivity}}{\text{Population}} \quad (3)$$

$$\text{ContactRate} = 3 \quad (4)$$

$$\text{Infectivity} = 0.15 \quad (5)$$

$$\text{Population} = 100,000 \quad (6)$$

Given the value for β , it is then possible to calculate the force of persuasion ρ , which defines the proportion of potential adopters that will convert at each time period. This equation (7) is similar to the epidemiological term known as the *attack rate*, which determines how many susceptible people become infected per time period in classic infectious disease models (Anderson et al., 1991).

$$\rho = \beta \times \text{Adopters} \quad (7)$$

With the value for ρ evaluated, the adoption rate is simply the product of this value with the number of potential adopters in the population, as shown in (8).

$$AR = \rho \times \text{Potential Adopters} \quad (8)$$

These equations have three properties that add to the robustness of the model, These are:

- if there are no contacts in the population, β will equal 0, and no adoption will occur;
- if infectivity is zero, β will also equal 0, which will stop any adoption;
- if there are no adopters, then the force of persuasion will be zero, and no adoptions will occur.

Equations (1)-(8) can be implemented using the **deSolve** package (Soetaert et al., 2010, 2016), which solves initial value problems written as ordinary differential equations (ODE), differential algebraic equations (DAE), and partial differential equations (PDE). For modeling diffusion processes, the R package **EpiDynamics** (Baquero and Marques, 2015) was also considered, however it does not currently provide functionality to solve two-compartment SI models, which forms the basis of the Bass model. The R package **EpiModel** (Jenness et al., 2016) also contains an API that is flexible to formulate a number deterministic compartmental models. For our scenario, the ODE solver in **deSolve** is used, where the differential equations are encapsulated in a function.

The initial segment of code¹ loads the packages, sets the simulation start time, finish time, and time step, and constructs the appropriate time vector that is needed by **deSolve**. Following that, the two integrals (stocks) are created, along with their initial values, and auxiliary parameters initialised within a vector. In naming variables, Hungarian Notation (Li and Prasad, 2005) is used, to distinguish between stocks (s), flows (f) and exogenous parameters (a), known in system dynamics as auxiliaries.

```
library(deSolve)

START <- 0; FINISH <- 50; STEP <- 0.01;
simtime <- seq(START, FINISH, by=STEP)

stocks <- c(sPotentialAdopters=99999, sAdopters=1)
auxs <- c(aTotalPopulation=100000, aContact.Rate=3, aInfectivity=0.15)
```

Next the function containing the model equations is constructed. This is called by the **deSolve** framework for every solution step of the ODE, and the parameters passed in include:

- The current simulation time,
- A vector containing all integrals (stocks) and their values,
- A vector containing the parameters used in the stock and flow model.

The R code for the function is shown below, and the corresponding model equations are also referred to.

```
model <- function(time, stocks, auxs){
  with(as.list(c(stocks, auxs)),{
    aBeta <- aContact.Rate * aInfectivity/ aTotalPopulation # Eqn (3)
    aRho <- aBeta * sAdopters # Eqn (7)
    fAR <- sPotentialAdopters * aRho # Eqn (8)
    dPA_dt <- -fAR # Eqn (1)
    dA_dt <- fAR # Eqn (2)

    return (list(c(dPA_dt, dA_dt),
                  AR=fAR, Rho=aRho))
  })
}
```

To run the simulation, the function `ode()` is called with five parameters, including the vector of stocks, the simulation time vector, the callback model function, any exogenous parameters (none in this case), and the numerical integration algorithm to use (Euler's method is deployed, as this is commonly used as an integration method for social systems (Sterman, 2000)). The results from `ode()` are then wrapped in a data frame, in order to make use of `ggplot()`.

```
o<-data.frame(ode(y=stocks, times=simtime, func = model,
                  parms=auxs, method="euler"))
```

Figure 2 shows the classic behavior of the Bass model, as the dynamics of the adopter exhibit s-shaped growth as the customer base saturates to its maximum possible value over time.

In summary, the aggregate model is useful to capture the overall dynamics of the diffusion process. However, for spatial diffusion models a revised equation structure is needed, and one such model based on the dynamics of infectious disease transmission - is now described.

¹See supplemental material included with the paper

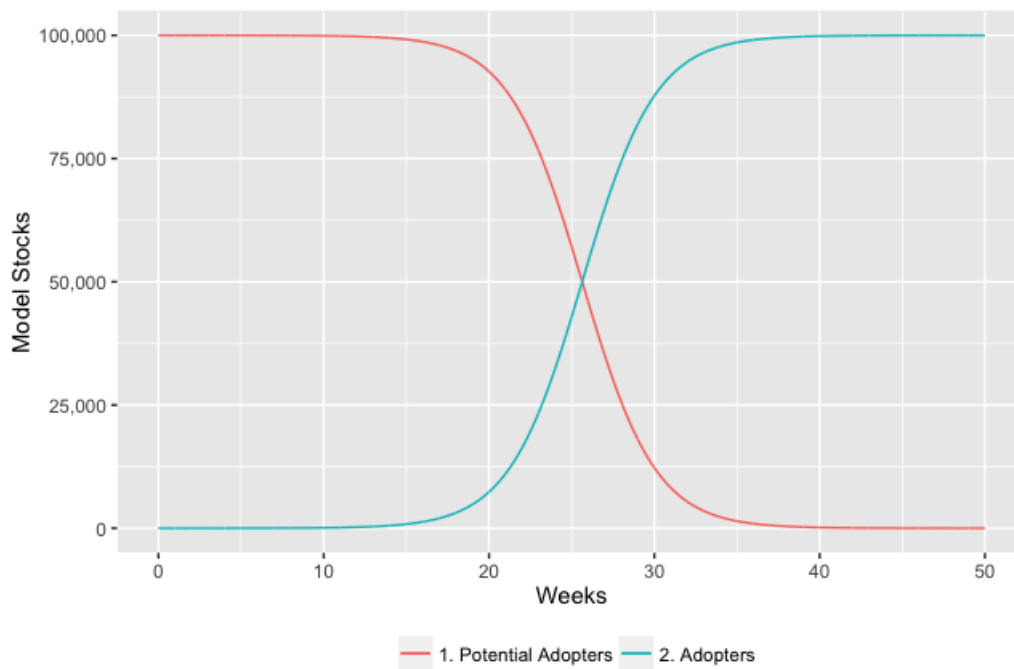


Figure 2: Simulation of the Bass model

Metapopulation diffusion model

Metapopulation structures are a powerful feature in dynamic modeling, as they facilitate the investigation of finer-grained behaviours and dynamics than can be achieved through an aggregate model. The metapopulation concept is to subdivide the entire population into distinct *subpopulations*, each with independent dynamics, and combine this with limited interaction between each subpopulation (Keeling and Rohani, 2008). The approach is appropriate if there are significant variations in model parameters across the population. For example, in disease dynamics, empirical studies show non-random mixing in populations, including data on the transmission of tuberculosis (Borgdorff et al., 1999), and contact patterns across Europe show highly assortative mixing patterns with age (Mossong et al., 2008). A similar argument can be made for product adoption models, and for this scenario, a regional spatial structure is proposed as the main mechanism for subdividing the model. Figure 3 shows a hypothetical regional structure, where the population centers (colours reflect population density) are in the middle of the region, and models an unbalanced regional development which is a feature of many countries.

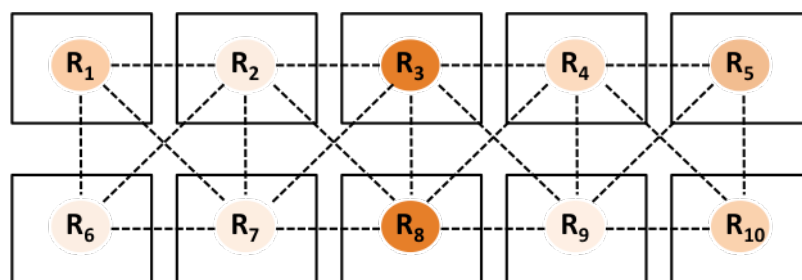


Figure 3: Spatial structure for metapopulation diffusion model

Out of an overall population of 100,000, the proportions in each region are captured in Table 1.

R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
10%	1%	21%	8%	17%	4%	5%	25%	3%	6%

Table 1: Population proportions by region

At a structural level, the diffusion model has 20 stocks, based on the stock and flow structure in Figure 1. The integral equations (1) and (2) are replicated for each region, and each region has its own adoption rate (8). This adoption rate is formulated as the product of each region's force of persuasion and the number of adopters, as specified in (9).

$$AR_i = \rho_i \times Potential\ Adopters_i \tag{9}$$

The force of persuasion ρ_i reflects the spatial properties of the model, and is a weighted sum of the per capita effective contact rates from region j to region i , multiplied by the number of adopters (A_j) in region j (10). This structure is based on standard epidemiological transmission equations between cohorts, where the j^{th} cohort is infectious (an adopter), and the i^{th} cohort is susceptible (a potential adopter).

$$\rho_i = \sum_{j=1}^N \beta_{ij} \cdot A_j \tag{10}$$

Equation (10) can also be conveniently expressed in matrix form, as shown in (11).

$$\begin{pmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_N \end{pmatrix} = \begin{pmatrix} \beta_{11} & \beta_{12} & \dots & \beta_{1N} \\ \beta_{21} & \beta_{22} & \dots & \beta_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{N1} & \beta_{N2} & \dots & \beta_{NN} \end{pmatrix} \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{pmatrix} \tag{11}$$

The β component, originally introduced in (3), is the effective contact rate per capita, and therefore the square matrix component of (11) can be further factored into (12) which shows the transmission calculations between each region. The denominator is the same for each matrix row, and is the population of each region containing the potential adopters.

$$\beta_{ij} = e_{ij} / N_i \tag{12}$$

The effective contact values (e) are based on the contact rates between the different sectors. The contact rates are calculated from the normal contact rate for a region (n), multiplied by a distancing weighting measure using a power law function (Brockmann et al., 2006), useful for individual-based models (Mungovan et al., 2011; Liu et al., 2012), given that these structures have also been successfully implemented in models of infectious disease (Meyer et al., 2014). This equation is shown in (13), and the distance is the Euclidean distance function that uses the grid coordinates as a reference (14), given the points (x_1, y_1) and (x_2, y_2) .

$$c_{ij} = n_j \cdot (d_{ij} + 1)^{-\alpha} \tag{13}$$

$$d_{ij} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{14}$$

To provide flexibility for experimentation, the parameter α (15) provides scope to moderate the contact rates between the different regions. With $\alpha = 0$, the contact rates do not change from the original region to other regions, whereas higher values will cause contact rates to decrease as the distance between regions increases.

$$\alpha \geq 0 \tag{15}$$

The effective contact rates (e), which model the word of mouth effect from region to region is shown in (16), which is the product of the contact rates (c) and infectivity (f), which models the probability of transmission. This equation provides the necessary information to process the matrix specified in (11).

$$e_{ij} = c_{ij} \cdot f_j \tag{16}$$

Tables 2-5 show sample parameter values for the model. In Table 2, sample normal contact rates per area are shown, with R5 and R8 having the highest values, given that these have the highest population proportions. Table 3 summarises arbitrary infectivity values, which do not vary significantly from region to region, as an individual's ability to persuade others is not assumed to be dependent on the overall population density. Table 4 shows the distance values calculated between each region, where the coordinates are based on the row/column location of a region, shown earlier in Figure 3. Table 5 then shows the final contact rates, using (13), with $\alpha = 1.0$, which shows that the contact rate falls as

R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
4.58	0.56	10.12	4.61	10.00	1.64	2.53	14.46	1.53	2.95

Table 2: Sample normal contact rates (n) per area, weighted by population density

R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
0.05	0.03	0.04	0.03	0.01	0.05	0.02	0.01	0.02	0.05

Table 3: Sample infectivity probabilities (f), by region (uniform random numbers)

the distance between regions increases, according to the power law function.

The metapopulation Bass diffusion model in R

The R implementation for the metapopulation Bass model is now presented. First, the relevant libraries are attached, and these include utilities to support plotting (`ggplot2` and `scales`), preparation of data (`reshape2` and `dplyr`), and for performing numerical integration (`deSolve`).

```
library(deSolve)
library(ggplot2)
library(scales)
library(reshape2)
library(dplyr)
```

Initially, a data frame is used to specify the model topology, although spatial data stored in GIS files could also be used as part of the implementation (Bivand et al., 2013, pp. 1–16).

```
TotalPopulation<-100000
name.reg<-c("R1","R2","R3","R4","R5",
            "R6","R7","R8","R9","R10")
row.reg<-c(1,1,1,1,1,2,2,2,2,2)
col.reg<-c(1,2,3,4,5,1,2,3,4,5)
pop.reg<-c(0.10,0.01,0.21,0.08,0.17,
           0.04,0.05,0.25,0.03,0.06)

sp<-data.frame(Regions=name.reg,
               Row=row.reg,
               Col=col.reg,
               Pop=pop.reg*TotalPopulation)
```

Before running the simulation, the $N \times N$ matrix from equation (11) needs to be calculated, based on equations (12), (13), (14), (15) and (16). The normal contact rate is drawn from a random uniform

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
R1	0.00	1.00	2.00	3.00	4.00	1.00	1.41	2.24	3.16	4.12
R2	1.00	0.00	1.00	2.00	3.00	1.41	1.00	1.41	2.24	3.16
R3	2.00	1.00	0.00	1.00	2.00	2.24	1.41	1.00	1.41	2.24
R4	3.00	2.00	1.00	0.00	1.00	3.16	2.24	1.41	1.00	1.41
R5	4.00	3.00	2.00	1.00	0.00	4.12	3.16	2.24	1.41	1.00
R6	1.00	1.41	2.24	3.16	4.12	0.00	1.00	2.00	3.00	4.00
R7	1.41	1.00	1.41	2.24	3.16	1.00	0.00	1.00	2.00	3.00
R8	2.24	1.41	1.00	1.41	2.24	2.00	1.00	0.00	1.00	2.00
R9	3.16	2.24	1.41	1.00	1.41	3.00	2.00	1.00	0.00	1.00
R10	4.12	3.16	2.24	1.41	1.00	4.00	3.00	2.00	1.00	0.00

Table 4: Regional distance matrix based on Euclidean distance

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
R1	4.58	0.28	3.37	1.15	2.00	0.82	1.05	4.47	0.37	0.58
R2	2.29	0.56	5.06	1.54	2.50	0.68	1.26	5.99	0.47	0.71
R3	1.53	0.28	10.12	2.31	3.33	0.51	1.05	7.23	0.63	0.91
R4	1.14	0.19	5.06	4.61	5.00	0.39	0.78	5.99	0.77	1.22
R5	0.92	0.14	3.37	2.31	10.00	0.32	0.61	4.47	0.63	1.47
R6	2.29	0.23	3.13	1.11	1.95	1.64	1.26	4.82	0.38	0.59
R7	1.90	0.28	4.19	1.43	2.40	0.82	2.53	7.23	0.51	0.74
R8	1.41	0.23	5.06	1.91	3.09	0.55	1.26	14.46	0.77	0.98
R9	1.10	0.17	4.19	2.31	4.14	0.41	0.84	7.23	1.53	1.47
R10	0.89	0.13	3.13	1.91	5.00	0.33	0.63	4.82	0.77	2.95

Table 5: Final contact rates (c) from region to region, with $\alpha = 1.00$

distribution based on the population proportion in each sector, where areas of higher density will have great interaction rates. The values for infectivity (a measure of the persuasiveness of an adopter) are also drawn from a uniform distribution, and the distance matrix is developed by calling the R function `dist()`. The value for α is arbitrarily chosen at 1.00.

```
normal.contacts<-runif(10,pop.reg*40,pop.reg*60)
infectivity<-runif(10,0.01,0.025)
names(infectivity)<-c("R1","R2","R3","R4","R5","R6","R7","R8","R9","R10")
ALPHA<-1.00
dm <- as.matrix(dist(sp[c("Col","Row")]))
```

In order to create the β matrix, a modified contact rate is calculated based on equation (16), and this is then transformed to an effective contact matrix by multiplying by the infectivity vector. The β matrix is then evaluated by simply dividing the effective contacts by the population of each sector, as specified earlier in equation (12).

```
cr <- t(normal.contacts*(dm+1)^-ALPHA)
ec <- t(t(cr)*infectivity)

beta <- ec/sp$Pop
```

With the contact information defined, the simulation model logic can then be implemented as part of the **deSolve** package. The key advantage here is that **deSolve** supports vectorization, and therefore the simulation model is scalable, as it uses matrix algebra to implement (11). Before defining the callback model function, the important simulation parameters are specified, including the start time, finish time, integration time step, the number of regions and the number of integrals (stocks) per region. The simulation time vector is defined, as well as a vector of integrals (20 in all), along with their initial values. For the initial conditions, the initial *customer zero* adopter is seeded in region eight (an arbitrary choice).

```
START<-0; FINISH<-50; STEP<-0.01;
NUM_REGIONS<-10; NUM_STOCKS_PER_REGION <-2

simtime <- seq(START, FINISH, by=STEP)

stocks <- c(PA_R1=sp$Pop[1], PA_R2=sp$Pop[2], PA_R3=sp$Pop[3],
            PA_R4=sp$Pop[4], PA_R5=sp$Pop[5], PA_R6=sp$Pop[6],
            PA_R7=sp$Pop[7], PA_R8=sp$Pop[8]-1, PA_R9=sp$Pop[9],
            PA_R10=sp$Pop[10], AD_R1=0, AD_R2=0,
            AD_R3=0, AD_R4=0, AD_R5=0,
            AD_R6=0, AD_R7=0, AD_R8=1,
            AD_R9=0, AD_R10=0)
```

In the model function, the stocks vector is converted into a 10x2 matrix, where all the **potential adopter** stocks are in the first column, and the **adopter** stocks reside in the second column. Matrix algebra is used to calculate the force of persuasion (12), and from this the adoption rates for each sector are calculated (9), followed by the derivatives. Additional information such as the individual adoption

rates, the total population, the total potential adopters and the total adopters are also returned from the function. A powerful feature of this function is that it is eminently scalable, and would work (assuming availability of sufficient computer memory) on much larger matrices and models.

```

model <- function(time, stocks, auxs){
  with(as.list(stocks),{
    states<-matrix(stocks,
                   nrow=NUM_REGIONS,
                   ncol=NUM_STOCKS_PER_REGION)

    PotentialAdopters <- states[,1]
    Adopters          <- states[,2]

    Rho               <- beta %%% Adopters      # Eqn (11)
    AR                <- Rho * PotentialAdopters # Eqn (9)

    dPA_dt           <- -AR                    # Based on Eqn(1)
    dAD_dt           <- AR                     # Based on Eqn(2)

    TotalPopulation  <- sum(stocks)
    TotalPotentialAdopters <- sum(PotentialAdopters)
    TotalAdopters    <- sum(Adopters)

    return (list(c(dPA_dt, dAD_dt),AR_R=AR,
                  TP=TotalPopulation,
                  TPA=TotalPotentialAdopters,
                  TAD=TotalAdopters))
  })
}

```

Once the model function has been specified, the **deSolve** package is called via the `ode()` function call, as described earlier. To prepare the data for visualization, it is filtered to focus on only the simulation data from discrete time steps (0, 1, 2, ..., 50), and this is melted into a three column data frame. The `grepl()` function is inside the `filter()` function to extract variables, and these are printed using the function `ggplot()`.

```

o<-data.frame(ode(y=stocks, times=simtime, func = model,
                 parms=NULL, method="euler"))

o1<-o[seq(from=1, to=length(simtime),by=1/STEP),]

tidy<-melt(o1,id.vars = "time")
names(tidy)<-c("Time", "Variable", "Value")

ar<-filter(tidy,grepl("AR_",Variable))
ad<-filter(tidy,grepl("AD_",Variable))

p1<-ggplot(ar,aes(x=Time,y=Value,color=Variable)) +
  geom_line() + geom_point()+
  ylab("Adoption Rate") + xlab("Time (Weeks)")

p2<-ggplot(ad,aes(x=Time,y=Value,color=Variable,group=Variable)) +
  geom_line() +
  geom_point()+
  ylab("Adopters") +
  xlab("Time (Weeks)")

```

The simulation output is displayed in Figures 4 and 5. The first of these figures shows the adoption rate (*AR*) values, which follow the classic bell-shaped contagion-like curve, as word of mouth spreads through the population with a corresponding increase in the "attack rate" or force of persuasion. As the number of potential adopters decline, so to does the adoption rate. The differences in peak times, although relatively small, are still clear on the diagram, which demonstrates the value of being able to model across spatial regions, and observe how the "epidemic", which is captured in the model equations, spreads across the different regions. The second plot captures the market saturation for each region, as potential adopters convert to adopters via the word of mouth diffusion process.

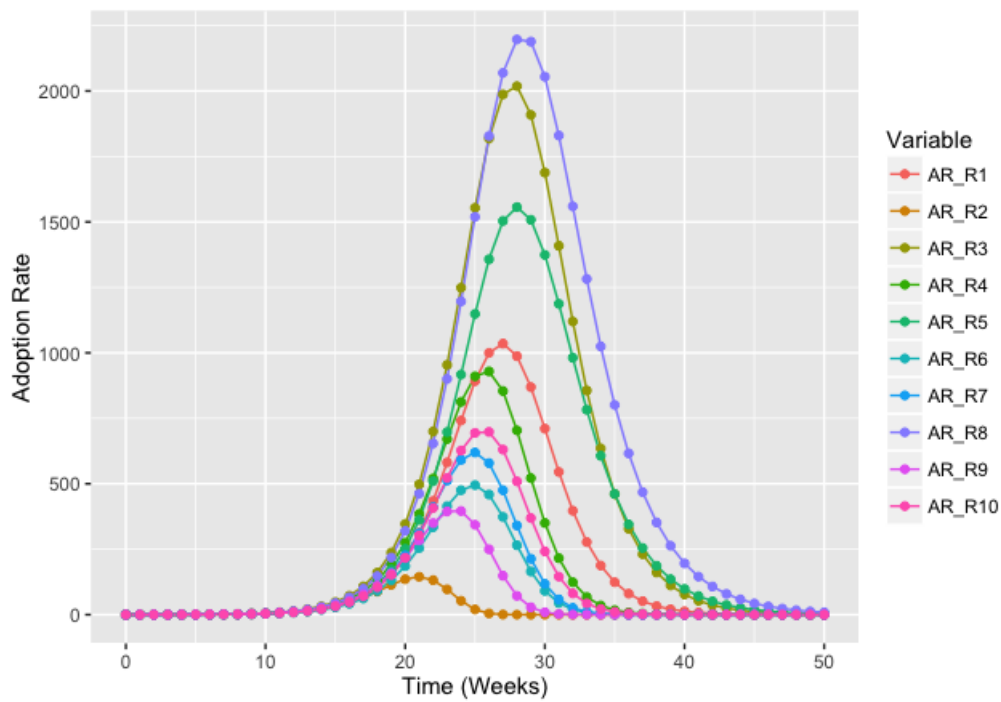


Figure 4: Simulation output: adoption rates by sector

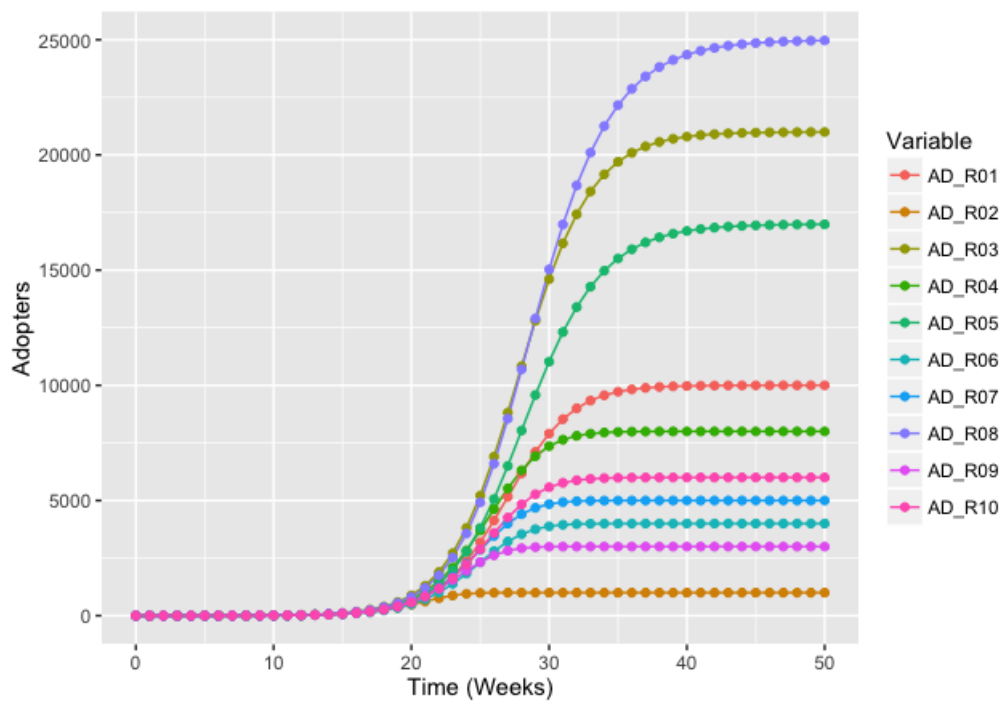


Figure 5: Simulation output: Adopter stocks behavior over time

While the model is based on hypothetical data, including the regional structure, the contact rates and infectivity, its underlying structure is robust and can be adapted for many spatial interaction processes. The simulation engine of the model is concise and can be used across a number of contexts, and real-world data can be conveniently used within the model, once it is prepared and resampled in matrix format.

Discussion

This paper shows how R can be used to model metapopulation integral models of product diffusion. For this class of problems, there are a number of advantages in considering R as a solution:

- R provides matrix support, and matrix equations such as the force of persuasion can be easily implemented.
- The package `deSolve` fully supports vectorisation, which means that the equation writing process is simplified, and the equation structures developed are scalable without having to alter the underlying model structure.
- R also provides excellent statistical analysis support, and model analysis techniques such as statistical screening (Ford and Flynn, 2005) can be conveniently scripted using R Duggan (2016, chapter 7).

Further extensions to the work would be to apply optimization techniques in order to fit product diffusion models to product sales data across regions, and therefore obtain more realistic simulations of the peak time (Vynnycky and Edmunds, 2008). These model extensions could also accommodate age cohort dynamics for market-based models, for example, those captured in epidemiological models such as influenza transmission (Keeling and Rohani, 2008; Vynnycky and White, 2010).

Bibliography

- R. M. Anderson, R. M. May, and B. Anderson. *Infectious Diseases of Humans: Dynamics and Control*. Oxford University Press, Oxford, UK., 1991. [p154]
- O. S. Baquero and F. S. Marques. *EpiDynamics: Dynamic Models in Epidemiology*, 2015. URL <https://cran.r-project.org/package=EpiDynamics>. R package version 0.3.0. [p155]
- F. M. Bass. A new product growth for model consumer durables. *Management Science*, 15(5):215–227, 1969. URL <https://doi.org/10.1287/mnsc.15.5.215>. [p153]
- R. S. Bivand, E. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*. Springer-Verlag, New York, NY, 2013. [p158]
- M. Borgdorff, N. Nagelkerke, D. Van Soolingen, and J. Broekmans. Transmission of tuberculosis between people of different ages in the netherlands: An analysis using dna fingerprinting. *The International Journal of Tuberculosis and Lung Disease*, 3(3):202–206, 1999. [p156]
- D. Brockmann, L. Hufnagel, and T. Geisel. The scaling laws of human travel. *Nature*, 439(7075):462–465, 2006. [p157]
- J. Duggan. *System Dynamics Modeling with R*. Springer-Verlag, Switzerland, 2016. URL <https://doi.org/10.1007/978-3-319-34043-2>. [p154, 162]
- A. Ford and H. Flynn. Statistical screening of system dynamics models. *System Dynamics Review*, 21(4): 273–303, 2005. URL <https://doi.org/10.1002/sdr.322>. [p162]
- S. Jenness, S. M. Goodreau, M. Morris, E. Beylerian, S. Bender-deMoll, and K. Weiss. *EpiModel: Mathematical Modeling of Infectious Disease*, 2016. URL <https://cran.r-project.org/package=EpiModel>. R package version 1.2.8. [p155]
- M. J. Keeling and P. Rohani. *Modeling Infectious Diseases in Humans and Animals*. Princeton University Press, Princeton, New Jersey 08540, 2008. [p156, 162]
- D. C. Lane. IFORS operational research Hall of Fame Jay Wright Forrester. *International Transactions in Operational Research*, 13(5):483–492, 2006. [p154]
- X. Li and C. Prasad. Effectively teaching coding standards in programming. In *Proceedings of the 6th Conference on Information Technology Education*, pages 239–244. ACM, 2005. [p155]

- H. Liu, E. Howley, and J. Duggan. Co-evolutionary analysis: a policy exploration method for system dynamics models. *System Dynamics Review*, 28(4):361–369, 2012. URL <https://doi.org/10.1002/sdr.1482>. [p157]
- S. Meyer, L. Held, and others. Power-law models for infectious disease spread. *The Annals of Applied Statistics*, 8(3):1612–1639, 2014. URL <https://doi.org/10.1214/14-aos743>. [p157]
- J. Mossong, N. Hens, M. Jit, P. Beutels, K. Auranen, R. Mikolajczyk, M. Massari, S. Salmaso, G. S. Tomba, J. Wallinga, and others. Social contacts and mixing patterns relevant to the spread of infectious diseases. *PLoS Med*, 5(3):e74, 2008. URL <https://doi.org/10.1371/journal.pmed.0050074>. [p156]
- D. Mungovan, E. Howley, and J. Duggan. The influence of random interactions and decision heuristics on norm evolution in social networks. *Computational and Mathematical Organization Theory*, 17(2): 152–178, 2011. URL <https://doi.org/10.1007/s10588-011-9085-7>. [p157]
- J. A. Norton and F. M. Bass. A diffusion theory model of adoption and substitution for successive generations of high-technology products. *Management science*, 33(9):1069–1086, 1987. URL <https://doi.org/10.1287/mnsc.33.9.1069>. [p153]
- H. Rahmandad and J. Sterman. Heterogeneity and network structure in the dynamics of diffusion: Comparing agent-based and differential equation models. *Management Science*, 54(5):998–1014, 2008. [p153]
- K. Soetaert, T. Petzoldt, and R. W. Setzer. Solving Differential Equations in R: Package deSolve. *Journal of Statistical Software*, 33, 2010. [p155]
- K. Soetaert, T. Petzoldt, and R. W. Setzer. *deSolve: Solvers for Initial Value Problems of Differential Equations (ODE, DAE, DDE)*, 2016. URL <https://cran.r-project.org/package=deSolve>. R package version 1.14. [p155]
- J. D. Sterman. *Business Dynamics: Systems Thinking and Modeling for a Complex World*, volume 19. Irwin/McGraw-Hill Boston, New York, 2000. [p154, 155]
- E. Vynnycky and W. Edmunds. Analyses of the 1957 (Asian) Influenza Pandemic in the United Kingdom and the Impact of School Closures. *Epidemiology and infection*, 136(2):166–179, 2008. [p162]
- E. Vynnycky and R. White. *An Introduction to Infectious Disease Modelling*. OUP Oxford, 2010. [p153, 162]

Jim Duggan
School of Engineering and Informatics,
National University of Ireland Galway,
Galway,
Ireland.
jim.duggan@nuigalway.ie

MDplot: Visualise Molecular Dynamics

by Christian Margreitter and Chris Oostenbrink

Abstract The **MDplot** package provides plotting functions to allow for automated visualisation of molecular dynamics simulation output. It is especially useful in cases where the plot generation is rather tedious due to complex file formats or when a large number of plots are generated. The graphs that are supported range from those which are standard, such as RMSD/RMSF (root-mean-square deviation and root-mean-square fluctuation, respectively) to less standard, such as thermodynamic integration analysis and hydrogen bond monitoring over time. All told, they address many commonly used analyses. In this article, we set out the **MDplot** package's functions, give examples of the function calls, and show the associated plots. Plotting and data parsing is separated in all cases, i.e. the respective functions can be used independently. Thus, data manipulation and the integration of additional file formats is fairly easy. Currently, the loading functions support GROMOS, GROMACS, and AMBER file formats. Moreover, we also provide a Bash interface that allows simple embedding of MDplot into Bash scripts as the final analysis step.

Availability: The package can be obtained in the latest major version from CRAN (<https://cran.r-project.org/package=MDplot>) or in the most recent version from the project's GitHub page at <https://github.com/MDplot/MDplot>, where feedback is also most welcome. **MDplot** is published under the GPL-3 license.

Introduction

The amount of data produced by molecular dynamics (MD) engines (such as GROMOS (Schmid et al., 2012; Eichenberger et al., 2011), GROMACS (Pronk et al., 2013), NAMD (Phillips et al., 2005), AMBER (Cornell et al., 1995), and CHARMM (Brooks et al., 2009)) has been constantly increasing over recent years. This is mainly due to more powerful and cheaper hardware. As a result of this, both the lengths and sheer number of MD simulations (i.e. trajectories) have increased enormously. Even large sets of simulations (e.g., in the context of drug design) are attainable nowadays; thus suggesting that the processing of the resulting information is undertaken automatically.

In this respect, automated yet flexible visualisation of molecular dynamics data would be highly advantageous: both in order to avoid repetitive tasks for the user and to yield the ultimately desired result instantly (see Figure 1). Moreover, generating some of the graphs can be cumbersome. An example would be the plotting of a time series of a clustering program or hydrogen bonds. Therefore, these cases are predestined to be handled by a plotting library. There have been attempts made in that direction, for example the package **bio3d** (Grant et al., 2006; Skjærven et al., 2014) (which allows the trajectories to be processed in terms of principle component analysis (PCA), RMSD and RMSF calculations), **MDtraj** (McGibbon et al., 2015), or **Rknots** (Comoglio and Rinaldi, 2012). However, to the best of our knowledge, there is currently no R package available that offers the wide range of plotting functions and engine-support that is provided by **MDplot**. R is the natural choice for this undertaking because of both its power in data handling and its vast plotting abilities.

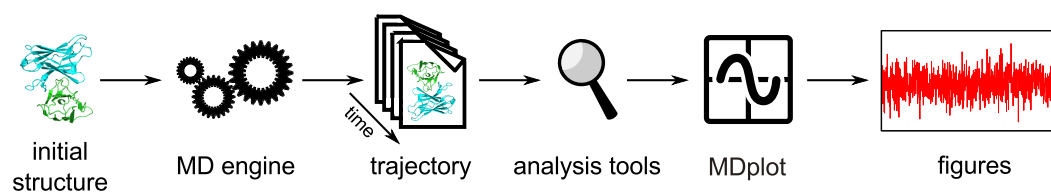


Figure 1: Shows the overall workflow typically applied in molecular dynamics simulations beginning with a single PDB (Berman et al., 2000) structure as the input for the simulation and ending with the graphical representation of the data obtained. For large amounts of data, generating figures might become a tedious, highly repetitive task.

In the following sections we outline all of the plotting functions that are currently supported. For each function, examples of the function calls based on the test data included in the package, the resulting plots, the return values, and a table of arguments are detailed. The respective code samples use the loading functions (reported below) to parse the input files located in folder 'extdata', which allows immediate testing and provides format information to users. Currently, the package supports

GROMOS, GROMACS, and AMBER file formats as input.¹ However, extensions in both format support and plotting functionalities are planned.

Plotting functions

The package currently offers 14 distinct plotting functions (Table ??), which cover many of the graphs that are commonly required. Although the focus of the package relies on the visualisation of data, in addition to this values are calculated to characterise the underlying data when appropriate. For example, `TIcurve()` calculates the thermodynamic integration free-energy values including error estimates and the hysteresis between the integration curves. In many cases, the plotting functions return useful information on the data used, e.g., range, mean and standard deviation of curves.

To provide simple access to these functions, they may be called from within a Bash script. Examples are provided at the end of the manuscript.

Plot function	Description
<code>clusters()</code>	Summary of clustering over trajectories (RMSD based).
<code>clusters_ts()</code>	Time series of cluster populations (RMSD based).
<code>dssp()</code>	Secondary structure annotation plot (DSSP based).
<code>dssp_ts()</code>	Time series of secondary structure elements (DSSP based).
<code>hbond()</code>	Hydrogen bonds summary plot.
<code>hbond_ts()</code>	Time series of hydrogen bonds.
<code>noe()</code>	Nuclear-Overhauser-effect violation plot.
<code>ramachandran()</code>	Dihedral angle plot.
<code>rmsd()</code>	Root-mean-square deviation plot.
<code>rmsd_average()</code>	Average root-mean-square deviation plot.
<code>rmsf()</code>	Root-mean-square fluctuation plot.
<code>TIcurve()</code>	Thermodynamic integration curves.
<code>timeseries()</code>	General time series plot.
<code>xrmsd()</code>	Cross-RMSD plot (heat-map of RMSD values).

Table 1: Lists all of the currently available plotting functions that have been implemented in **MDplot**. Most functions accept a boolean parameter (`barePlot`), that indicates printing of the plotting area only, i.e. stripped from any additional features such as axis labels.

The `clusters()` function

Molecular dynamics simulation trajectories can be considered to be a set of atom configurations along the time axis. Clustering is a method, that can be applied in order to extract common structural features from these. The configurations are classified and grouped together based on the root-mean-square deviation (RMSD). These subsets of configurations around the cluster's central member structure and their relative occurrences allow for comparisons between different and within individual simulations. `clusters()` allows to plot a summary of all of the (selected) clusters over a set of trajectories (Figure 2).

```
clusters(load_clusters("inst/extdata/clusters_example.txt.gz",
                    names=c("wild-type", "mut1", "mut2",
                          "mut3", "mut4", "mut5")),
        clustersNumber=9, main="MDplot::clusters()", ylab="# configurations")
```

Return value: Returns an $n \times m$ -matrix with n being the number of input trajectories and m the number of different clusters. Each element in the matrix holds the number of snapshots, in which the respective cluster occurred in the respective trajectory.

¹In this manuscript, the code samples use GROMOS input (since the default value of the loading functions' parameter `mdEngine` is "GROMOS"). For information on how to load GROMACS or AMBER files, please have a look at the manual pages of the respective loading functions.

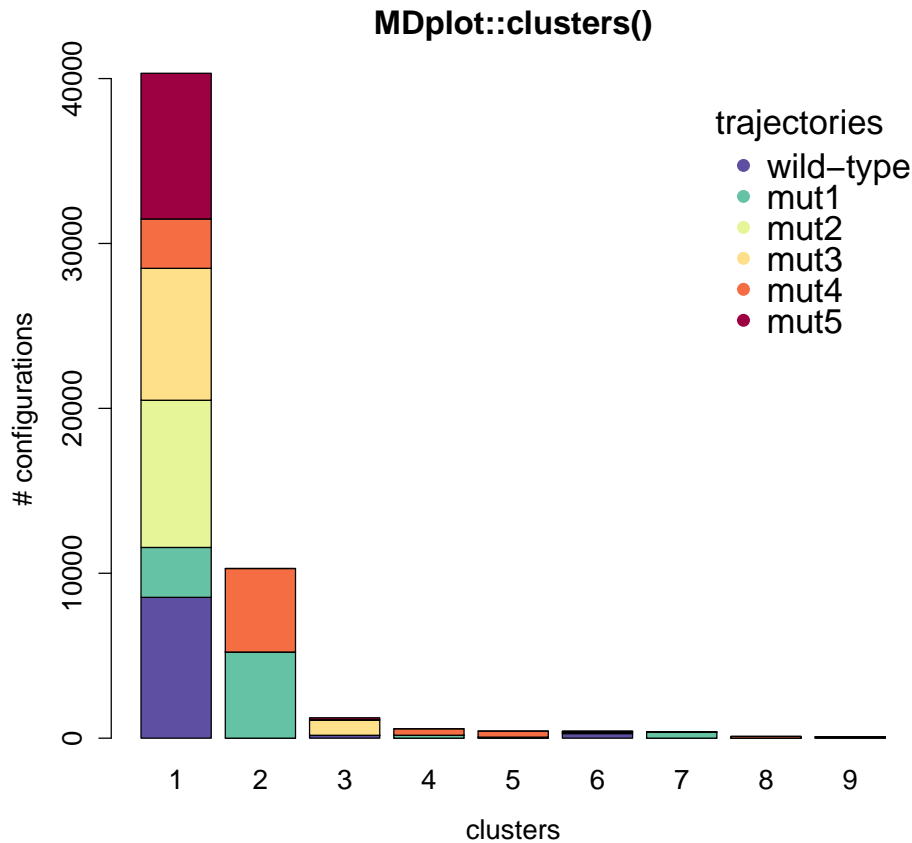


Figure 2: The clusters are plotted along the x -axis and the number of configurations for each trajectory for every cluster on the y -axis. The number of clusters is limited in this example to nine with the `clustersNumber` argument, which can be useful to omit scarcely populated clusters.

Argument name	Default value	Description
<code>clusters</code>	<i>none</i>	Matrix with clusters: trajectories are given in row-wise, clusters in column-wise fashion as provided by <code>load_clusters()</code> , the associated loading function.
<code>clustersNumber</code>	NA	When specified, only these first clusters are shown.
<code>legendTitle</code>	"trajectories"	The title of the legend.
<code>barePlot</code>	FALSE	A Boolean indicating whether the plot is to be made without any additional information or not.
...	<i>none</i>	Additional arguments.

Table 2: Arguments of the `clusters()` function.

The `clusters_ts()` function

In structural clustering, it is often instructive to have a look at the development over time rather than the overall summary. This functionality is provided by `clusters_ts()`. In the top sub-plot the overall distribution is given, while the time series is shown at the bottom. The clusters are sorted beginning with the most populated one, in descending order. Selections can be made and clusters that are not selected do also not appear in the time series plot (white areas). The time axis may be shown in nanoseconds (see Figure 3 for an example).

```
clusters_ts(load_clusters_ts("inst/extdata/clusters_ts_example.txt.gz",
                           lengths=c(4000, 4000, 4000, 4000, 4000, 4000)),
```

```
names=c("wild-type", "mut1", "mut2",
        "mut3", "mut4", "mut5"),
clustersNumber=7, main="MDplot::clusters_ts() example",
timeUnit="ns", snapshotsPerTimeInterval=100)
```

Return value: Returns a summary $(n + 1) \times m$ -matrix with n being the number of input trajectories and m the number of different clusters (which have been plotted). Each element in the matrix holds the number of snapshots, in which the respective cluster occurred in the respective trajectory. In addition, the first line is the overall summary counted over all trajectories.

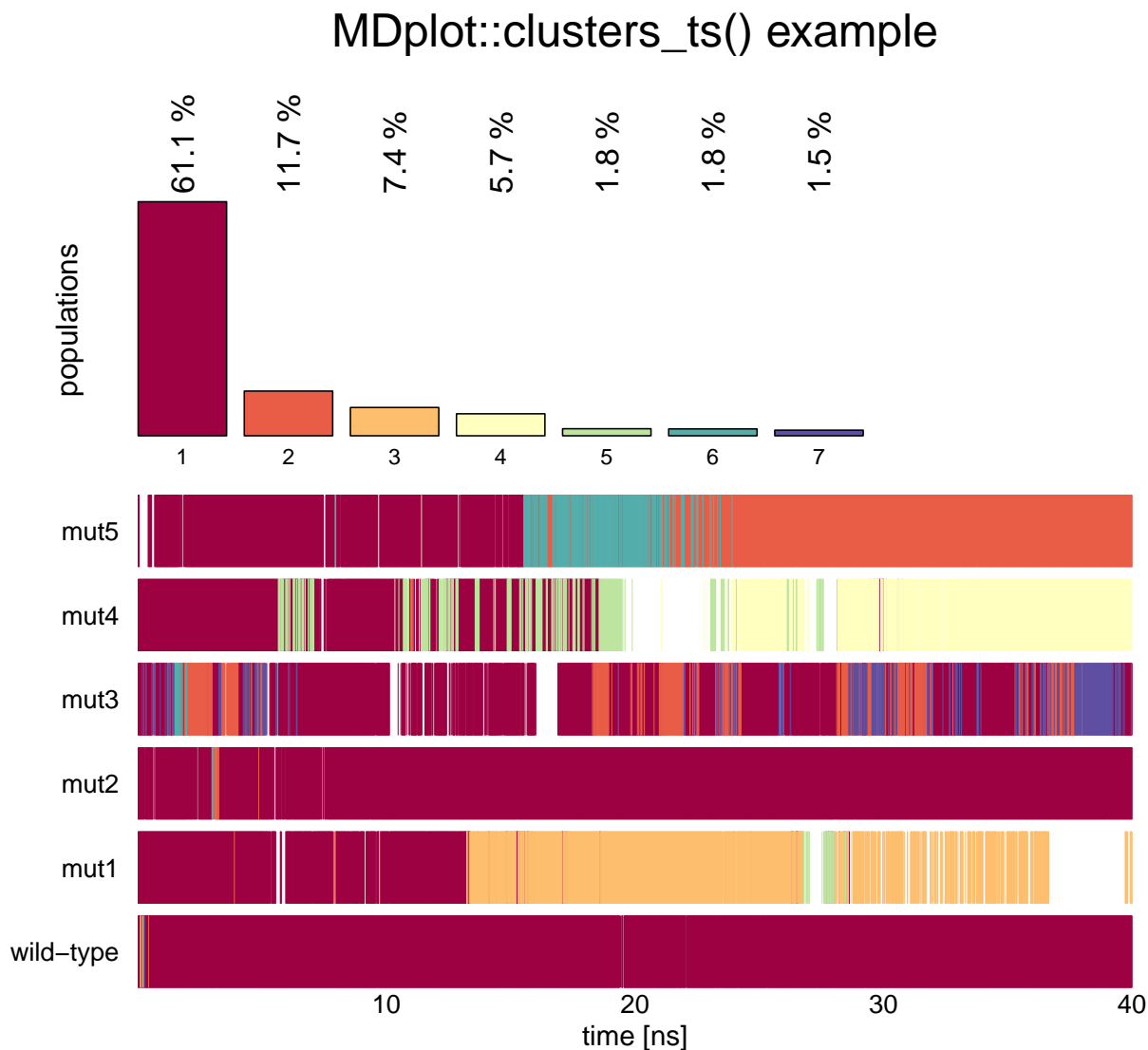


Figure 3: The plot shows a selection of the seven most populated clusters of six trajectories. Regions that do not belong to one of the first seven clusters are shown in white.

Argument name	Default value	Description
clustersDataTS	<i>none</i>	List of cluster information as provided by <code>load_clusters_ts()</code> , the associated loading function.
clustersNumber	NA	An integer specifying the number of clusters that is to be plotted.
selectTraj	NA	Vector of indices of trajectories that are plotted (as given in the input file).
selectTime	NA	Range of time in snapshots.
timeUnit	NA	Abbreviation of time unit.
snapshotsPerTimeInt	1000	Number of snapshots per time unit.
...	<i>none</i>	Additional arguments.

Table 3: Arguments of the `clusters_ts()` function.

The `dssp()` function

In terms of proteins the secondary structure can be annotated by the widely used program DSSP (Definition of Secondary Structure of Proteins) (Kabsch and Sander, 1983). This algorithm uses the backbone hydrogen bond pattern in order to assign secondary structure elements such as α -helices, β -strands, and turns to protein sequences. The plotting function `dssp()` has three different visualisation methods and plots the overall result over the trajectory and over the residues. The user can specify selections of residues and which elements should be taken into consideration (Figure 4).

```
layout(matrix(1:3, nrow=1), widths=c(0.33,0.33,0.33))
dssp(load_dssp("inst/extdata/dssp_example.txt.gz"),
     main="plotType=dots", showResidues=c(1,35))
dssp(load_dssp("inst/extdata/dssp_example.txt.gz"),
     main="plotType=curves", plotType="curves", showResidues=c(1,35))
dssp(load_dssp("inst/extdata/dssp_example.txt.gz"),
     main="plotType=bars", plotType="bars", showResidues=c(1,35))
```

Return value: Returns a matrix, where the first column is the residue-number and the remaining ones denote secondary structure classes. Residues are given row-wise and values range from 0 to 100 percent.

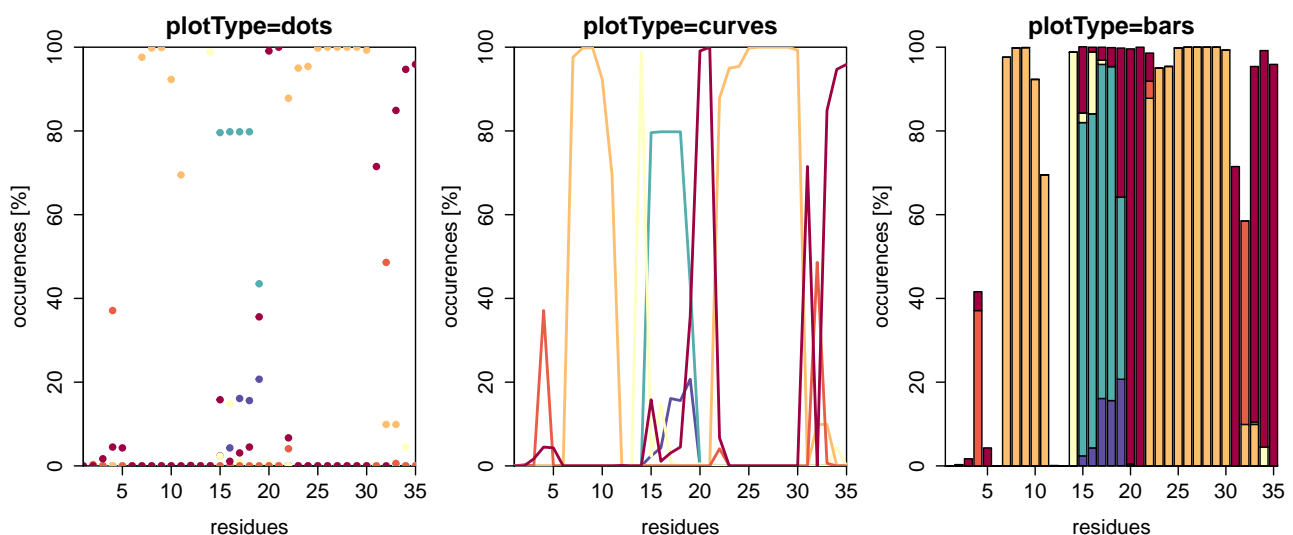


Figure 4: Example of `dssp()` with `plotType` set to "dots" (default), "curves" or "bars". Note that the fractions do not necessarily sum up to a hundred percent, because some residues might not be in defined secondary structure elements all the time. In this figure, there is no legend plotted due to space limitations (see Figure 5 for a colour-code explanation).

Argument name	Default value	Description
dsspData	<i>none</i>	Table containing information on the secondary structure elements. Can be generated by function <code>load_dssp()</code> .
printLegend	FALSE	If TRUE, a legend is printed on the right hand side of the plot.
useOwnLegend	FALSE	If FALSE, the names of the secondary structure elements are considered to be in default order.
elementNames	NA	Vector of names for the secondary structure elements.
colours	NA	A vector of colours that can be specified to replace the default ones.
showValues	NA	A vector of boundaries for the values.
showResidues	NA	A vector of boundaries for the residues.
plotType	"dots"	Either "dots", "curves", or "bars".
selectedElements	NA	A vector of names of the elements selected.
barePlot	FALSE	Boolean, indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 4: Arguments of the `dssp()` function.

The `dssp_ts()` function

The secondary structure information as described for the function `dssp()` can also be visualised along the time axis using function `dssp_ts()` (Figure 5). The time can be annotated in snapshots or time units (e.g., nanoseconds).

```
dssp_ts(load_dssp_ts("inst/extdata/dssp_ts_example"),printLegend=TRUE,
        main="MDplot::dssp_ts()",timeUnit="ns",
        snapshotsPerTime=1000)
```

Argument name	Default value	Description
tsData	<i>none</i>	List of lists, which are composed of a name (string) and a values table (x ... snapshots, y ... residues). Can be generated by <code>load_dssp_ts()</code> .
printLegend	TRUE	If TRUE, a legend is printed on the right hand side of the plot.
timeBoundaries	NA	A vector of boundaries for the time in snapshots.
residueBoundaries	NA	A vector of boundaries for the residues.
timeUnit	NA	If set, the snapshots are transformed into the respective time (depending on parameter <code>snapshotsPerTime</code>).
snapshotsPerTimeInt	1000	Number of snapshots per respective <code>timeUnit</code> .
barePlot	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 5: Arguments of the `dssp_ts()` function.

The `hbond()` function

In the context of biomolecules, hydrogen bonds are of particular importance. These bonds take place between a donor, a hydrogen, and an acceptor atom. This function plots the summary output of

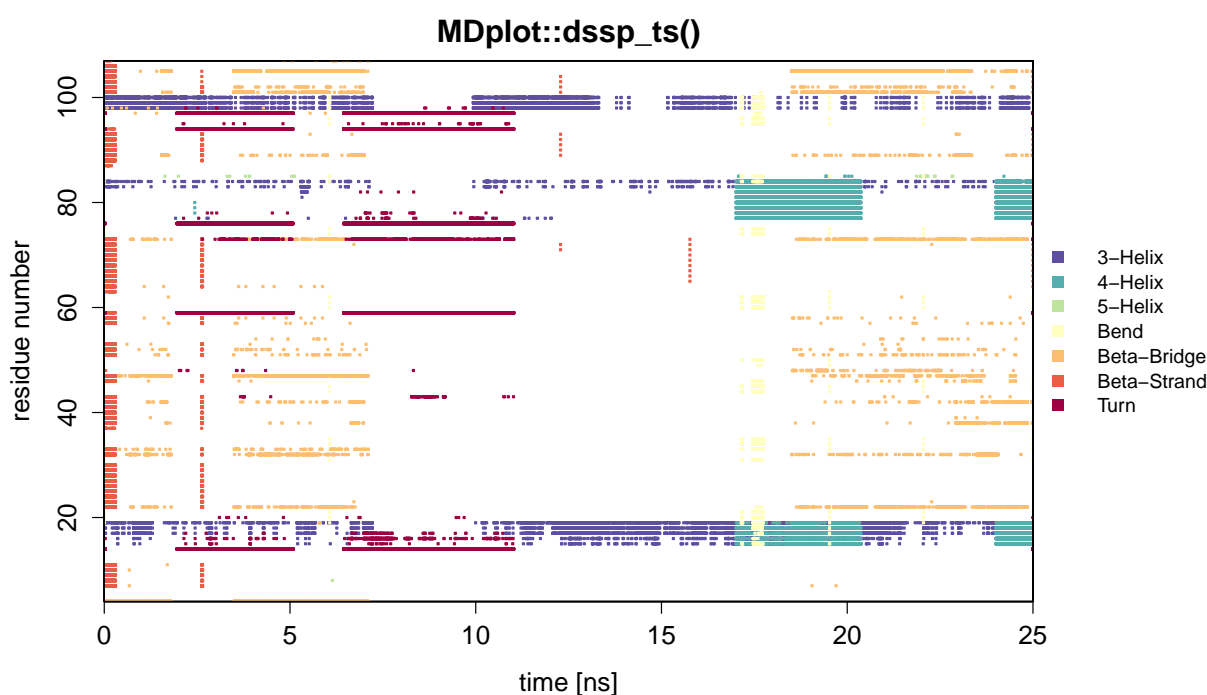


Figure 5: Example showing all of the defined secondary structure elements per residue over time. Note, that for this example plot a sparse data set was used to reduce the size of the data file (hence the large white areas in the middle).

hydrogen bond calculations and allows selection of donor and acceptor residues. Occurrence over the whole trajectory is indicated by a colour scale. Note, that in case multiple hydrogen bond interactions between two particular residues take place (conveyed by different sets of atoms), the interaction with prevalence will be used for colour-coding (and by default, this interaction is marked with a black circle, see below). An example is given in Figure 6.

```
hbond(load_hbond("inst/extdata/hbond_example.txt.gz"),
      main="MDplot::hbond()", donorRange=c(0,65))
```

Return value: Returns a table containing the information used for plotting in columns as follows:

- `resDonor` Residue number (donor).
- `resAcceptor` Residue number (acceptor).
- `percentage` Percentage, that has been used for colour-coding.
- `numberInteractions` Number of hydrogen bond interactions taking place between the specified donor and acceptor residues.

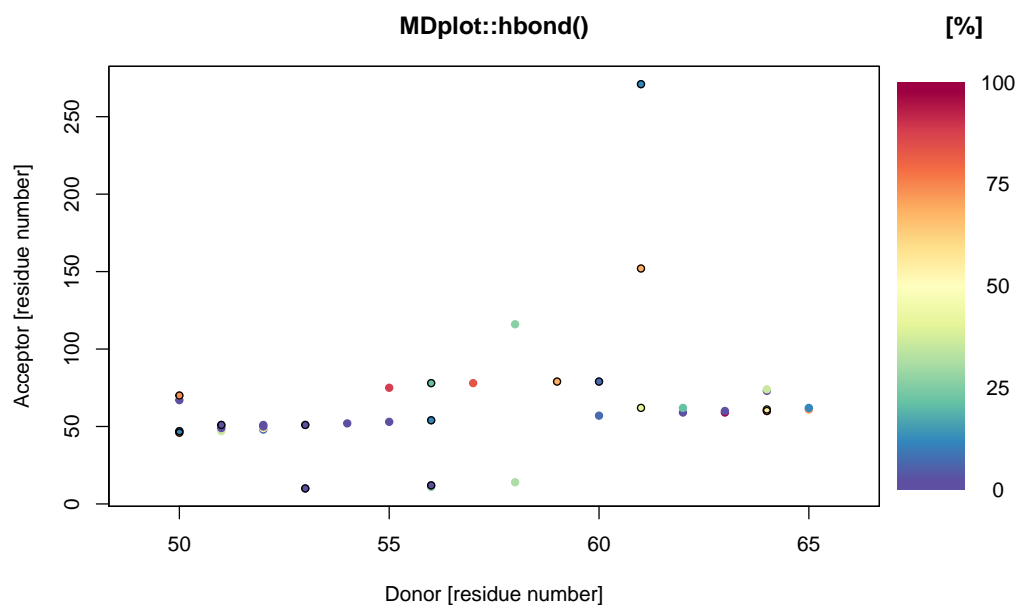


Figure 6: The acceptor residues are plotted on the x -axis whilst the donors are shown on the y -axis. The different colours indicate the occurrences throughout the whole trajectory.

Argument name	Default value	Description
hbonds	<i>none</i>	Table containing the hydrogen bond information in columns "hbondID", "resDonor", "resDonorName", "resAcceptor", "resAcceptorName", "atomDonor", "atomDonorName", "atomH", "atomAcceptor", "atomAcceptorName", "percentage" (automatically generated by function <code>load_hbond()</code>).
plotMethod	"residue-wise"	Allows to set the detail of hydrogen bond information displayed. Options are: "residue-wise".
acceptorRange	NA	A vector specifying the range of acceptor residues.
donorRange	NA	A vector specifying the range of donor residues.
printLegend	TRUE	A Boolean enabling the legend.
showMultipleInteractions	TRUE	If TRUE, this option causes multiple interactions between the same residues as being represented by a black circle around the coloured dot.
barePlot	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 6: Arguments of the `hbond()` function.

The `hbond_ts()` function

The time series of hydrogen bond occurrences can be visualised using the function `hbond_ts()`, which plots them either according to their identifiers or in a human readable form in three- or one-letter code (the participating atoms can be shown as well) on the y -axis and the time on the x -axis. If the GROMOS input format is used, this function requires two different files: the summary of the `hbond` program and the time series file. The occurrence of a hydrogen bond is represented by a black bar and the occurrence summary can be added on the right hand side as a sub-plot (Figure 7). In addition to the time series file, depending on the MD engine format used, an additional summary file might also be necessary (see the documentation of the function `load_hbond_ts()` for further information).

```
hbond_ts(timeseries=load_hbond_ts("inst/extdata/hbond_ts_example.txt.gz"),
        summary=load_hbond("inst/extdata/hbond_example.txt.gz"),
        main="MDplot::hbond_ts()", acceptorRange=c(22, 75),
        hbondIndices=list(c(0, 24)), plotOccurrences=TRUE, timeUnit="ns",
        snapshotsPerTimeInt=100, printNames=TRUE, namesToSingle=TRUE,
        printAtoms=TRUE)
```

Return value: Returns an $n \times 2$ -matrix, with the first column being the list of hydrogen bond identifiers plotted and the second one the occurrence (in percent) over the selected time range.

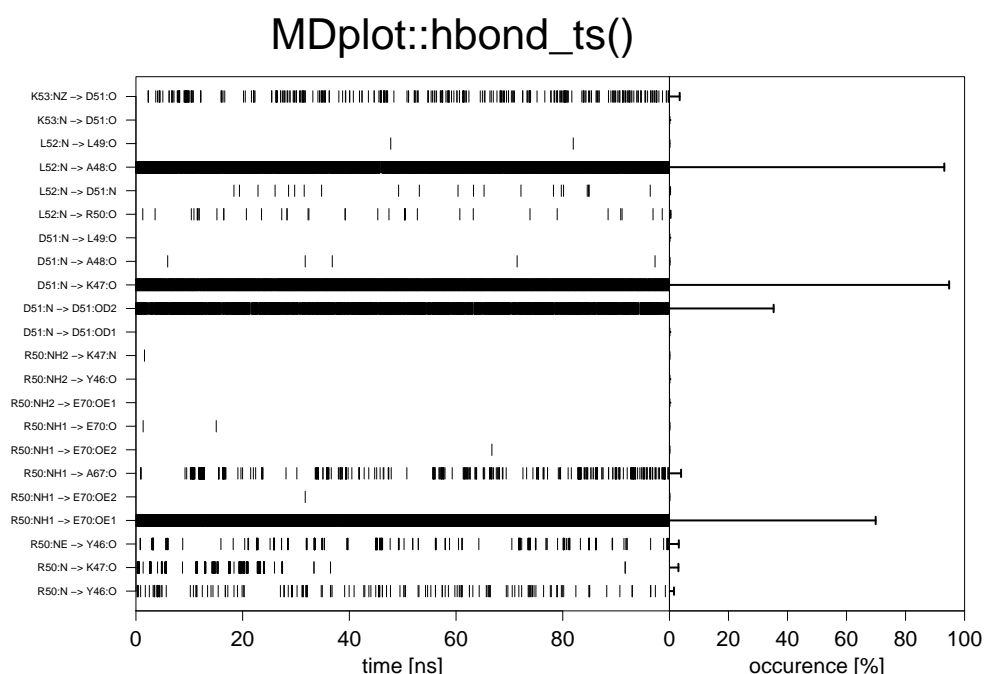


Figure 7: Example figure generated by `hbond_ts()` for both an identifier and acceptor residues' selection. The labels for the hydrogen bonds may be printed as identifiers or with names composed of residue names (in single- or three-letter code) and those of the participating atoms.

Argument name	Default value	Description
timeseries	<i>none</i>	Table containing the time series information (e.g., produced by <code>load_hbond_ts()</code>).
summary	<i>none</i>	Table containing the summary information (e.g., produced by <code>load_hbond()</code>).
acceptorRange	NA	A vector of acceptor residues.
donorRange	NA	A vector of donor residues.
plotOccurrences	FALSE	Specifies whether the overall summary should be plotted on the right hand side.
scalingFactorPlot	NA	Used to manually set the scaling factor (if necessary).
printNames	FALSE	Enables human readable names rather than the hydrogen bond identifiers.
namesToSingle	FALSE	If <code>printNames</code> is TRUE, this flag instructs one-letter codes instead of three-letter ones.
printAtoms	FALSE	Enables atom names in hydrogen bond identification on the y -axis.
timeUnit	NA	Specifies the time unit on the x -axis.
snapshotsPerTimeInt	1000	Specifies how many snapshots make up one time unit (see above).
timeRange	NA	A vector specifying a certain time range.
hbondIndices	NA	A list containing vectors to select hydrogen bonds by their identifiers.
barePlot	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 7: Arguments of the `hbond_ts()` function.

The `noe()` function

The nuclear-Overhauser-effect is one of the most important measures of structure validity in the context of molecular dynamics simulations. These interactions are transmitted through space and arise from spin-spin coupling, which can be measured by nuclear magnetic resonance (NMR) spectroscopy. These measurements provide pivotal distance restraints which should be matched on average during molecular dynamics simulations of the same system and can hence be used for parameter validation. The plotting function `noe()` allows to visualise the number of distance restraint violations and their respective spatial deviation. As shown in Figure 8, multiple replicates or different protein systems are supported simultaneously. Note that negative violations are not considered.

```
noe(load_noe(files=c("inst/extdata/noe_example_1.txt.gz",
                    "inst/extdata/noe_example_2.txt.gz")),
    main="MDplot::noe()")
```

Return value: Returns a matrix, in which the first column holds the bin boundaries used and the following columns represent either the percentage or absolute numbers of the violations per bin, depending on the specification.

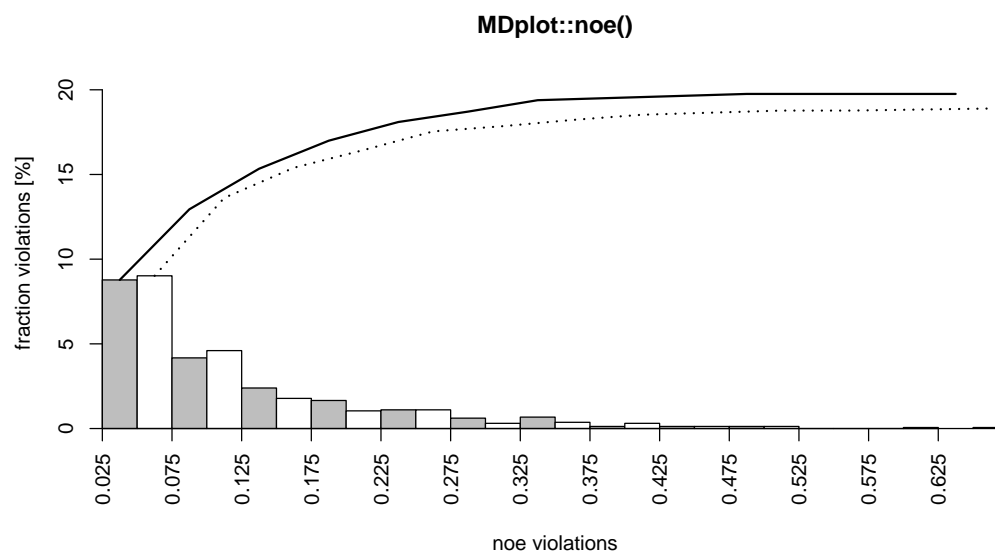


Figure 8: Example plot showing two different replicates of a protein simulation (they share the same molecule, but have different initial velocities). Note, that the maximum value (x -axis) over all replicates is used for the plot. The sum over all violations from left to right is shown by an additional curve on top. The number of violations may be given as fractions (in %), as shown above, or absolute numbers (flag `printPercentages` either `TRUE` or `FALSE`).

Argument name	Default value	Description
<code>noeData</code>	<i>none</i>	Input matrix. Generated by function <code>load_noe()</code> .
<code>printPercentages</code>	<code>TRUE</code>	If <code>TRUE</code> , the violations will be reported in a relative manner (percent) rather than absolute numbers.
<code>colours</code>	<code>NA</code>	Vector of colours to be used for the bars.
<code>lineTypes</code>	<code>NA</code>	If <code>plotSumCurves</code> is <code>TRUE</code> , this vector might be used to specify the types of curves plotted.
<code>names</code>	<code>NA</code>	Vector to name the input columns (legend).
<code>plotSumCurves</code>	<code>TRUE</code>	If <code>TRUE</code> , the violations are summed up from left to right to show the overall behaviour.
<code>maxYAxis</code>	<code>NA</code>	Can be used to manually set the y -axis of the plot.
<code>printLegend</code>	<code>FALSE</code>	A Boolean indicating if legend is to be plotted.
...	<i>none</i>	Additional arguments.

Table 8: Arguments of the `noe()` function.

The `ramachandran()` function

This graph type (Ramachandran et al., 1963) is often used to show the sampling of the ϕ/ψ protein backbone dihedral angles in order to assign propensities of secondary structure elements to the protein of interest (so-called Ramachandran plots). These plots can provide crucial insight into energy barriers arising as required, for example, in the context of parameter validation (Margreitter and Oostenbrink, 2016). The function `ramachandran()` offers a 2D (Figure 9) and 3D (Figure 10) variant with the former offering the possibility to print user-defined secondary structure regions as well. The number of bins for the two axes and the colours used for the legend can be specified by the user.

```
ramachandran(load_ramachandran("inst/extdata/ramachandran_example.txt.gz"),
             heatFun="log", plotType="sparse", xBins=90, yBins=90,
             main="ramachandran() (plotType=sparse)",
             plotContour=TRUE)
ramachandran(load_ramachandran("inst/extdata/ramachandran_example.txt.gz"),
             heatFun="norm", plotType="fancy", xBins=90, yBins=90,
```

```
main="ramachandran() (plotType=fancy)",
printLegend=TRUE)
```

Return value: Returns a list of binned dihedral angle occurrences.

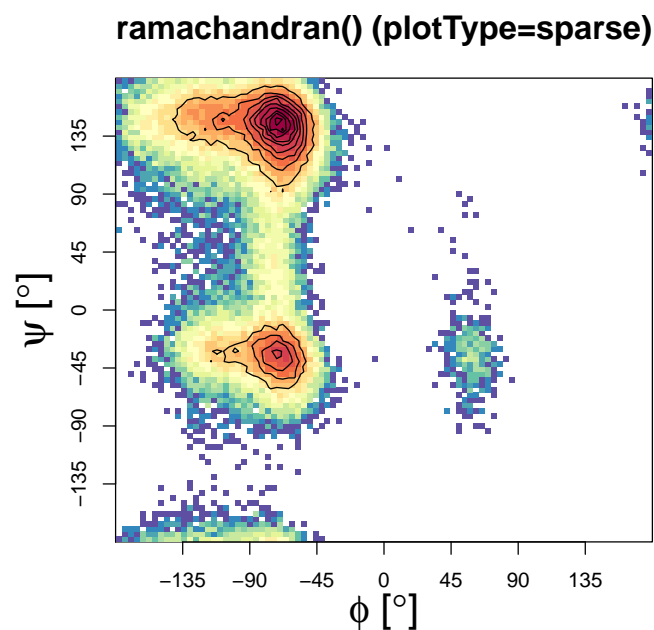


Figure 9: Two-dimensional plot version "sparse" of the ramachandran() function with enabled contour plotting. The number of bins can be specified for both dimensions independently.

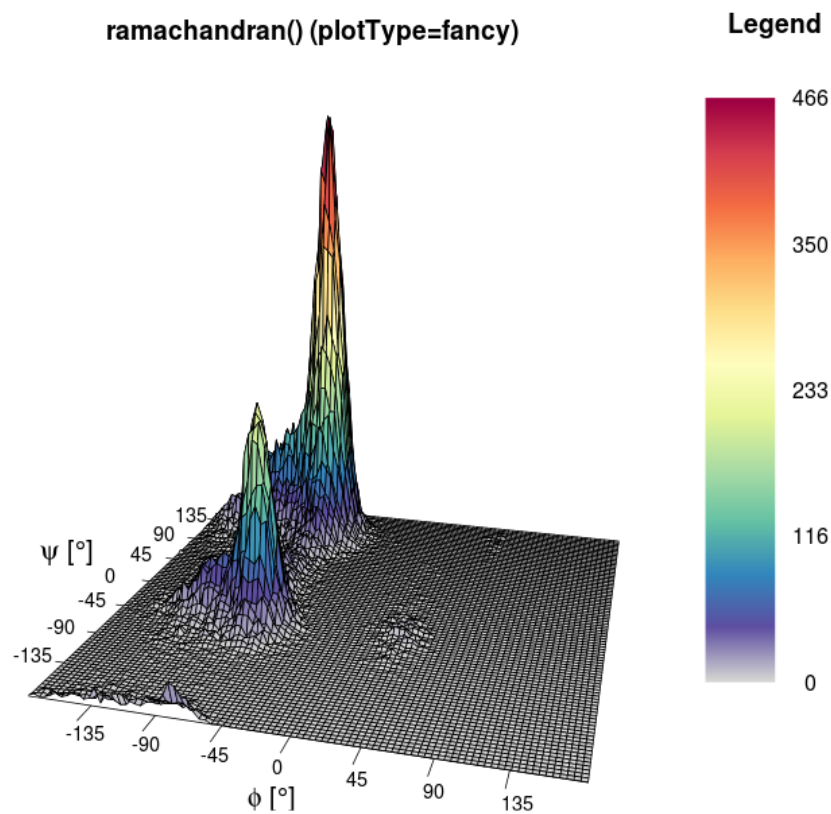


Figure 10: Three-dimensional example of the ramachandran() function. In addition to the colour, the height (z-axis) also represents the number of dihedrals per bin.

Argument name	Default value	Description
dihedrals	<i>none</i>	Matrix with angles (two columns). Generated by function <code>load_ramachandran()</code> .
xBins	150	Number of bins used to plot (<i>x</i> -axis).
yBins	150	Number of bins used to plot (<i>y</i> -axis).
heatFun	"norm"	Function selector for calculation of the colour. The possibilities are either: "norm" for linear calculation or "log" for logarithmic calculation.
structureAreas	<code>c()</code>	List of areas, which are plotted as black lines.
plotType	"sparse"	Type of plot to be used, either "sparse" (default, using function <code>hist2d()</code>), "comic" (own binning, supports very few datapoints), or "fancy" (3D, using function <code>persp()</code>).
printLegend	FALSE	A Boolean specifying whether a heat legend is to be plotted or not.
plotContour	FALSE	A Boolean specifying whether a contour should be added or not.
barePlot	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 9: Arguments of the `ramachandran()` function.

The `rmsd()` function

The atom-positional root-mean-square deviation (RMSD) is one of the most commonly used plot types in the field of biophysical simulations. In the context of atom configurations, it is a measure for the positional divergence of one or multiple atoms. The input requires a list of alternating vectors of time indices and RMSD values. Multiple data sets can be plotted, given in separate input files. Figure 11 shows an example for two trajectories.

```
rmsd(load_rmsd(c("inst/extdata/rmsd_example_1.txt.gz",
               "inst/extdata/rmsd_example_2.txt.gz")),
      printLegend=TRUE, names=c("WT", "mut"), main="MDplot::rmsd()")
```

Return value: Returns a list of lists, where each sub-list represents a RMSD curve and contains the components:

- `minValue` The minimum value over the whole time range.
- `maxValue` The maximum value over the whole time range.
- `meanValue` The mean value calculated over the whole time range.
- `sd` The standard deviation calculated over the whole time range.

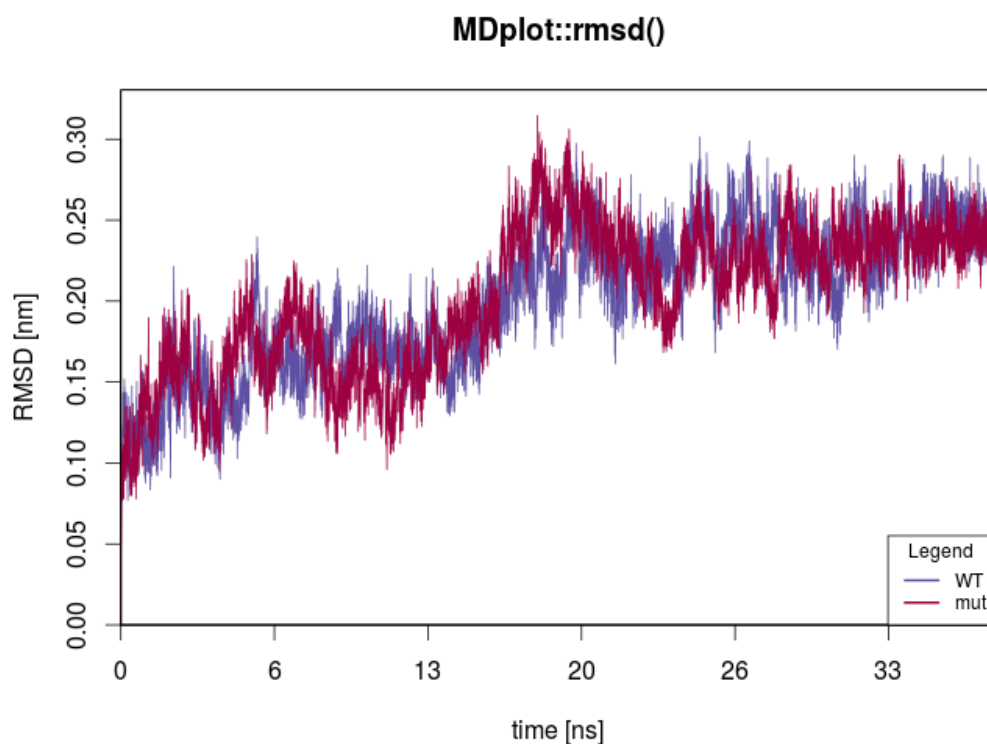


Figure 11: This plot shows the RMSD curves for two different trajectories. The time is given in nanoseconds, which requires a properly set factor parameter.

Argument name	Default value	Description
rmsdData	<i>none</i>	List of (alternating) indices and RMSD value vectors, as produced by <code>load_rmsd()</code> .
printLegend	TRUE	A Boolean which triggers the plotting of the legend.
factor	1000	A number specifying how many snapshots are within one timeUnit.
timeUnit	"ns"	Specifies the time unit.
rmsdUnit	"nm"	Specifies the RMSD unit.
colours	NA	A vector of colours used for plotting.
names	NA	A vector holding the names of the trajectories.
legendPosition	"bottomright"	Indicates the position of the legend: either "bottomright", "bottomleft", "topleft", or "topright".
barePlot	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 10: Arguments of the `rmsd()` function.

The `rmsd_average()` function

Nowadays, for many molecular systems multiple replicates of simulations are performed in order to enhance the sampling of the phase space. However, since the amount of analysis data grows accordingly, a joint representation of the results may be desirable. For the case of backbone-atom and other RMSD plots, the **MDplot** package supports average plotting. Instead of plotting every curve individually, the mean and the minimum and maximum values of all trajectories at a given time point is plotted. Thus, the spread of multiple simulations is represented as a 'corridor' over time.

```
rmsd_average(rmsdInput=list(load_rmsd("inst/extdata/rmsd_example_1.txt.gz" ),
                             load_rmsd("inst/extdata/rmsd_example_2.txt.gz")),
            maxYAxis=0.375,main="MDplot::rmsd_average()")
```

Return value: Returns an $n \times 4$ -matrix, with the rows representing different snapshots and the columns the respective values as follows:

- `snapshot` Index of the snapshot.
- `minimum` The minimum RMSD value over all input sources at a given time.
- `mean` The mean RMSD value over all input sources at a given time.
- `maximum` The maximum RMSD value over all input sources at a given time.

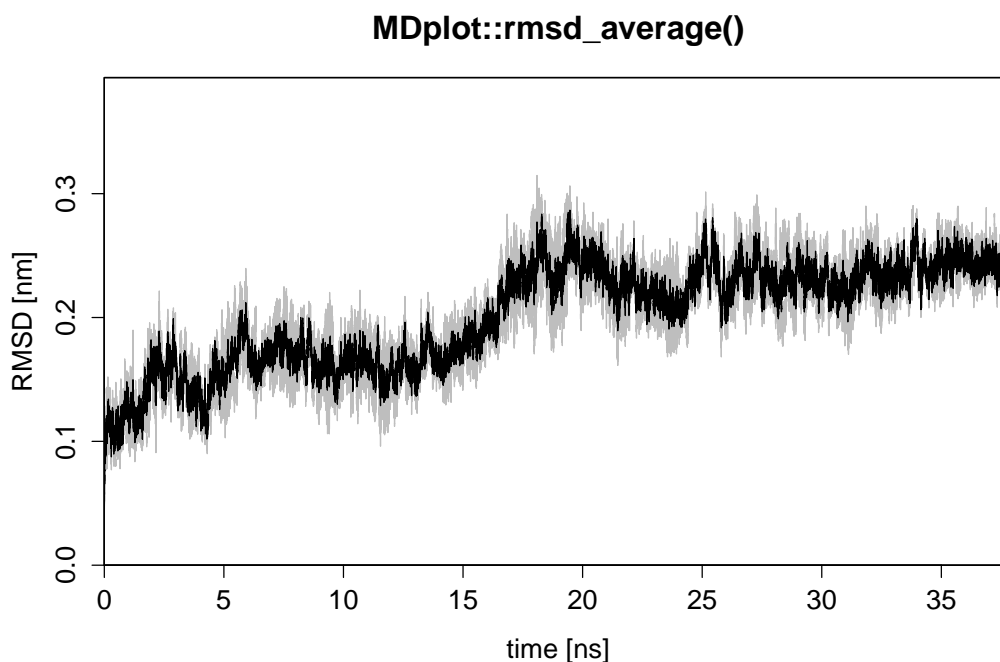


Figure 12: In black, the mean RMSD value at a given timepoint and in grey the respective minimum and maximum values are given. In this example, two rather similar curves have been used.

Argument name	Default value	Description
<code>rmsdInput</code>	<i>none</i>	List of snapshot and RMSD value pairs, as, for example, provided by loading function <code>load_rmsd()</code> .
<code>levelFactor</code>	NA	If there are many datapoints, this parameter may be used to use only the <code>levelFactor</code> th datapoints to obtain a clean graph.
<code>snapshotsPerTimeInt</code>	1000	Number, specifying how many snapshots are comprising one <code>timeUnit</code> .
<code>timeUnit</code>	"ns"	Specifies the time unit.
<code>rmsdUnit</code>	"nm"	Specifies the RMSD unit.
<code>maxYAxis</code>	NA	Can be used to manually set the y -axis of the plot.
<code>barePlot</code>	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 11: Arguments of the `rmsd_average()` function.

The `rmsf()` function

The atom-positional root-mean-square fluctuation (RMSF) represents the degree of positional variation of a given atom over time. The input requires one column with all residues or atoms and a second one holding RMSF values. Figure 13 shows, as an example, the RMSF of the first 75 atoms, calculated for two independent simulations.

```
rmsf(load_rmsf(c("inst/extdata/rmsf_example_1.txt.gz",
               "inst/extdata/rmsf_example_2.txt.gz")),
     printLegend=TRUE, names=c("WT", "mut"), range=c(1, 75),
     main="MDplot::rmsf()")
```

Return value: A list of vectors, alternately holding atom indices and their respective values.

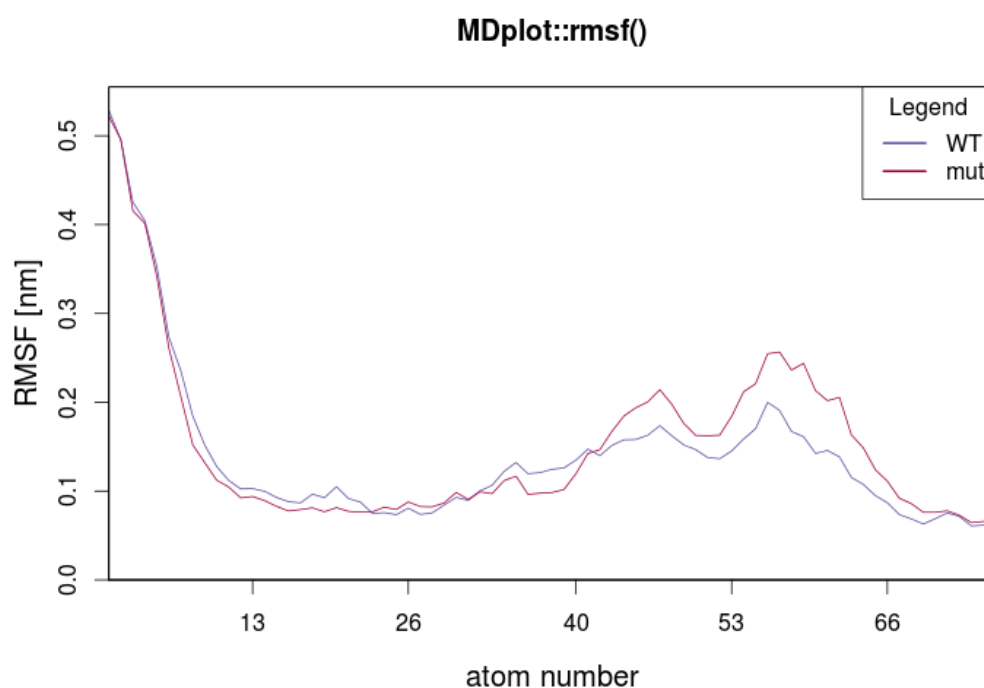


Figure 13: Plot showing two different RMSF curves.

Argument name	Default value	Description
rmsfData	<i>none</i>	List of (alternating) atom numbers and RMSF values, as, for example, produced by <code>load_rmsf()</code> .
printLegend	TRUE	A Boolean controlling the plotting of the legend.
rmsfUnit	"nm"	Specifies the RMSF unit.
colours	NA	A vector of colours used for plot.
residuewise	FALSE	A Boolean specifying whether atoms or residues are plotted on the x -axis.
atomsPerResidue	NA	If <code>residuewise</code> is TRUE, this parameter can be used to specify the number of atoms per residue for plotting.
names	NA	A vector of the names of the trajectories.
range	NA	Range of atoms.
legendPosition	"topright"	Indicates position of legend: either "bottomright", "bottomleft", "topleft", or "topright".
barePlot	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 12: Arguments of the `rmsf()` function.

The `TIcurve()` function

For calculations of the free energy difference occurring when transforming one chemical compound into another (alchemical changes) or for estimates of free energy changes upon binding, thermodynamic integration (Kirkwood, 1935) is one of the most trusted and applied approaches. The derivative of the Hamiltonian, as a function of a coupling parameter λ , is calculated over a series of λ state points (typically around 15). The integral of this curve is equivalent to the change in free energy (Figure 14). The function `TIcurve()` performs the integration and, if the data for both the forward and backward processes are provided, the hysteresis between them.

```
TIcurve(load_TIcurve(c("inst/extdata/TIcurve_fb_forward_example.txt.gz",
                     "inst/extdata/TIcurve_fb_backward_example.txt.gz")),
        invertedBackwards=TRUE, main="MDplot:TIcurve()")
```

Return value: Returns a list with the following components:

- `lambdapoints` A list containing a (at least) $n \times 3$ -matrix for every data input series.
- `integrationresults` A matrix containing one row of "deltaG" and "error" columns from the integration for every data input series.
- `hysteresis` If two (i.e. forward and backward) data input series are provided, the resulting hysteresis is reported (and set to be NA otherwise).

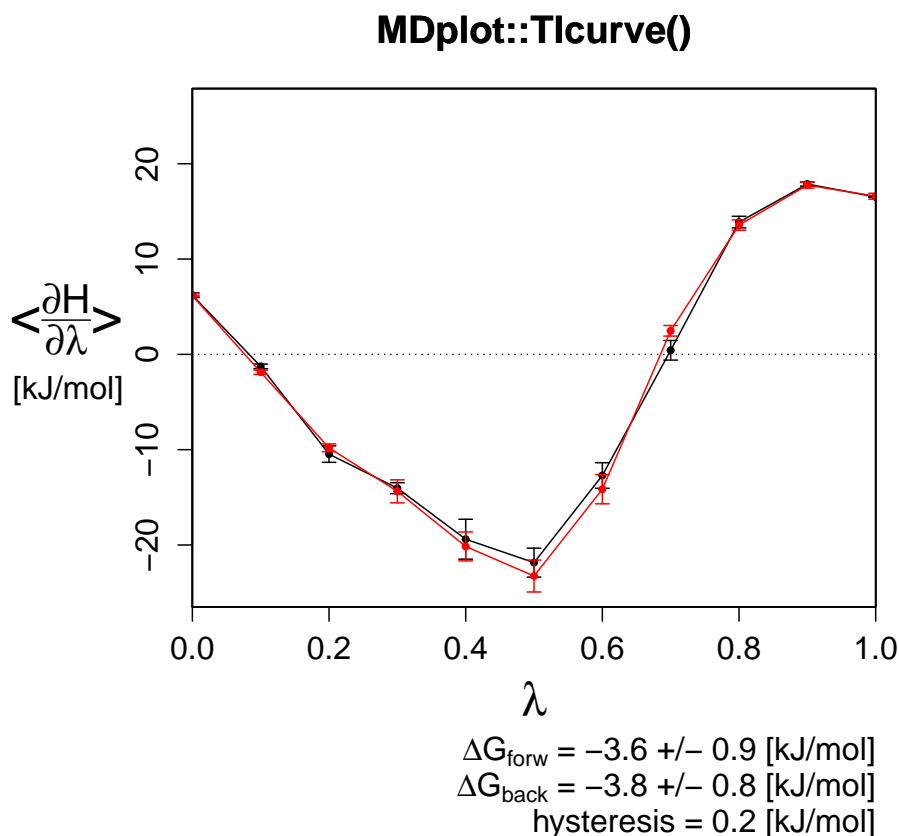


Figure 14: A forward and backward thermodynamic integration curve with the resulting hysteresis between them (precision as permitted by the error).

Argument name	Default value	Description
lambdas	<i>none</i>	List of matrices (automatically generated by <code>load_Tlcurve()</code>) holding the thermodynamic integration information.
invertedBackwards	FALSE	If a forward and backward TI are provided and the lambda points are enumerated reversely (i.e. 0.3 of one TI is equivalent to 0.7 of the other), this flag can be set to be TRUE in order to automatically mirror the values appropriately.
energyUnit	"kJ/mol"	Defines the energy unit used for the plot.
printValues	TRUE	If TRUE, the free energy values are printed.
printErrors	TRUE	A Boolean indicating whether error bars are to be plotted.
errorBarThreshold	0	If the error at a given lambda point is below this threshold, it is not plotted.
barePlot	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 13: Arguments of the `Tlcurve()` function.

The `timeseries()` function

This function provides a general interface for any time series given as a time-value pair (Figure 15).

```
timeseries(load_timeseries(c("inst/extdata/timeseries_example_1.txt.gz",
                             "inst/extdata/timeseries_example_2.txt.gz"))),
```

```
main="MDplot::timeseries()",
names=c("fluc1", "fluc2"),
snapshotsPerTimeInt=100)
```

Return value: Returns a list of lists, each of the latter holding for every data input series:

- `minValue` The minimum value over the whole set.
- `maxValue` The maximum value over the whole set.
- `meanValue` The mean value over the whole set.
- `sd` The standard deviation over the whole set.

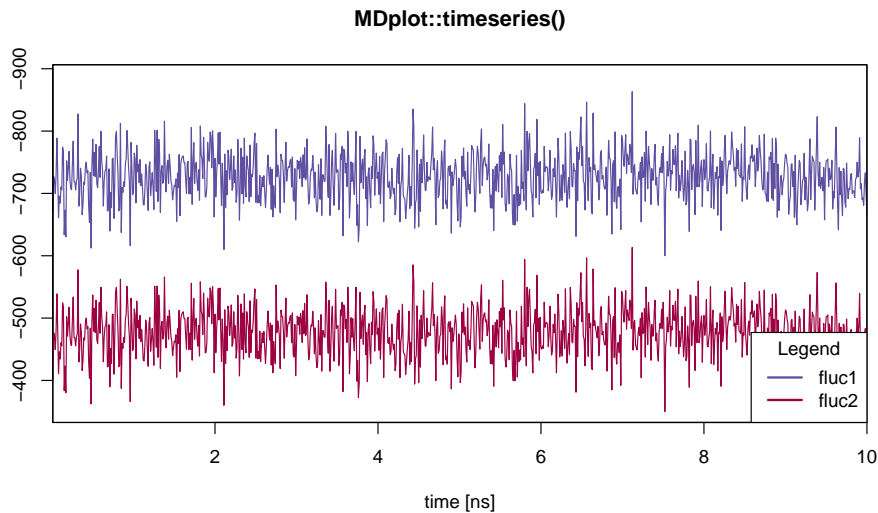


Figure 15: Shows time series with parameter `snapshotsPerTimeInt` set in a way such, that the proper time in nanoseconds is plotted. In addition, the legend has been moved to the bottom-right position.

Argument name	Default value	Description
<code>tsData</code>	<i>none</i>	List of (alternating) indices and response values, as produced by <code>load_timeseries()</code> .
<code>printLegend</code>	TRUE	Parameter enabling the plotting of the legend.
<code>snapshotsPerTimeInt</code>	1000	Number specifying how many snapshots make up one <code>timeUnit</code> .
<code>timeUnit</code>	"ns"	Specifies the time unit.
<code>valueName</code>	NA	Name of response variable.
<code>valueUnit</code>	NA	Specifies the response variable's unit.
<code>colours</code>	NA	A vector of colours used for plotting.
<code>names</code>	NA	A vector of names of the trajectories.
<code>legendPosition</code>	"bottomright"	Indicates position of legend: either "bottomright", "bottomleft", "topleft", or "topright".
<code>barePlot</code>	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 14: Arguments of the `timeseries()` function.

The `xrmsd()` function

This function generates a plot which shows a heat-map of the atom positional root-mean-square differences between snapshots (figure 16). The structures are listed on the x - and y -axes. The heat-map shows the difference between one structure and another using a coloured bin. The legend is adapted in accordance to the size of the values.

```
xrmsd(load_xrmsd("inst/extdata/xrmsd_example.txt.gz"),
      printLegend=TRUE,main="MDplot::xrmsd()")
```

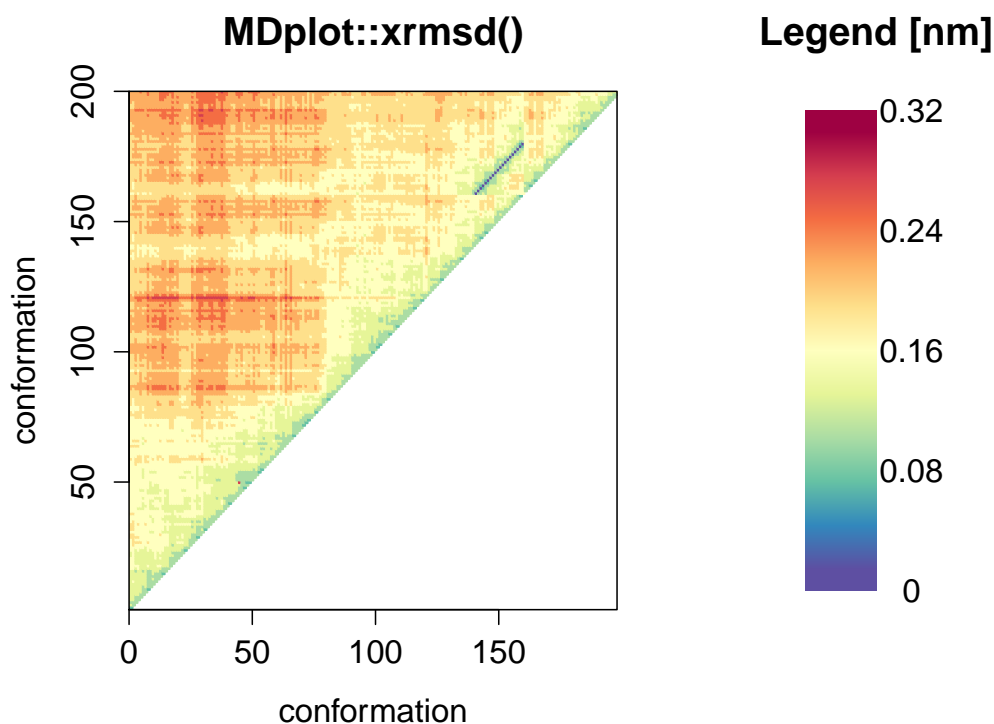


Figure 16: An example `xrmsd()` plot showing only the upper half because of the mirroring of the values.

Argument name	Default value	Description
<code>xrmsdValues</code>	<i>none</i>	Input matrix (three rows: x -values, y -values, RMSD-values). Can be generated by function <code>load_xrmsd()</code> .
<code>printLegend</code>	TRUE	If TRUE, a legend is printed on the right hand side.
<code>xaxisRange</code>	NA	A vector of boundaries for the x -snapshots.
<code>yaxisRange</code>	NA	A vector of boundaries for the y -snapshots.
<code>colours</code>	NA	User-specified vector of colours to be used for plotting.
<code>rmsdUnit</code>	"nm"	Specifies in which unit the RMSD values are given.
<code>barPlot</code>	FALSE	A Boolean indicating whether the plot is to be made without any additional information.
...	<i>none</i>	Additional arguments.

Table 15: Arguments of the `xrmsd()` function.

Additional functions and the Bash interface

Given that the plotting functions expect input to be stored in a defined data structure, the step of loading and parsing data from the text input files has been implemented in separate loading functions. Currently, they support GROMOS, GROMACS, and AMBER file formats and further developments are planned to cover additional ones as well.

In order to allow for direct calls from Bash scripts, users might use the Rscript interface located in the folder 'bash' which serves as a wrapper shell. Pictures in the file formats PNG, TIFF, or PDF can be used provided that the users' R installation supports them. If `help=TRUE` is set, all the other options are ignored and a full list of options for every command is printed. In general, the names of the arguments of the functions are the same for calls by script. The syntax for these calls is `Rscript MDplot_bash.R {function name} [argument1=...] [argument2=...]`, which can be combined with Bash variables (see below). The file path can be given in an absolute manner or relative to the Rscript folder path. The package holds a file called 'bash/test.sh' which contains several examples.

```
#!/bin/bash
# clusters
Rscript MDplot_bash.R clusters files=../extdata/clusters_example.txt.gz \
    title="Cluster analysis" size=900,900 \
    outformat=tiff outfile=clusters.tiff \
    clustersNumber=7 \
    names=WT,varA,varB,varC2,varD3,varE4

# xrmsd
Rscript MDplot_bash.R xrmsd files=../extdata/xrmsd_example.txt.gz title="XRMSD" \
    size=1100,900 outformat=pdf outfile=XRMSD.pdf \
    xaxisRange=75,145

# ramachandran
Rscript MDplot_bash.R ramachandran files=../extdata/ramachandran_example.txt.gz \
    title="Ramachandran plot" size=1400,1400 resolution=175 \
    outformat=tiff outfile=ramachandran.tiff angleColumns=1,2 \
    bins=75,75 heatFun=norm printLegend=TRUE plotType=fancy
```

The loading functions

In order to ease data preparation, loading functions have been devised which are currently able to load the output of standard GROMOS, GROMACS, and AMBER analysis tools and store these data such, that they can be interpreted by the **MDplot** plotting functions.² Loading functions are named after their associated plotting function with 'load_' as prefix. For other molecular dynamics engines than the aforementioned ones, the user has to specify how their output should be read. However, in case other file formats are requested we appreciate suggestions, requests, and contributions (to be made on our GitHub page). For detailed descriptions of the data structures used, we refer to the manual pages of the loading functions and the respective examples. For storage reasons the example input files have been compressed using gzip with R being able to load both compressed and uncompressed files.

Conclusions

In this paper we have presented the package **MDplot** and described its application in the context of molecular dynamics simulation analysis. Automated figure generation is likely to aid in the understanding of results at the first glance and may be used in presentations and publications. Planned extensions include both the integration of new functionalities such as a DISICL (secondary structure classification (Nagy and Oostenbrink, 2014a,b)) as well as the provision of loading interfaces for additional molecular dynamics engines. Further developments will be published on the projects' GitHub page and on CRAN.

²Functions `load_timeseries()` and `load_TIcurve()` do not require engine-specific loading and function `noe()` is only available for GROMOS because no input files for the other engines could be retrieved.

Acknowledgements

The authors would like to thank Prof. Friedrich Leisch for his useful comments and guidance in the package development process, Markus Fleck for providing GROMACS analysis files, Silvia Bonomo for her help in dealing with AMBER, Sophie Krecht for her assistance in typesetting, and Jamie McDonald for critical reading of the manuscript.

Funding

This work was supported by the European Research Council (ERC; grant number 260408), the Austrian Science Fund (FWF; grant number P 25056) and the Vienna Science and Technology Fund (WWTF; grant number LS08-QM03).

Bibliography

- H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235, 2000. [p164]
- B. R. Brooks, C. L. Brooks, A. D. MacKerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus. CHARMM: The Biomolecular Simulation Program. *Journal of Computational Chemistry*, 30(10):1545–1614, 2009. ISSN 0192-8651. URL <https://doi.org/10.1002/jcc.21287>. [p164]
- F. Comoglio and M. Rinaldi. Rknots: Topological analysis of knotted biopolymers with R. *Bioinformatics*, 28(10):1400, 2012. [p164]
- W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman. A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules. *Journal of the American Chemical Society*, 117(19):5179–5197, 1995. ISSN 0002-7863. URL <https://doi.org/10.1021/ja00124a002>. [p164]
- A. P. Eichenberger, J. R. Allison, J. Dolenc, D. P. Geerke, B. A. C. Horta, K. Meier, C. Oostenbrink, N. Schmid, D. Steiner, D. Wang, and W. F. van Gunsteren. GROMOS++ Software for the Analysis of Biomolecular Simulation Trajectories. *Journal of Chemical Theory and Computation*, 7(10):3379–3390, 2011. ISSN 1549-9618. URL <https://doi.org/10.1021/ct2003622>. [p164]
- B. J. Grant, A. P. C. Rodrigues, K. M. ElSawy, J. A. McCammon, and L. S. D. Caves. Bio3d: An R package for the comparative analysis of protein structures. *Bioinformatics*, 22(21):2695–2696, 2006. ISSN 1367-4803, 1460-2059. [p164]
- W. Kabsch and C. Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983. ISSN 1097-0282. URL <https://doi.org/10.1002/bip.360221211>. [p168]
- J. G. Kirkwood. Statistical mechanics of fluid mixtures. *Journal of Chemical Physics*, pages 300–313, 1935. [p180]
- C. Margreitter and C. Oostenbrink. Optimization of Protein Backbone Dihedral Angles by Means of Hamiltonian Reweighting. *Journal of Chemical Information and Modeling*, 56(9):1823–1834, 2016. URL <https://doi.org/10.1021/acs.jcim.6b00399>. [p174]
- R. T. McGibbon, K. A. Beauchamp, M. P. Harrigan, C. Klein, J. M. Swails, C. X. Hernández, C. R. Schwantes, L.-P. Wang, T. J. Lane, and V. S. Pande. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophysical Journal*, 109(8):1528 – 1532, 2015. URL <https://doi.org/10.1016/j.bpj.2015.08.015>. [p164]
- G. Nagy and C. Oostenbrink. Dihedral-Based Segment Identification and Classification of Biopolymers II: Polynucleotides. *Journal of Chemical Information and Modeling*, 54(1):278–288, 2014a. URL <https://doi.org/10.1021/ci400542n>. [p184]
- G. Nagy and C. Oostenbrink. Dihedral-based segment identification and classification of biopolymers I: Proteins. *Journal of Chemical Information and Modeling*, 54(1):266–277, 2014b. ISSN 1549-960X. URL <https://doi.org/10.1021/ci400541d>. [p184]

- J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005. ISSN 1096-987X. URL <https://doi.org/10.1002/jcc.20289>. [p164]
- S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl. GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics*, 29(7):845–854, 2013. ISSN 1367-4803, 1460-2059. [p164]
- G. N. Ramachandran, C. Ramakrishnan, and V. Sasisekharan. Stereochemistry of polypeptide chain configurations. *Journal of Molecular Biology*, 7:95–99, 1963. ISSN 0022-2836. [p174]
- N. Schmid, C. D. Christ, M. Christen, A. P. Eichenberger, and W. F. van Gunsteren. Architecture, implementation and parallelisation of the GROMOS software for biomolecular simulation. *Computer Physics Communications*, 183(4):890–903, 2012. ISSN 0010-4655. URL <https://doi.org/10.1016/j.cpc.2011.12.014>. [p164]
- L. Skjærven, X.-Q. Yao, G. Scarabelli, and B. J. Grant. Integrating protein structural dynamics and evolutionary analysis with Bio3D. *BMC Bioinformatics*, 15(1), 2014. URL <https://doi.org/10.1186/s12859-014-0399-6>. [p164]

Christian Margreitter
Institute of Molecular Modeling and Simulation
University of Natural Resources and Life Sciences (BOKU)
Austria
christian.margreitter@gmail.com

Chris Oostenbrink
Institute of Molecular Modeling and Simulation
University of Natural Resources and Life Sciences (BOKU)
Austria
chris.oostenbrink@boku.ac.at

On Some Extensions to GA Package: Hybrid Optimisation, Parallelisation and Islands Evolution

by Luca Scrucca

Abstract Genetic algorithms are stochastic iterative algorithms in which a population of individuals evolve by emulating the process of biological evolution and natural selection. The R package **GA** provides a collection of general purpose functions for optimisation using genetic algorithms. This paper describes some enhancements recently introduced in version 3 of the package. In particular, hybrid GAs have been implemented by including the option to perform local searches during the evolution. This allows to combine the power of genetic algorithms with the speed of a local optimiser. Another major improvement is the provision of facilities for parallel computing. Parallelisation has been implemented using both the master-slave approach and the islands evolution model. Several examples of usage are presented, with both real-world data examples and benchmark functions, showing that often high-quality solutions can be obtained more efficiently.

Introduction

Optimisation problems of both practical and theoretical importance deal with the search of an optimal configuration for a set of variables to achieve some specified goals. Potential solutions may be encoded with real-valued, discrete, binary or permutation decision variables depending on the problem to be solved. Direct search or derivative-free methods, gradient-based and Newton-type methods, all encompass traditional local optimisation algorithms for real-valued functions (Chong and Zak, 2013; Givens and Hoeting, 2013, Chap. 2). In contrast, discrete and combinatorial optimisation problems involve decision variables expressed using integers or binary values and consist in searching for the best solution from a set of discrete elements (Papadimitriou and Steiglitz, 1998; Givens and Hoeting, 2013, Chap. 3).

A large number of heuristics and metaheuristics algorithms have been proposed for solving complex optimisation tasks. Specific (ad-hoc) heuristic techniques are able to identify solutions in a reasonably short amount of time, but the solutions obtained are generally not guaranteed to be optimal or accurate. On the contrary, metaheuristics offer a tradeoff between exact and heuristics methods, in the sense that they are generic techniques that offer good solutions, often the global optimum value sought, in a moderate execution time by efficiently and effectively exploring the search space (Luke, 2013). This class of algorithms typically implements some form of stochastic optimisation and includes: Evolutionary Algorithm (EA; Back et al., 2000a,b), Iterated Local Search (ILS; Lourenço et al., 2003), Simulated Annealing (SA; Kirkpatrick et al., 1983), Differential Evolution (DE; Storn and Price, 1997), Particle Swarm Optimisation (PSO; Kennedy and Eberhart, 1995), Tabu Search (TS; Glover and Laguna, 2013), Ant Colony Optimisation (ACO; Dorigo and Stützle, 2004), and Covariance Matrix Adaptation Evolution Strategy (CMA-ES; Hansen, 2006).

EAs are stochastic iterative algorithms in which a population of individuals evolve by emulating natural selection (Eiben and Smith, 2003; De Jong, 2006; Simon, 2013). Each individual of the population represents a tentative solution to the problem. The quality of the proposed solution is expressed by the value of a fitness function assigned to each individual. This value is then used by EAs to guide the search and improve the fitness of the population. Compared to other metaheuristics algorithms, EAs are able to balance between exploration of new areas of the search space and exploitation of good solutions. The trade-off between exploration and exploitation is controlled by some tuning parameters, such as the population size, the genetics operators (i.e. selection, crossover, and mutation), and the probability of applying them. Genetic Algorithms (GAs) are search and optimisation procedures that are motivated by the principles of natural genetics and natural selection. GAs are the "earliest, most well-known, and most widely-used EAs" (Simon, 2013, p. 35).

R offers several tools for solving optimisation problems. A comprehensive listing of available packages is contained in the CRAN task view on "*Optimization and Mathematical Programming*" (Theussl and Borchers, 2015). An extensive treatment of optimisation techniques applied to problems that arise in statistics and how to solve them using R is provided by Nash (2014). A gentle introduction to metaheuristics optimisation methods in R is contained in Cortez (2014). Some R packages implementing evolutionary optimisation algorithms are: **rgenoud**, **Rmalschains**, **DEoptim**, **GenSA**, **ps**, **cmaes**, **tabuSearch**.

The R package **GA** is a flexible general-purpose set of tools for optimisation using genetic algo-

gorithms and it is fully described in [Scrucca \(2013\)](#). Real-valued, integer, binary and permutation GAs are implemented, whether constrained or not. Discrete or combinatorial optimisation problems, where the search space is made of a finite or countably infinite set of potential solutions, can be easily treated by adopting a binary or permutation representation. Several genetic operators for selection, crossover, and mutation are available, and more can be defined by experienced R users.

This paper describes some recent additions to the **GA** package. The first improvement involves the option to use hybrid GAs. Although for many objective functions GAs are able to work in an efficient way and find the area of the global optimum, they are not especially fast at finding the optimum when in a locally quadratic region. Hybrid GAs combine the power of GAs with the speed of a local optimiser, allowing researchers to find a global solution more efficiently than with the conventional evolutionary algorithms. Because GAs can be easily and conveniently executed in parallel machines, the second area of improvement is that associated with parallel computing. Two approaches, the master-slave and islands models, have been implemented and are fully described. Several examples, using both real-world data examples and benchmark functions, are presented and discussed.

GA package

In the following we assume that the reader has already installed the latest version (≥ 3.0) of the package from CRAN with

```
> install.packages("GA")
```

and the package is loaded into an R session using the usual command

```
> library(GA)
```

Hybrid genetic algorithms

EAs are very good at identifying near-optimal regions of the search space (*exploration*), but they can take a relatively long time to locate the exact local optimum in the region of interest (*exploitation*). Traditionally, exploitation is done through selection, whilst exploration is performed by search operators, such as mutation and crossover ([Eiben and Schippers, 1998](#)). However, exploitation can also be pursued by controlling crossover and mutation, e.g. by reducing the mutation probability as the search progresses. Balancing between exploration and exploitation is vital for successful application of EAs ([Črepinšek et al., 2013](#)).

A further possibility for improving exploitation is to try to incorporate efficient local search algorithms into EAs. There are different ways in which local searches or problem-specific information can be integrated in EAs (see [Eiben and Smith, 2003](#), Chap. 10). For instance, a local search may be started from the best solution found by a GA after a certain number of iterations, so that, once a promising region is identified, the convergence to the global optimum can be sped up. These evolutionary methods have been named in various ways, such as *hybrid GAs*, *memetic GAs*, and *genetic local search algorithms*. Some have argued that the inclusion of a local search in GAs implies the use of a form of Lamarckian evolution. This fact has been criticised from a biological point of view, but "despite the theoretical objections, hybrid genetic algorithms typically do well at optimization tasks" ([Whitley, 1994](#), p. 82).

In case of real-valued optimisation problems, the **GA** package provides a simple to use implementation of hybrid GAs by setting the argument `optim = TRUE` in a `ga()` function call. This allows to perform local searches using the base R function `optim()`, which makes available general-purpose optimisation methods, such as Nelder–Mead, quasi-Newton with and without box constraints, and conjugate-gradient algorithms.

Having set `optim = TRUE`, the local search method to be used and other parameters can be controlled with the optional argument `optimArgs`. This must be a list with the following structure and defaults:

```
optimArgs = list(method = "L-BFGS-B",  
                 poptim = 0.05,  
                 pressel = 0.5,  
                 control = list(fnscale = -1, maxit = 100))
```

where

method	The method to be used among those available in <code>optim</code> function. By default, the BFGS algorithm with box constraints is used, where the bounds are those provided in the <code>ga()</code> function call. Further methods are available as described in the Details section in <code>help(optim)</code> .
poptim	A value in the range $(0, 1)$ which gives the the probability of applying the local search at each iteration.
pressel	A value in the range $(0, 1)$ which specifies the pressure selection.
control	A list of parameters for fine tuning the <code>optim</code> algorithm. See <code>help(optim)</code> for details.

In the implementation available in **GA**, the local search is applied stochastically during the GA iterations with probability $\text{poptim} \in [0, 1]$; by default, once every $1/0.05 = 20$ iterations on average. The local search algorithm is started from a random selected solution drawn with probability proportional to fitness and with the selection process controlled by the parameter $\text{pressel} \in [0, 1]$. Let f_i be the fitness value associated with the i th solution for $i = 1, \dots, n$, where n is the `popSize`, and let r_i be the corresponding rank in non increasing order. Then, for a given `pressel` the probability of selection is computed as $p_i = \text{pressel} \times (1 - \text{pressel})^{r_i - 1}$, and then normalised as $p_i = p_i / \sum_{i=1}^n p_i$. Figure 1 shows the probability of selection as a function of the fitness value for different levels of selection pressure. The values on the y -axis are computed using the function `optimProbSel()`, which is used in the **GA** package for computing the probability of selection for each individual of the genetic population. When the pressure is set at 0, the same probability of selection is assigned to all solutions. Larger probabilities are assigned to larger f_i values as the pressure value increases. In the extreme case of pressure selection equal to 1, only the largest f_i receives a probability of selection equal to 1, whereas the others have no chance of being selected. Thus, smaller values of `pressel` tend to assign equal probabilities to all the solutions, and larger values tend to assign larger values to those solutions having better fitness.

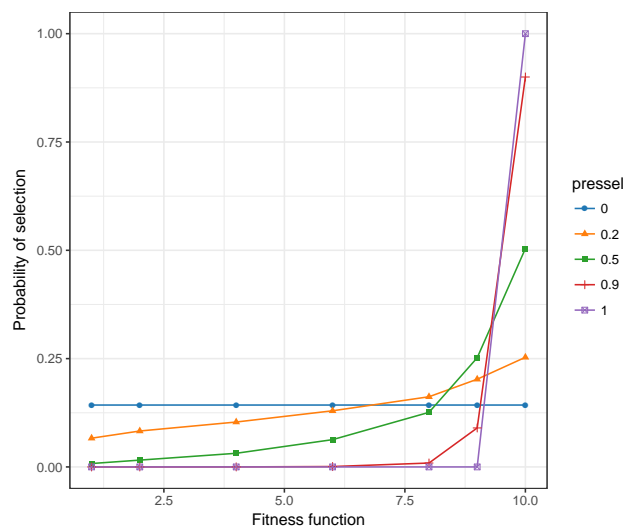


Figure 1: Graph of the probability of selecting a solutions for starting a local search in HGA as a function of the fitness and for different levels of selection pressure.

Note that when a `ga()` function call is issued with `optim = TRUE`, a local search is always applied at the end of GA evolution, i.e. after the last iteration and even in case of `poptim = 0`, but starting from the solution with the highest fitness value. The rationale for this is to allow for local optimisation as a final improvement step.

Portfolio selection

In portfolio selection the goal is to find the optimal portfolio, i.e. the portfolio that provides the highest return and lowest risk. This is achieved by choosing the optimal set of proportions of various financial assets (Ruppert and Matteson, 2015, Chap. 16). In this section an example of mean–variance efficient portfolio selection (Gilli et al., 2011, Chap. 13) is illustrated.

Suppose we have selected 10 stocks from which to build a portfolio. We want to determine how

much of each stock to include in our portfolio. The *expected return rate* of our portfolio is

$$E(R) = \sum_{i=1}^{10} w_i E(R_i),$$

where $E(R_i)$ is the expected return rate on asset i , and w_i is the fraction of the portfolio value due to asset i . Note that the portfolio weights w_i must satisfy the constraints $w_i \geq 0$, and $\sum_{i=1}^{10} w_i = 1$. At the same time, we want to minimise the *variance of portfolio returns* given by

$$\sigma_p^2 = w' \Sigma w,$$

where Σ is the covariance matrix of stocks returns, and $w' = (w_1, \dots, w_{10})$, under the constraint that the portfolio must have a minimum expected return of 1%, i.e. $E(R) \geq 0.01$. Provided that only linear constraints are included, the problem of mean-variance portfolio selection is typically solved by quadratic programming. However, GAs provide a general approach to portfolio selection that can be used for problems with both linear and nonlinear constraints (Gilli and Schumann, 2012).

Consider the following stocks with monthly return rates obtained by Yahoo finance using the `quantmod` package:

```
> library(quantmod)
> myStocks <- c("AAPL", "XOM", "GOOGL", "MSFT", "GE", "JNJ", "WMT", "CVX", "PG", "WFC")
> getSymbols(myStocks, src = "yahoo")
> returns <- lapply(myStocks, function(s)
                    monthlyReturn(eval(parse(text = s)),
                                   subset = "2013::2014"))
> returns <- do.call(cbind, returns)
> colnames(returns) <- myStocks
```

The monthly return rates for the portfolio stocks are shown in Figure 2 and obtained with the code:

```
> library(timeSeries)
> plot(as.timeSeries(returns), at = "chic", minor.ticks="month",
       mar.multi = c(0.2, 5.1, 0.2, 1.1), oma.multi = c(4, 0, 4, 0),
       col = .colorwheelPalette(10), cex.lab = 0.8, cex.axis = 0.8)
> title("Portfolio Returns")
```

Summary statistics for the portfolio stocks are computed as:

```
> nStocks <- ncol(returns) # number of portfolio assets
> R <- colMeans(returns)   # average monthly returns
> S <- cov(returns)        # covariance matrix of monthly returns
> s <- sqrt(diag(S))       # volatility of monthly returns
> plot(s, R, type = "n", panel.first = grid(),
       xlab = "Std. dev. monthly returns", ylab = "Average monthly returns")
> text(s, R, names(R), col = .colorwheelPalette(10), font = 2)
```

The last two commands draw a graph of the average vs standard deviation for the monthly returns (see Figure 3a). From this graph we can see that there exists a high degree of heterogeneity among stocks, with AAPL having the largest standard deviation and negative average return, whereas some stocks have small volatility and high returns, such as WFC and MSFT. Clearly, the latter are good candidate for inclusion in the portfolio. The exact amount of each stock also depends on the correlation among stocks through the variance of portfolio returns σ_p^2 , and so we need to formalise our objective function under the given constraints.

In order to compute the GA fitness function, we define the following functions:

```
> weights <- function(w)      # normalised weights
  { drop(w/sum(w)) }
> ExpReturn <- function(w)    # expected return
  { sum(weights(w)*R) }
> VarPortfolio <- function(w) # objective function
  {
    w <- weights(w)
    drop(w %**% S %**% w)
  }
```

We may define the fitness function to be maximised as the (negative) variance of the portfolio penalised by an amount which is function of the distance between the expected return of the portfolio and the target value:

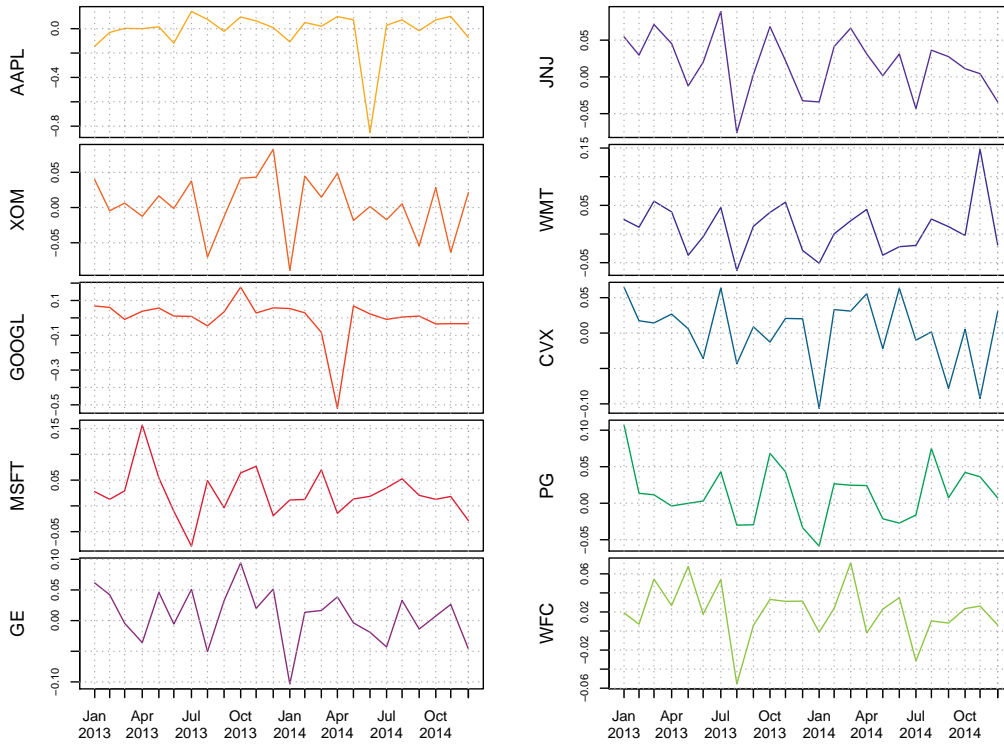


Figure 2: Monthly return rates for a portfolio of selected stocks.

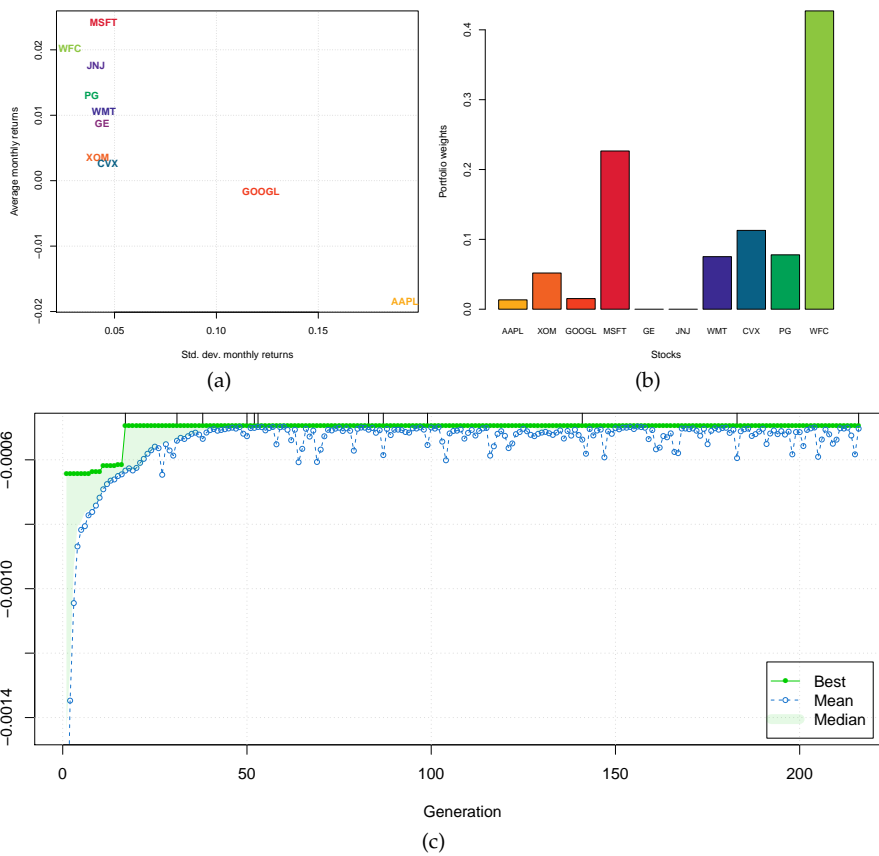


Figure 3: (a) Plot of average monthly returns vs the standard deviation for the selected stocks. (b) Portfolio stocks composition estimated by HGA. (c) Trace of HGA iterations.

```
> fitness <- function(w)      # fitness function
{
  ER <- ExpReturn(w)-0.01
  penalty <- if(ER < 0) 100*ER^2 else 0
  -(VarPortfolio(w) + penalty)
}
```

A hybrid GA with local search can be obtained with the following call:

```
> GA <- ga(type = "real-valued", fitness = fitness,
           min = rep(0, nStocks), max = rep(1, nStocks), names = myStocks,
           maxiter = 1000, run = 200, optim = TRUE)
> summary(GA)
+-----+
|           Genetic Algorithm           |
+-----+
```

GA settings:

```
Type                = real-valued
Population size      = 50
Number of generations = 1000
Elitism              = 2
Crossover probability = 0.8
Mutation probability = 0.1
Search domain =
  AAPL  XOM  GOOGL  MSFT  GE  JNJ  WMT  CVX  PG  WFC
Min    0   0     0     0   0   0   0   0   0   0
Max    1   1     1     1   1   1   1   1   1   1
```

GA results:

```
Iterations           = 216
Fitness function value = -0.00049345
Solution =
  AAPL    XOM    GOOGL    MSFT  GE  JNJ    WMT    CVX    PG    WFC
[1,] 0.030918 0.11534 0.034683 0.52062 0   0 0.17201 0.26144 0.18096 0.98719
> plot(GA)
```

The last command produces the graph on Figure 3c, which shows the trace of best, mean, and median values during the HGA iterations. We also added some vertical dashes at the top of the graph to indicate when the local search occurred. It is interesting to note that the inclusion of a local search greatly speeds up the termination of the GA search, which converges after 216 iterations. Without including the local optimisation step, a fitness function value within a 1% from the maximum value found above is attained after 1,633 iterations, whereas the same maximum fitness value cannot be achieved even after 100,000 iterations.

The estimated portfolio weights and the corresponding expected return and variance are computed as:

```
> (w <- weights(GA@solution))
  AAPL    XOM    GOOGL    MSFT    GE    JNJ    WMT    CVX
0.013424 0.050081 0.015059 0.226047 0.000000 0.000000 0.074685 0.113512
  PG    WFC
0.078572 0.428621
> ExpReturn(w)
[1] 0.016178
> VarPortfolio(w)
[1] 0.00049345
> barplot(w, xlab = "Stocks", ylab = "Portfolio weights",
          cex.names = 0.7, col = .colorwheelPalette(10))
```

The last command draws a barchart of the optimal portfolio selected, and it is shown in Figure 3b.

Poisson change-point model

In the study of stochastic processes a common problem is to determine whether or not the functioning of a process has been modified over time. Change-point models assume that such a change is occurring at some point in time in a relatively abrupt manner (Lindsey, 2004).

In a single change-point model the distribution of a response variable Y_t at time t is altered at the unknown point in time τ , so we can write

$$Y_t \sim \begin{cases} f(y_t; \theta_1) & t < \tau \\ f(y_t; \theta_2) & t \geq \tau \end{cases} \quad (1)$$

where $f(\cdot)$ is some given parametric distribution depending on θ_k for $k = \{1, 2\}$, and τ is an unknown parameter giving the change-point time. Some or all of the elements of the vector of parameters θ_k in model (1) may change over time. In more complex settings, the distribution function itself may be different before and after the change point.

Given a sample $\{y_t; t = 1, \dots, T\}$ of observations over time, the log-likelihood function of the change-point problem is

$$\ell(\theta_1, \theta_2, \tau; y_1, \dots, y_T) = \sum_{t < \tau} \log f(y_t; \theta_1) + \sum_{t \geq \tau} \log f(y_t; \theta_2) \quad (2)$$

Further, for a Poisson change-point model we assume that $f(y_t; \theta_k)$ is the Poisson distribution with mean parameter θ_k . Maximisation of (2) can also be seen as a discrete optimisation problem.

Consider the British coal-mining disasters dataset which provides the annual counts of disasters (having at least 10 deaths) from 1851 to 1962 (Jarrett, 1979; Raftery and Akman, 1986). The data from Table 1 of Carlin et al. (1992) are the following:

```
> data <- data.frame(
  y = c(4, 5, 4, 1, 0, 4, 3, 4, 0, 6, 3, 3, 4, 0, 2, 6, 3, 3, 5, 4, 5, 3, 1,
        4, 4, 1, 5, 5, 3, 4, 2, 5, 2, 2, 3, 4, 2, 1, 3, 2, 2, 1, 1, 1, 1, 3,
        0, 0, 1, 0, 1, 1, 0, 0, 3, 1, 0, 3, 2, 2, 0, 1, 1, 1, 0, 1, 0, 1, 0,
        0, 0, 2, 1, 0, 0, 0, 1, 1, 0, 2, 3, 3, 1, 1, 2, 1, 1, 1, 1, 2, 4, 2,
        0, 0, 0, 1, 4, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1),
  year = 1851:1962,
  t = 1:112)
```

Graphs of annual counts and cumulative sums over time are shown in Figure 4. These can be obtained using the following code:

```
> plot(y ~ year, data = data, ylab = "Number of mine accidents/yr")
> plot(cumsum(y) ~ year, data = data, type = "s",
      ylab = "Cumsum number of mine accidents/yr")
```

Both graphs seem to suggest a two-regime behaviour for the number of coal-mining disasters.

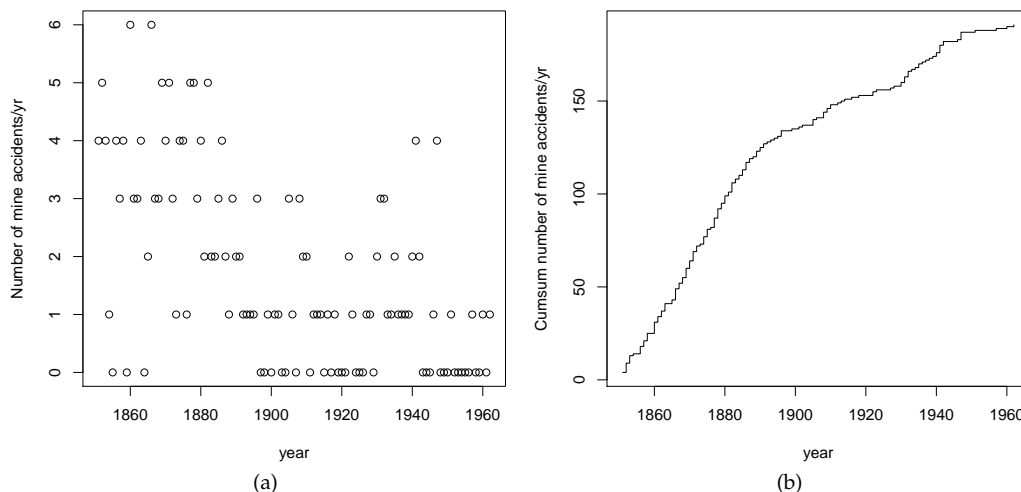


Figure 4: Plots of the number of yearly coal-mining accidents (a) and cumulative sum of mine accidents (b) from 1851 to 1962 in Great Britain.

We start the analysis by fitting a no change-point model, i.e. assuming a homogeneous Poisson process with constant mean. Clearly, in this simple case the MLE of the Poisson parameter is the sample mean of counts. However, for illustrative purposes we write down the log-likelihood and we maximise it with a hybrid GA.

```

> loglik1 <- function(th, data)
{
  mu <- exp(th) # Poisson mean
  sum(dpois(data$y, mu, log = TRUE))
}
> GA1 <- ga(type = "real-valued",
  fitness = loglik1, data = data,
  min = log(1e-5), max = log(6), names = "th",
  maxiter = 200, run = 50,
  optim = TRUE)
> exp(GA1@solution[1,])
1.7054
> mean(data$y)
[1] 1.7054

```

For the change-point model in (1), the mean function can be expressed as

$$\mu_t = \exp\{\theta_1 + (\theta_2 - \theta_1)I(t \geq \tau)\},$$

where τ is the time of change-point, θ_1 is the mean of the first regime, i.e. when $t < \tau$, θ_2 is the mean of the second regime, i.e. when $t \geq \tau$, and $I(\cdot)$ denotes the indicator function (which is equal to 1 if its argument is true and 0 otherwise). In R the above mean function and the log-likelihood from (2) can be written as

```

> meanFun <- function(th, t)
{
  tau <- th[3] # change-point parameter
  th <- th[1:2] # mean-related parameters
  X <- cbind(1, t >= tau) # design matrix
  exp(drop(X %*% th))
}
> loglik2 <- function(th, data)
{
  mu <- meanFun(th, data$t) # vector of Poisson means
  sum(dpois(data$y, mu, log = TRUE))
}

```

The vector `th` contains the three parameters that have to be estimated from the sample dataset `data`. Note that, for convenience, it is defined as $(\theta_1, \theta_2^*, \tau)'$, where $\theta_2^* = (\theta_2 - \theta_1)$ is the differential mean effect of second regime.

Direct maximisation of the log-likelihood in `loglik2()` by iterative derivative-based methods is not viable due to lack of differentiability with respect to τ . However, hybrid GAs can be efficiently used in this case as follows:

```

> GA2 <- ga(type = "real-valued",
  fitness = loglik2, data = data,
  min = c(log(1e-5), log(1e-5), min(data$t)),
  max = c(log(6), log(6), max(data$t)+1),
  names = c("th1", "th2", "tau"),
  maxiter = 1000, run = 200,
  optim = TRUE)
> summary(GA2)
+-----+
|           Genetic Algorithm           |
+-----+

GA settings:
Type                = real-valued
Population size     = 50
Number of generations = 1000
Elitism              = 2
Crossover probability = 0.8
Mutation probability = 0.1
Search domain =
  th1      th2 tau
Min -11.5129 -11.5129 1

```

```
Max 1.7918 1.7918 113
```

```
GA results:
```

```
Iterations = 318
```

```
Fitness function value = -168.86
```

```
Solution =
```

```
      th1    th2    tau
[1,] 1.1306 -1.2344 41.446
```

```
> (mean <- exp(cumsum(GA2@solution[1,1:2]))) # mean function parameters
```

```
      th1    th2
3.09756 0.90141
```

```
> (tau <- GA2@solution[1,3]) # change-point
```

```
      tau
41.446
```

Note that both the estimated change-point and the means are quite close to those reported by [Raftery and Akman \(1986\)](#), and [Carlin et al. \(1992\)](#), using Bayesian methodology.

The two estimated models can be compared using a model selection criterion, such as the Bayesian information criterion (BIC; [Schwartz, 1978](#)), defined as

$$\text{BIC} = 2\ell(\hat{\theta}; y) - \nu \log(n)$$

where $\ell(\hat{\theta}; y)$ is the log-likelihood evaluated at the MLE $\hat{\theta}$, n is the number of observations, and ν is the number of estimated parameters. Using this definition, larger values of BIC are preferable.

```
> (tab <- data.frame(
  loglik = c(GA1@fitnessValue, GA2@fitnessValue),
  df = c(ncol(GA1@solution), ncol(GA2@solution)),
  BIC = c(2*GA1@fitnessValue - log(nrow(data))*ncol(GA1@solution),
    2*GA2@fitnessValue - log(nrow(data))*ncol(GA2@solution))))
```

```
      loglik df      BIC
1 -203.86  1 -412.43
2 -168.86  3 -351.88
```

A comparison of BIC values clearly indicates a preference for the change-point model. We may summarise the estimated model by drawing a graph of observed counts over time with the estimated means before and after the change-point:

```
> mu <- meanFun(GA2@solution, data$t)
> col <- c("red3", "dodgerblue2")
> with(data,
  { plot(t, y)
    abline(v = tau, lty = 2)
    lines(t[t < tau], mu[t < tau], col = col[1], lwd = 2)
    lines(t[t >= tau], mu[t >= tau], col = col[2], lwd = 2)
    par(new=TRUE)
    plot(year, cumsum(y), type = "n", axes = FALSE, xlab = NA, ylab = NA)
    axis(side = 3); mtext("Year", side = 3, line = 2.5)
  })
```

and a graph of observed cumulative counts and the estimated cumulative mean counts:

```
> with(data,
  { plot(t, cumsum(y), type = "s", ylab = "Cumsum number of mine accidents/yr")
    abline(v = tau, lty = 2)
    lines(t[t < tau], cumsum(mu)[t < tau], col = col[1], lwd = 2)
    lines(t[t >= tau], cumsum(mu)[t >= tau], col = col[2], lwd = 2)
    par(new=TRUE)
    plot(year, cumsum(y), type = "n", axes = FALSE, xlab = NA, ylab = NA)
    axis(side = 3); mtext("Year", side = 3, line = 2.5)
  })
```

Both graphs are reported in [Figure 5](#). The latter plot is particularly illuminating of the good fit achieved by the selected model.

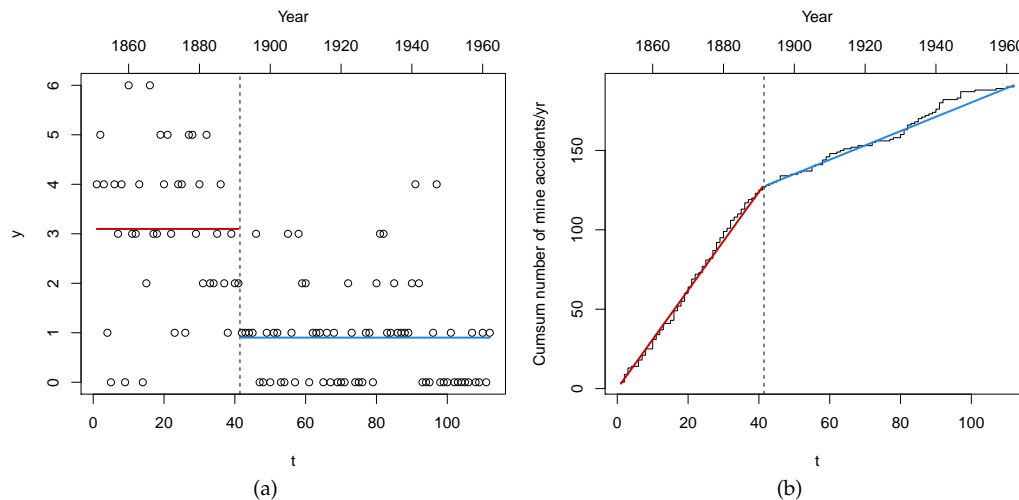


Figure 5: Summary plots for the change-point model fitted to the British coal-mining accidents dataset: (a) plot of observed counts over time with the estimated means before and after the estimated change-point (vertical dashed line); (b) plot of observed cumulative counts (step function) and the cumulative estimated mean counts.

Parallel genetic algorithms

Parallel computing in its essence involves the simultaneous use of multiple computing resources to solve a computational problem. This is viable when a task can be divided into several parts that can be solved simultaneously and independently, either on a single multi-core processors machine or on a cluster of multiple computers.

Support for parallel computing in R is available since 2011 (version 2.14.0) through the base package **parallel**. This provides parallel facilities previously contained in packages **multicore** and **snow**. Several approaches to parallel computing are available in R (McCallum and Weston, 2011), and an extensive and updated list of R packages is reported in the CRAN Task View on *High-Performance and Parallel Computing with R* (Eddelbuettel, 2016, *HighPerformanceComputing*).

GAs are regarded as “embarrassingly parallel” problems, meaning that they require a large number of independent calculations with negligible synchronisation and communication costs. Thus, GAs are particularly suitable for parallel computing, and it is not surprising that such idea has been often exploited to speed up computations (see for instance Whitley (1994) in the statistical literature).

Luque and Alba (2011) identify several types of parallel GAs. In the master-slaves approach there is a single population, as in sequential GAs, but the evaluation of fitness is distributed among several processors (*slaves*). The *master* process is responsible for the distribution of the fitness function evaluation tasks performed by the slaves, and for applying genetic operators such as selection, crossover, and mutation (see Figure 6). Since the latter operations involve the entire population, it is also known as global parallel GAs (GPGA). This approach is generally efficient when the computational time involving the evaluation of the fitness function is more expensive than the communication overhead between processors.

Another approach is the case of distributed multiple-population GAs, where the population is partitioned into several subpopulations and assigned to separated islands. Independent GAs are executed in each island, and only occasionally sparse exchanges of individuals are performed among these islands (see Figure 7). This process, called migration, introduces some diversity into the subpopulations, thus preventing the search from getting stuck in local optima. In principle islands can evolve sequentially, but increased computational efficiency is obtained by running GAs in each island in parallel. This approach is known as coarse-grained GAs or island parallel GAs (ISLPGA).

By default, searches performed with the **GA** package occur sequentially. In some cases, particularly when the evaluation of the fitness function is time consuming, parallelisation of the search algorithm may be able to speed up computing time. Starting with version 2.0, the **GA** package provides facilities for using parallel computing in genetic algorithms following the GPGA approach. Recently, with version 3.0, the ISLPGA model has also been implemented in the **GA** package. The following subsections describes usage of both approaches.

Parallel computing in the **GA** package requires the following packages to be installed: **parallel** (available in base R), **doParallel**, **foreach**, and **iterators**. Moreover, **doRNG** is needed for reproducibil-

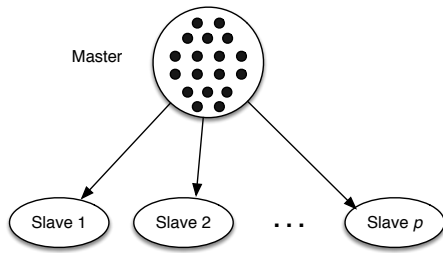


Figure 6: Master-slaves or global parallel GA scheme (GPGA). The master process stores the population, executes genetic operations, and distributes individuals to the slaves, which only evaluate the fitness of individuals.

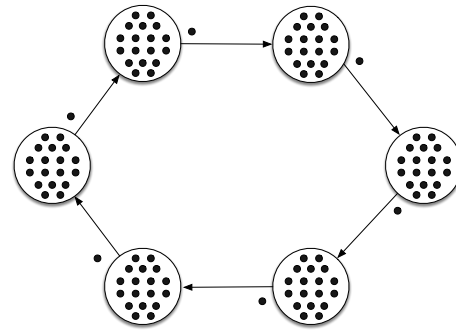


Figure 7: Islands parallel GA scheme (ISLPGA). In a multiple-population parallel GA each process is a simple GA which evolves independently. Individuals occasionally migrate between one island and its neighbours.

ity of results.

Global parallel implementation

The GPGA approach to parallel computing in GA can be easily obtained by manipulating the optional argument `parallel` in the `ga()` function call. This argument accepts several different values. A logical value may be used to specify if parallel computing should be used (`TRUE`) or not (`FALSE`, default) for evaluating the fitness function. A numeric value can also be supplied, in which case it gives the number of cores/processors to employ; by default, all the available cores, as provided by `detectCores()`, are used.

Two types of parallel functionalities are available depending on system OS: on Windows only *snow* type functionality is present, whereas on POSIX operating systems, such as Unix, GNU/Linux, and Mac OSX, both *snow* and *multicore* (default) functionalities are available. In the latter case, a string can be used as the argument to `parallel` to set out which parallelisation tool should be used.

A final option is available if a researcher plans to use a cluster of multiple machines. In this case, `ga()` can be executed in parallel using all, or a subset of, the cores available to each machine assigned to the cluster. However, this option requires more work from the user, who needs to set up and register a parallel back end. The resulting cluster object should then be passed as input for the `parallel` argument.

Islands parallel implementation

The ISLPGA approach to parallel computing in GA has been implemented in the `gaisl()` function. This function accepts the same input arguments as the `ga()` function (see [Scrucca, 2013, Section 3](#)), with the following additional arguments:

- `numIslands` An integer value which specifies the number of islands to use in the genetic evolution (by default is set to 4).
- `migrationRate` A value in the range (0,1) which gives the proportion of individuals that undergo migration between islands in every exchange (by default equal to 0.10).
- `migrationInterval` An integer value specifying the number of iterations at which exchange of individuals takes place. This interval between migrations is called an *epoch*, and it is set at 10 by default.

The implemented ISLPGA uses a simple *ring topology*, in which each island is connected unidirectionally with another island, hence forming a single continuous pathway (see Figure 7). Thus, at each exchange step the top individuals, selected according to the specified `migrationRate`, substitute random individuals (with the exception of the elitist ones) in the connected island.

By default, the function `gaisl()` uses `parallel = TRUE`, i.e. the islands algorithm is run in parallel, but other values can also be provided as described in the previous subsection. Note that it is possible to specify a number of islands larger than the number of available cores. In such a case, the parallel

algorithm will be run using blocks of islands, with the block size depending on the maximal number of cores available or the number of processors as specified by the user.

It has been noted that using parallel islands GAs often leads to, not only faster algorithms, but also superior numerical performance even when the algorithms run on a single processor. This because each island can search in very different regions of the whole search space, thus enhancing the exploratory attitude of evolutionary algorithms.

Simulation study

In this Section results from a simulation study are presented and discussed. The main goal is to compare the performance of sequential GAs with the two forms of parallel algorithms implemented in the GA package, namely GPGA and ISLPGA, for varying number of cores and different fitness computing times. A fictitious fitness function is used to allow for controlling the computing time required at each evaluation. This is achieved by including the argument `pause` which suspend the execution for a specified time interval (in seconds):

```
> fitness <- function(x, pause = 0.1)
  {
    Sys.sleep(pause)
    x*runif(1)
  }
```

The simulation design parameters used are the following:

```
> ncores <- c(1, 2, 4, 8, 16) # number of cores/processors
> pause <- c(0.01, 0.1, 1, 2) # pause during fitness evaluation
> nrep <- 10 # number of simulation replications
```

Thus, `ncores` specifies that up to 16 cores or CPU processors are used in the parallel GAs solutions for increasing time spent on fitness evaluation as specified by `pause` (in seconds). Each combination of design parameters is replicated `nrep = 10` times and results are then averaged.

GAs are run under the GPGA approach using `popSize = 50` and `maxiter = 100`. For ISLPGA runs the `numIslands` argument is set at the specified number of cores, with `popSize = 160` and `maxiter = 100`. The increased population size allows to work with at least 10 individuals on each island when `numIslands` is set at the maximum number of cores. In both cases, the remaining arguments in `ga()` or `gaisl()` function are set at their defaults. Note that run times cannot be compared between the two approaches because they use different population sizes.

The study was performed on a 16 cores Intel® Xeon® CPU E5-2630 running at 2.40GHz and with 128GB of RAM. The R code used in the simulation study is provided in the accompanying supplemental material.

Graphs in the left panel of Figures 8 and 9 show the average execution times needed for varying number of cores and different fitness computing times. As expected, increasing the number of cores allows to run GAs faster, but the improvement is not linear, in particular for the GPGA approach.

By using a machine with P cores/processors, we would like to obtain an increase in calculation speed of P times. However, this is typically not the case because in the implementation of a parallel algorithm there are some inherent non-parallelisable parts and communication costs between tasks (Nakano, 2012). The speedup achieved using P processors is computed as $s_P = t_1/t_P$, where t_i is the execution time spent using i cores. Graphs in the right panel of Figures 8 and 9 show the speedup obtained in our simulation study. For the GPGA approach the speedup is quite good but it is always sub-linear, in particular for the less demanding fitness evaluation time and when the number of cores increases. On the other hand, the ISLPGA implementation shows a very good speedup (nearly linear).

Amdahl's law (Amdahl, 1967) is often used in parallel computing to predict the theoretical maximum speedup when using multiple processors. According to this, if f is the fraction of non-parallelisable task, i.e. the part of the algorithm that is strictly serial, and P is the number of processors in use, then the speedup obtained on a parallel computing platform follows the equation

$$S_P = \frac{1}{f + (1 - f)/P}. \quad (3)$$

In the limit, the above ratio converges to $S_{\max} = 1/f$, which represents the maximum speedup attainable in theory, i.e. by a machine with an infinite number of processors. Figures 10 and 11 show the observed speedup factors S_P and the estimated Amdahl's law curves fitted by nonlinear least squares. In all the cases, Amdahl's law appears to well approximate the observed behaviour. The horizontal dashed lines are drawn at the maximum speedup S_{\max} , which is computed based on the

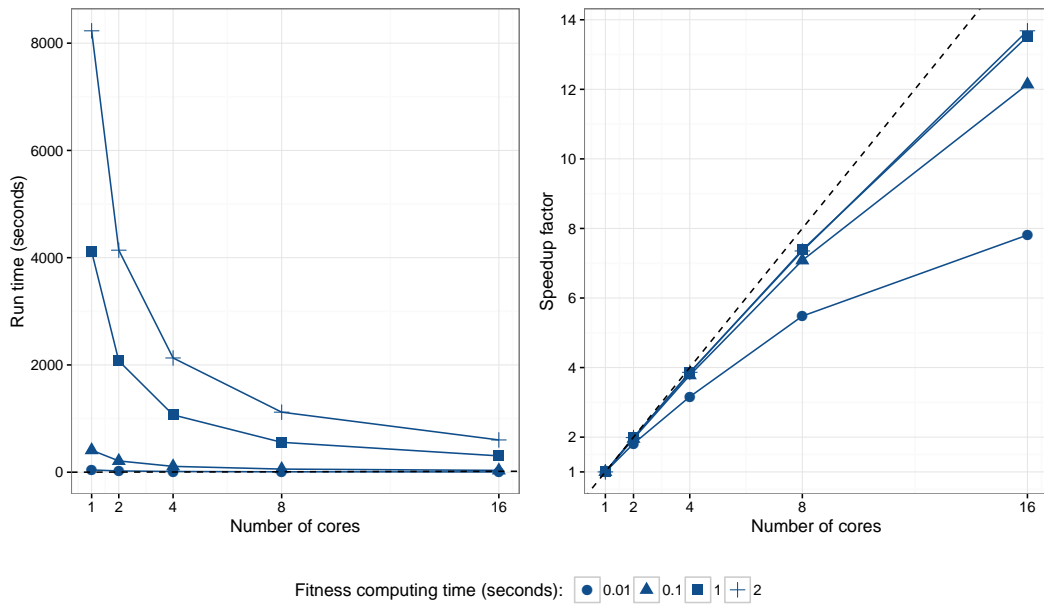


Figure 8: Empirical GPGA performance for varying number of cores/processors and different fitness computing times. Graph on the left panel shows the average running times, whereas graph on the right panel shows the speedup factor compared to the sequential run (i.e. when only 1 core is used). In the latter plot, the dashed line represents the “ideal” linear speedup.

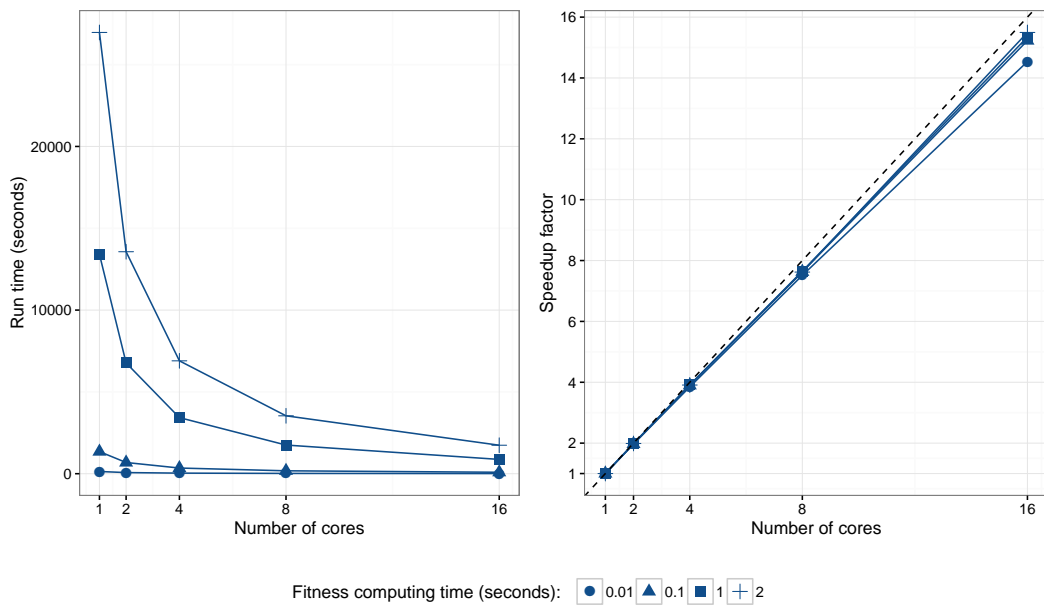


Figure 9: Empirical ISLPGA performance for varying number of cores/processors and different fitness computing times. Graph on the left panel shows the average running times, whereas graph on the right panel shows the speedup factor compared to the sequential run (i.e. when only 1 core is used). In the latter plot, the dashed line represents the “ideal” linear speedup.

estimated fraction of non-parallelisable task f (see also Table 2). As the time required for evaluating the fitness function increases, the maximum speedup attainable also increases. As noted earlier, the ISLPGA approach shows an improved efficiency compared to the simple GPGA.

Table 2: Fraction of non-parallelisable task (f) estimated by nonlinear least squares using the Amdahl’s law (3), and corresponding theoretical speedup (S_{max}) for the GPGA and ISLPGA approaches.

	GPGA				ISLPGA			
	0.01	0.1	1	2	0.01	0.1	1	2
f	0.0695	0.0209	0.0122	0.0114	0.0069	0.0036	0.0031	0.0025
S_{max}	14.38	47.76	81.88	87.88	145.29	278.57	327.12	408.58

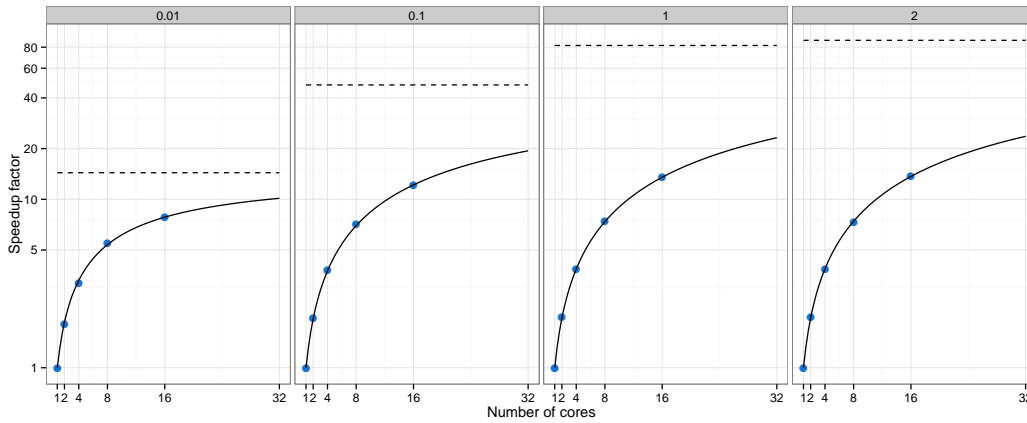


Figure 10: Amdahl’s law curves for the GPGA approach. Points refer to the speedup factors observed using different number of cores/processors, whereas the curves are estimated using nonlinear least squares. Horizontal dashed lines refer to the maximum speedup theoretically attainable. Each panel corresponds to a different fitness computing time (in seconds), and vertical axes are on log scale.

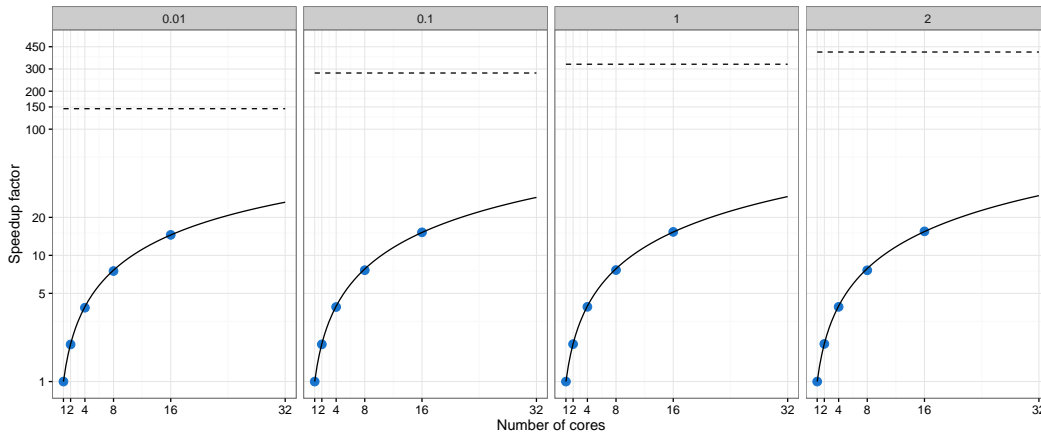


Figure 11: Amdahl’s law curves for the ISLPGA approach. Points refer to the speedup factors observed using different number of cores/processors, whereas the curves are estimated using nonlinear least squares. Horizontal dashed lines refer to the maximum speedup theoretically attainable. Each panel corresponds to a different fitness computing time (in seconds), and vertical axes are on log scale.

ARIMA order selection

Autoregressive moving average (ARMA) models are a broad class of parametric models for stationary time series popularised by [Box and Jenkins \(1976\)](#). They provide a parsimonious description of a stationary stochastic process in terms of two polynomials, one for the auto-regression and the second for the moving average. Nonstationary time series can be modelled by including an initial differencing step (“integrated” part of the model). This leads to autoregressive integrated moving average (ARIMA) models, a popular modelling approach in real-world processes.

ARIMA models can be fitted by MLE after identifying the order (p, d, q) for the autoregressive, integrated, and moving average components, respectively. This is typically achieved by preliminary inspection of the autocovariance function (ACF) and partial autocovariance function (PACF). Model selection criteria, such as the Akaike information criterion (AIC), the corrected AIC (AICc), and the

Bayesian information criterion (BIC), are also used for order selection.

The function `auto.arima()` in package `forecast` (Hyndman, 2016; Hyndman and Khandakar, 2008) provides an automatic algorithm which combines unit root tests, minimisation of the AICc in a stepwise greedy search, and MLE, to select the order of an ARIMA model. Here, an island parallel GAs approach is used for order selection.

Consider the quarterly U.S. GNP from 1947(1) to 2002(3) expressed in billions of chained 1996 dollars and seasonally adjusted. The data are available on package `astsa` and described in Shumway and Stoffer (2013).

```
> data(gnp, package="astsa")
> plot(gnp)
```

The plot of the time series obtained with the last command is shown in Figure 12a.

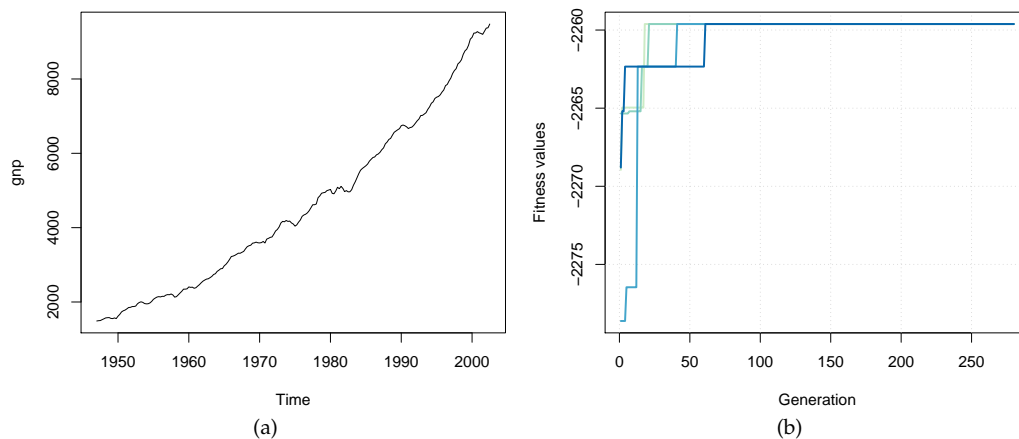


Figure 12: (a) Plot of quarterly U.S. GNP from 1947(1) to 2002(3). (b) Trace of island parallel GAs search for ARIMA order selection.

The selection of the “optimal” $ARIMA(p, d, q)$ model can be pursued by using binary GAs to maximise the BIC. The decision variables to be optimised are expressed in binary digits using the following function:

```
> decode <- function(string, bitOrders)
{
  string <- split(string, rep.int(seq.int(bitOrders), times = bitOrders))
  orders <- sapply(string, function(x) { binary2decimal(gray2binary(x)) })
  return(unnamed(orders))
}
```

For example, using 3 bits for encoding p and q , and 2 bits for d , an $ARIMA(3,1,1)$ model can be expressed with the binary string $(0, 1, 0, 0, 1, 0, 0, 1)$:

```
> decode(c(0,1,0, 0,1, 0,0,1), bitOrders = c(3,2,3))
[1] 3 1 1
```

Note that the `decode()` function assumes that the input binary string is expressed using Gray encoding, which ensures that consecutive values have the same Hamming distance (Hamming, 1950).

The fitness function to be used in the GA search is defined as follows:

```
> fitness <- function(string, data, bitOrders)
{
  orders <- decode(string, bitOrders)
  mod <- try(Arima(data, order = orders, include.constant = TRUE, method = "ML"),
            silent = TRUE)
  if(inherits(mod, "try-error")) NA else -mod$bic
}
```

Note that the objective function is defined as (minus) the BIC for the specified ARIMA model, with the latter fitted using the `Arima()` function available in the R package `forecast`. If a different criterion

for model selection is preferred, for instance the Akaike information criterion (AIC), only the last line of the above `fitness()` function needs to be modified.

An island binary parallel GA is then used to search for the best ARIMA model, using a migration interval of 20 generations, and the default migration rate of 0.1:

```
> GA <- gaisl(type = "binary", nBits = 8,
             fitness = fitness, data = gnp, bitOrders = c(3,2,3),
             maxiter = 1000, run = 100, popSize = 50,
             numIslands = 4, migrationInterval = 20)
> plot(GA)
> summary(GA)
+-----+
|           Genetic Algorithm           |
|           Islands Model              |
+-----+
```

GA settings:

```
Type           = binary
Number of islands = 4
Islands pop. size = 12
Migration rate   = 0.1
Migration interval = 20
Elitism          = 1
Crossover probability = 0.8
Mutation probability = 0.1
```

GA results:

```
Iterations           = 280
Epochs              = 14
Fitness function values = -2259.615 -2259.615 -2259.615 -2259.615
Solutions =
      x1 x2 x3 x4 x5 x6 x7 x8
[1,]  0  1  1  1  1  0  0  1
[2,]  0  1  1  1  1  0  0  1
[3,]  0  1  1  1  1  0  0  1
[4,]  0  1  1  1  1  0  0  1
```

Figure 12b shows the trace of the ISLPGA search for each of the four islands used. All the islands converge to the same final solution, as also shown by the summary output above. The selected model is an ARIMA(2,2,1), which can be fitted using:

```
> (orders <- decode(GA@solution[1,], c(3,2,3)))
[1] 2 2 1
> mod <- Arima(gnp, order = orders, include.constant = TRUE, method = "ML")
> mod
Series: gnp
ARIMA(2,2,1)

Coefficients:
      ar1      ar2      ma1
      0.2799  0.1592 -0.9735
s.e.  0.0682  0.0682  0.0143

sigma^2 estimated as 1451: log likelihood=-1119.01
AIC=2246.02  AICc=2246.21  BIC=2259.62
```

It is interesting to compare the above solution with that obtained with the automatic procedure implemented in `auto.arima()` using the same criterion:

```
> mod1 <- auto.arima(gnp, ic = "bic")
> print(mod1)
Series: gnp
ARIMA(1,2,1)

Coefficients:
      ar1      ma1
```

```

0.3243 -0.9671
s.e. 0.0665 0.0162

sigma^2 estimated as 1486: log likelihood=-1121.71
AIC=2249.43 AICc=2249.54 BIC=2259.62
> mod1$bic
[1] 2259.622
> mod$bic
[1] 2259.615
    
```

The model returned by `auto.arima()` is an ARIMA(1,2,1), so a simpler model where an AR(1) component is chosen instead of an AR(2). The BIC values are almost equivalent, with a slightly smaller value for the ARIMA(2,2,1) model identified by ISLPGA. However, by looking at some diagnostic plots it seems that a second-order AR component is really needed to account for autocorrelation at several lags as indicated by the Ljung-Box test of autocorrelation (see Figure 13; the code used to produce the plots is available in the supplementary material).

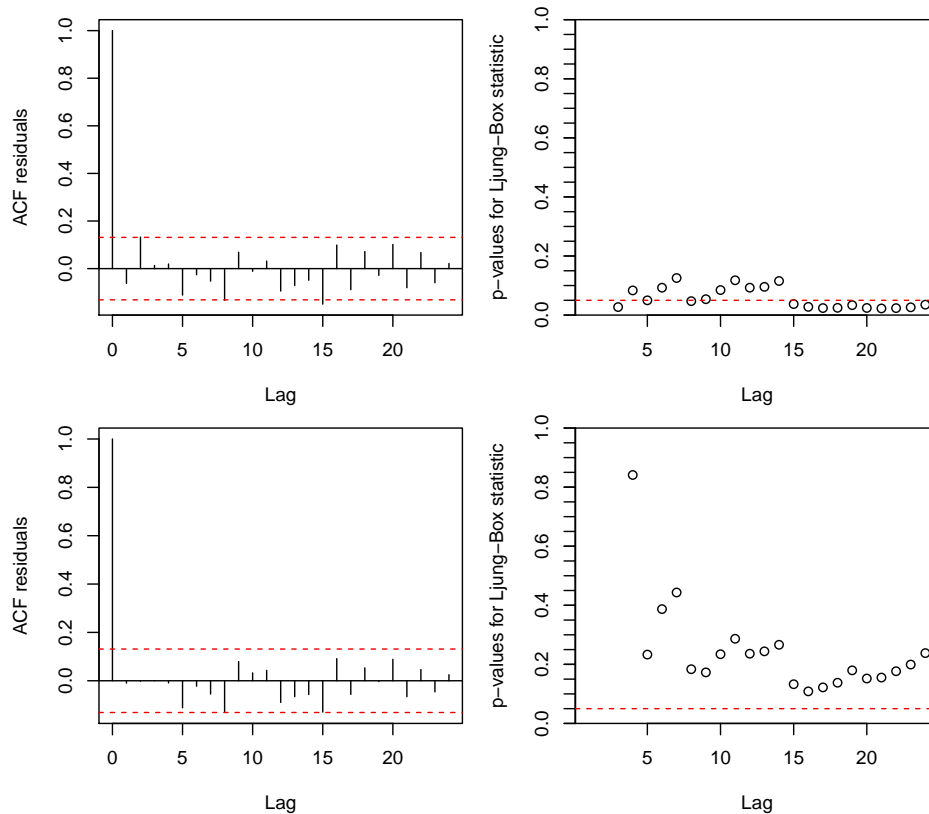


Figure 13: ACF of residuals and p-values for the Ljung-Box test of autocorrelation for the ARIMA(1,2,1) model (top graphs) and the ARIMA(2,2,1) model (bottom graphs) fitted to the quarterly U.S. GNP data from 1947(1) to 2002(3).

Benchmark function optimisation

Mullen (2014) compared several optimisation algorithms using 48 benchmark functions available in the `globalOptTests` package. GA was one of the several R packages investigated in such a comparison. However, with the settings used in this study, its overall performance was not particularly brilliant, ranking 14th out of 18 methods, thus leaving plenty of room for improvements. For instance, the Griewank function was one of the most problematic cases, and it is defined as

$$f(x_1, \dots, x_d) = 1 + \frac{1}{4000} \sum_{i=1}^d x_i^2 - \prod_{i=1}^d \cos(x_i / \sqrt{i}).$$

This a multimodal, non-separable function, with several local optima within the search region. For any dimensionality d , it has one global minimum of zero located at the point $(0, \dots, 0)$.

We replicated the simulation study in Mullen (2014) using the standard sequential GA (GA), the parallel island GA with 4 islands (GAISL), the hybrid GA with local search (HGA), and the island GA with local search (HGAISL). Results for the Griewank function based on 100 replications are shown in Figure 14. The use of hybrid GAs, particularly in combination with the islands evolution, clearly yields more accurate solutions and with less dispersion. The same behavior has been observed in many other benchmark functions available in the `globalOptTests` package.

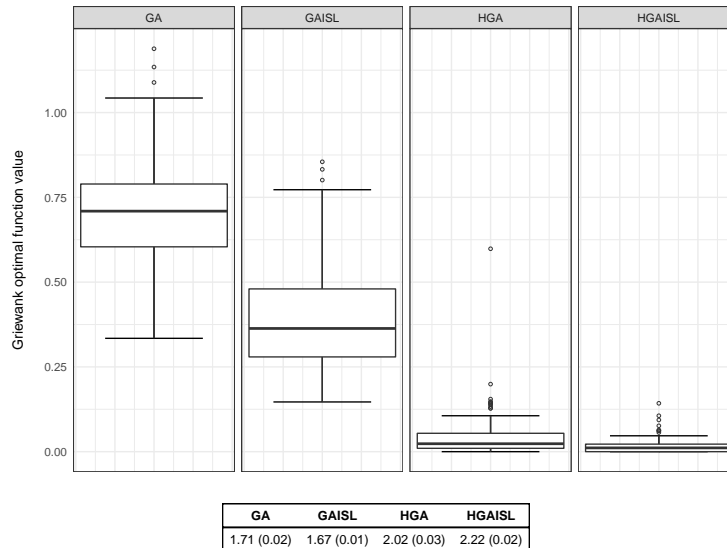


Figure 14: Results from 100 replications of Griewank function optimisation using standard GAs (GA), island GAs (GAISL), hybrid GAs with local search (HGA), and island GAs with local search (HGAISL). The table at the bottom reports the average (and standard deviation) computing times in seconds used by each algorithm.

Summary

GA is a flexible R package for solving optimisation problems with genetic algorithms. This paper presents some improvements recently added to the package. We have discussed the implementation of hybrid GAs, which employ local searches during the evolution of a GA to improve accuracy and efficiency. Further speedup can also be achieved by parallel computing. This has been implemented following two different approaches. In the first one, the so-called master-slave approach, the fitness function is evaluated in parallel, either on a single multi-core machine or on a cluster of multiple computers. In the second approach, called islands model, the evolution takes place independently on several sub-populations assigned to different islands, with occasional migration of solutions between islands. Both enhancements often lead to high-quality solutions more efficiently.

Future plans include the possibility to improve the overall performance by rewriting some key functions in C++ using the `Rcpp` package. In particular, coding of genetic operators in C++ should provide sensible benefits in terms of computational speedup. Another interesting option to perform local search is the inclusion of Hooke-Jeeves direct search algorithm as described in Satman (2015). Finally, the package `memoise` enables to store the results of an expensive fitness function call and returns the cached result when the same input arguments occur again. This strategy could be conveniently employed in the case of binary and permutation GAs.

Acknowledgements

The author acknowledges the CINECA award under the ISCRA initiative (<http://www.hpc.cineca.it/services/iscra>) for the availability of high performance computing resources and support.

Bibliography

G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967. [p198]

- T. Back, D. B. Fogel, and Z. Michalewicz. *Evolutionary Computation 1: Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol and Philadelphia, 2000a. [p187]
- T. Back, D. B. Fogel, and Z. Michalewicz. *Evolutionary Computation 2: Advanced Algorithms and Operators*. IOP Publishing Ltd., Bristol and Philadelphia, 2000b. [p187]
- G. E. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, CA, 1976. [p200]
- B. P. Carlin, A. E. Gelfand, and A. F. Smith. Hierarchical bayesian analysis of changepoint problems. *Applied Statistics*, 41:389–405, 1992. [p193, 195]
- E. K. Chong and S. H. Zak. *An Introduction to Optimization*. John Wiley & Sons, Hoboken, New Jersey, 2013. [p187]
- P. Cortez. *Modern Optimization with R*. Springer-Verlag, 2014. [p187]
- K. A. De Jong. *Evolutionary Computation: a Unified Approach*. MIT press, 2006. [p187]
- M. Dorigo and T. Stützle. *Ant Colony Optimization*. The MIT Press, Cambridge, 2004. [p187]
- D. Eddelbuettel. CRAN task view: High-performance and parallel computing with R, 2016. URL <http://CRAN.R-project.org/view=HighPerformanceComputing>. Version 2016-01-05. [p196]
- A. E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35:35–50, 1998. [p188]
- A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin Heidelberg, 2003. [p187, 188]
- M. Gilli and E. Schumann. Heuristic optimisation in financial modelling. *Annals of Operations Research*, 193(1):129–158, 2012. [p190]
- M. Gilli, D. Maringer, and E. Schumann. *Numerical Methods and Optimization in Finance*. Academic Press, 2011. [p189]
- G. H. Givens and J. A. Hoeting. *Computational Statistics*. John Wiley & Sons, 2nd edition edition, 2013. [p187]
- F. Glover and M. Laguna. *Tabu Search*. Springer-Verlag, 2013. [p187]
- R. W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950. [p201]
- N. Hansen. The cma evolution strategy: a comparing review. In J. A. Lozano, P. Larrañga, I. Inza, and E. Bengoetxea, editors, *Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithms.*, pages 75–102. Springer-Verlag, 2006. [p187]
- R. J. Hyndman. *forecast: Forecasting Functions for Time Series and Linear Models*, 2016. URL <http://github.com/robjhyndman/forecast>. R package version 7.3. [p201]
- R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008. URL <http://www.jstatsoft.org/article/view/v027i03>. [p201]
- R. G. Jarrett. A note on the intervals between coal-mining disasters. *Biometrika*, 66(1):191–193, 1979. [p193]
- J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948, 1995. URL <https://doi.org/10.1109/icnn.1995.488968>. [p187]
- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598): 671–680, 1983. [p187]
- J. K. Lindsey. *Statistical Analysis of Stochastic Processes in Time*. Cambridge University Press, 2004. [p192]
- H. R. Lourenço, O. C. Martin, and T. Stützle. *Iterated Local Search*. Springer-Verlag, 2003. [p187]
- S. Luke. *Essentials of Metaheuristics*. Lulu, 2nd edition, 2013. URL <http://cs.gmu.edu/~sean/book/metaheuristics/>. Freely available at <http://cs.gmu.edu/~sean/book/metaheuristics/>. [p187]

- G. Luque and E. Alba. *Parallel Genetic Algorithms: Theory and Real World Applications*. Springer-Verlag, 2011. [p196]
- E. McCallum and S. Weston. *Parallel R*. O'Reilly Media, 2011. [p196]
- K. M. Mullen. Continuous global optimization in R. *Journal of Statistical Software*, 60(6):1–45, 2014. URL <http://www.jstatsoft.org/v60/i06/>. [p203, 204]
- J. Nakano. Parallel computing techniques. In J. E. Gentle, W. K. Härdle, and Y. Mori, editors, *Handbook of Computational Statistics*, pages 243–271. Springer-Verlag, 2nd ed. edition, 2012. [p198]
- J. C. Nash. *Nonlinear Parameter Optimization Using R Tools*. John Wiley & Sons, 2014. [p187]
- C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Inc., Mineola, NY, 1998. [p187]
- A. Raftery and V. Akman. Bayesian analysis of a poisson process with a change-point. *Biometrika*, 73(1):85–89, 1986. [p193, 195]
- D. Ruppert and D. S. Matteson. *Statistics and Data Analysis for Financial Engineering*. Springer-Verlag, 2nd edition, 2015. URL <https://doi.org/10.1007/978-1-4939-2614-5>. [p189]
- M. H. Satman. Hybridization of floating-point genetic algorithms using Hooke-Jeeves algorithm as an intelligent mutation operator. *Journal of Mathematical and Computational Science*, 5(3):320, 2015. [p204]
- G. Schwartz. Estimating the dimension of a model. *The Annals of Statistics*, 6:31–38, 1978. [p195]
- L. Scrucca. GA: A package for genetic algorithms in R. *Journal of Statistical Software*, 53(4):1–37, 2013. URL <http://www.jstatsoft.org/v53/i04/>. [p188, 197]
- R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Its Applications*. Springer-Verlag, 3rd edition, 2013. [p201]
- D. Simon. *Evolutionary Optimization Algorithms*. John Wiley & Sons, 2013. [p187]
- R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. ISSN 1573-2916. URL <https://doi.org/10.1023/a:1008202821328>. [p187]
- S. Theussl and H. W. Borchers. CRAN task view: Optimization and mathematical programming, 2015. URL <http://CRAN.R-project.org/view=Optimization>. Version 2015-11-27. [p187]
- M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3):1–35, 2013. ISSN 0360-0300. URL <https://doi.org/10.1145/2480741.2480752>. [p188]
- D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994. [p188, 196]

Luca Scrucca
Università degli Studi di Perugia
Via A. Pascoli 20, 06123 Perugia
Italy
luca.scrucca@unipg.it

imputeTS: Time Series Missing Value Imputation in R

by Steffen Moritz and Thomas Bartz-Beielstein

Abstract The **imputeTS** package specializes on univariate time series imputation. It offers multiple state-of-the-art imputation algorithm implementations along with plotting functions for time series missing data statistics. While imputation in general is a well-known problem and widely covered by R packages, finding packages able to fill missing values in univariate time series is more complicated. The reason for this lies in the fact, that most imputation algorithms rely on inter-attribute correlations, while univariate time series imputation instead needs to employ time dependencies. This paper provides an introduction to the **imputeTS** package and its provided algorithms and tools. Furthermore, it gives a short overview about univariate time series imputation in R.

Introduction

In almost every domain from industry (Billinton et al., 1996) to biology (Bar-Joseph et al., 2003), finance (Taylor, 2007) up to social science (Gottman, 1981) different time series data are measured. While the recorded datasets itself may be different, one common problem are missing values. Many analysis methods require missing values to be replaced with reasonable values up-front. In statistics this process of replacing missing values is called *imputation*.

Time series imputation thereby is a special sub-field in the imputation research area. Most popular techniques like Multiple Imputation (Rubin, 1987), Expectation-Maximization (Dempster et al., 1977), Nearest Neighbor (Vacek and Ashikaga, 1980) and Hot Deck (Ford, 1983) rely on inter-attribute correlations to estimate values for the missing data. Since univariate time series do not possess more than one attribute, these algorithms cannot be applied directly. Effective univariate time series imputation algorithms instead need to employ the inter-time correlations.

On CRAN there are several packages solving the problem of imputation of multivariate data. Most popular and mature (among others) are **AMELIA** (Honaker et al., 2011), **mice** (van Buuren and Groothuis-Oudshoorn, 2011), **VIM** (Kowarik and Templ, 2016) and **missMDA** (Josse and Husson, 2016). However, since these packages are designed for multivariate data imputation only they do not work for univariate time series.

At the moment **imputeTS** (Moritz, 2017a) is the only package on CRAN that is solely dedicated to univariate time series imputation and includes multiple algorithms. Nevertheless, there are some other packages that include imputation functions as addition to their core package functionality. Most noteworthy being **zoo** (Zeileis and Grothendieck, 2005) and **forecast** (Hyndman, 2017). Both packages offer also some advanced time series imputation functions. The packages **spacetime** (Pebesma, 2012), **timeSeries** (Rmetrics Core Team et al., 2015) and **xts** (Ryan and Ulrich, 2014) should also be mentioned, since they contain some very simple but quick time series imputation methods. For a broader overview about available time series imputation packages in R see also (Moritz et al., 2015). In this technical report we evaluate the performance of several univariate imputation functions in R on different time series.

This paper is structured as follows: Section **Overview imputeTS package** gives an overview, about all features and functions included in the **imputeTS** package. This is followed by **Usage examples** of the different provided functions. The paper ends with a **Conclusions** section.

Overview imputeTS package

The **imputeTS** package can be found on CRAN and is an easy to use package that offers several utilities for ‘univariate, equi-spaced, numeric time series’.

Univariate means there is just one attribute that is observed over time. Which leads to a sequence of single observations $o_1, o_2, o_3, \dots, o_n$ at successive points $t_1, t_2, t_3, \dots, t_n$ in time. Equi-spaced means, that time increments between successive data points are equal $|t_1 - t_2| = |t_2 - t_3| = \dots = |t_{n-1} - t_n|$. Numeric means that the observations are measurable quantities that can be described as a number.

In the first part of this section, a general overview about all available functions and datasets is given.

This is followed by more detailed overviews about the three areas covered by the package: 'Plots & Statistics', 'Imputation' and 'Datasets'. Information about how to apply these functions and tools can be found later in the [Usage examples](#) section.

General overview

As can be seen in Table 1, beyond several imputation algorithm implementations the package also includes plotting functions and datasets. The imputation algorithms can be divided into rather simple but fast approaches like mean imputation and more advanced algorithms that need more computation time like kalman smoothing on a structural model.

Simple Imputation	Imputation	Plots & Statistics	Datasets
na.locf	na.interpolation	plotNA.distribution	tsAirgap
na.mean	na.kalman	plotNA.distributionBar	tsAirgapComplete
na.random	na.ma	plotNA.gapsize	tsHeating
na.replace	na.seadec	plotNA.imputations	tsHeatingComplete
na.remove	na.seasplit	statsNA	tsNH4
			tsNH4Complete

Table 1: General Overview imputeTS package

As a whole, the package aims to support the user in the complete process of replacing missing values in time series. This process starts with analyzing the distribution of the missing values using the `statsNA` function and the plots of `plotNA.distribution`, `plotNA.distributionBar`, `plotNA.gapsize`. In the next step the actual imputation can take place with one of the several algorithm options. Finally, the imputation results can be visualized with the `plotNA.imputations` function. Additionally, the package contains three datasets, each in a version with and without missing values, that can be used to test imputation algorithms.

Plots & statistics functions

An overview about the available plots and statistics functions can be found in Table 2. To get a good impression what the plots look like section [Usage examples](#) is recommended.

Function	Description
<code>plotNA.distribution</code>	Visualize Distribution of Missing Values
<code>plotNA.distributionBar</code>	Visualize Distribution of Missing Values (Barplot)
<code>plotNA.gapsize</code>	Visualize Distribution of NA gap sizes
<code>plotNA.imputations</code>	Visualize Imputed Values
<code>statsNA</code>	Print Statistics about the Missing Data

Table 2: Overview Plots & Statistics

The `statsNA` function calculates several missing data statistics of the input data. This includes overall percentage of missing values, absolute amount of missing values, amount of missing value in different sections of the data, longest series of consecutive NAs and occurrence of consecutive NAs. The `plotNA.distribution` function visualizes the distribution of NAs in a time series. This is done using a standard time series plot, in which areas with missing data are colored red. This enables the user to see at first sight where in the series most of the missing values are located. The `plotNA.distributionBar` function provides the same insights to users, but is designed for very large time series. This is necessary for time series with 1000 and more observations, where it is not possible to plot each observation as a single point. The `plotNA.gapsize` function provides information about consecutive NAs by showing the most common NA gap sizes in the time series. The `plotNA.imputations` function is designated for visual inspection of the results after applying an imputation algorithm. Therefore, newly imputed observations are shown in a different color than the rest of the series.

Imputation functions

An overview about all available imputation algorithms can be found in Table 3. Even if these functions are really easy applicable, some examples can be found later in section [Usage examples](#). More detailed information about the theoretical background of the algorithms can be found in the **imputeTS** manual (Moritz, 2017b).

Function	Option	Description
na.interpolation	linear	Imputation by Linear Interpolation
	spline	Imputation by Spline Interpolation
	stine	Imputation by Stineman Interpolation
na.kalman	StructTS	Imputation by Structural Model & Kalman Smoothing
	auto.arima	Imputation by ARIMA State Space Representation & Kalman Sm.
na.locf	locf	Imputation by Last Observation Carried Forward
	nocb	Imputation by Next Observation Carried Backward
na.ma	simple	Missing Value Imputation by Simple Moving Average
	linear	Missing Value Imputation by Linear Weighted Moving Average
	exponential	Missing Value Imputation by Exponential Weighted Moving Average
na.mean	mean	Missing Value Imputation by Mean Value
	median	Missing Value Imputation by Median Value
	mode	Missing Value Imputation by Mode Value
na.random		Missing Value Imputation by Random Sample
na.replace		Replace Missing Values by a Defined Value
na.seadec		Seasonally Decomposed Missing Value Imputation
na.seasplit		Seasonally Splitted Missing Value Imputation
na.remove		Remove Missing Values

Table 3: Overview Imputation Algorithms

For convenience similar algorithms are available under one function name as parameter option. For example linear, spline and stineman interpolation are all included in the `na.interpolation` function. The `na.mean`, `na.locf`, `na.replace`, `na.random` functions are all simple and fast. In comparison, `na.interpolation`, `na.kalman`, `na.ma`, `na.seasplit`, `na.seadec` are more advanced algorithms that need more computation time. The `na.remove` function is a special case, since it only deletes all missing values. Thus, it is not really an imputation function. It should be handled with care since removing observations may corrupt the time information of the series. The `na.seasplit` and `na.seadec` functions are as well exceptions. These perform seasonal split / decomposition operations as a preprocessing step. For the imputation itself, one out of the other imputation algorithms can be used (which one can be set as option). Looking at all available imputation methods, no single overall best method can be pointed out. Imputation performance is always very dependent on the characteristics of the input time series. Even imputation with mean values can sometimes be an appropriate method. For time series with a strong seasonality usually `na.kalman` and `na.seadec` / `na.seasplit` perform best. In general, for most time series one algorithm out of `na.kalman`, `na.interpolation` and `na.seadec` will yield the best results. Meanwhile, `na.random`, `na.mean`, `na.locf` will be at the lower end accuracy wise for the majority of input time series.

Datasets

As can be seen in Table 4, all three datasets are available in a version with missing data and in a complete version. The provided time series are designated as benchmark datasets for univariate time series imputation. They shall enable users to quickly compare and test imputation algorithms. Without these datasets the process of testing time series imputation algorithms would require to manually delete certain observations. The benchmark data simplifies this: imputation algorithms can directly be applied to the dataset versions with missing values, which then can be compared to the complete

dataset versions afterwards. Since the time series are specified, researchers can use these to compare their algorithms against each other.

Reached RMSE or MAPE values on these datasets are easily understandable results to quote and compare against. Nevertheless, comparing algorithms using these fixed datasets can only be a first indicator of how well algorithms perform in general. Especially for the very short `tsAirgap` series (with just 13 NA values) random lucky guesses can considerably influence the results. A complete benchmark would include: 'Different missing data percentages', 'Different datasets', 'Different random seeds for missing data simulation'.

Overall there is a relatively small time series provided in `tsAirgap`, a medium one in `tsNH4` and a large time series in `tsHeating`. The `tsHeating` and `tsNH4` are both sensor data, while `tsAirgap` is count data.

Dataset	Description
<code>tsAirgap</code>	Time series of monthly airline passengers (with NAs)
<code>tsAirgapComplete</code>	Time series of monthly airline passengers (complete)
<code>tsHeating</code>	Time series of a heating systems' supply temperature (with NAs)
<code>tsHeatingComplete</code>	Time series of a heating systems' supply temperature (complete)
<code>tsNH4</code>	Time series of NH4 concentration in a waste-water system (with NAs)
<code>tsNH4Complete</code>	Time series of NH4 concentration in a waste-water system (complete)

Table 4: Overview Datasets

tsAirgap

The `tsAirgap` time series has 144 rows and the incomplete version includes 14 NA values. It represents the monthly totals of international airline passengers from 1949 to 1960. The time series originates from [Box et al. \(2015\)](#) and is a commonly used example in time series analysis literature. Originally known as 'AirPassengers' or 'airpass' this version is renamed to 'tsAirgap' in order to improve differentiation from the complete series (gap signifies that NAs were introduced). The characteristics (strong trend, strong seasonal behavior) make the `tsAirgap` series a great example for time series imputation.

As already mentioned in order to use this series for comparing imputation algorithm results, there are two time series provided. One series without missing values (`tsAirgapComplete`), which can be used as ground truth. Another series with NAs, on which the imputation algorithms can be applied (`tsAirgap`). While the missing data for `tsNH4` and `tsHeating` were each introduced according to patterns observed in very similar time series from the same source, the missing observations in `tsAirgap` were created based on general missing data patterns.

tsNH4

The `tsNH4` time series has 4552 rows and the incomplete version includes 883 NA values. It represents the NH4 concentration in a waste-water system measured from 30.11.2010 - 16:10 to 01.01.2011 - 6:40 in 10 minute steps. The time series is derived from the dataset of the Genetic and Evolutionary Computation Conference (GECCO) Industrial Challenge 2014 ¹.

As already mentioned in order to use this series for comparing imputation algorithm results, there are two time series provided. One series without missing values (`tsNH4Complete`), which can be used as ground truth. Another series with NAs (`tsNH4`), on which the imputation algorithms can be applied. The pattern for the NA occurrence was derived from the same series / sensors, but from an earlier time interval. Thus, it is a very realistic missing data pattern. Beware, since the time series has a lot of observations, some of the more complex algorithms like `na.kalman` will need some time till they are finished.

tsHeating

The `tsHeating` time series has 606837 rows and the incomplete version includes 57391 NA values. It represents a heating systems' supply temperature measured from 18.11.2013 - 05:12:00 to 13.01.2015 - 15:08:00 in 1 minute steps. The time series originates from the GECCO Industrial Challenge 2015 ². This was a challenge about 'Recovering missing information in heating system operating data'. Goal was to impute missing values in heating system sensor data as accurate as possible.

As already mentioned in order to use this series for comparing imputation algorithm results, there are two time series provided. One series without missing values (`tsHeatingComplete`), which can be used

¹<http://www.spotseven.de/gecco-challenge/gecco-challenge-2014/>

²<http://www.spotseven.de/gecco-challenge/gecco-challenge-2015/>

as ground truth. Another series with NAs (`tsHeating`), on which the imputation algorithms can be applied. The NAs thereby were inserted according to patterns found in similar time series. According to patterns found / occurring in other heating systems. Beware, since it is a very large time series, some of the more complex algorithms like `na.kalman` may need up to several days to complete on standard hardware.

Usage examples

To start working with the **imputeTS** package, install either the stable version from CRAN or the development version from GitHub (<https://github.com/SteffenMoritz/imputeTS>). The stable version from CRAN is hereby recommended.

Imputation algorithms

All imputation algorithms are used the same way. Input has to be either a numeric time series or a numeric vector. As output, a version of the input data with all missing values replaced by imputed values is returned. Here is a small example, to show how to use the imputation algorithms. (all imputation functions start with `na.`'algorithm name')

For this we first need to create an example input series with missing data.

```
# Create a short example time series with missing values
x <- ts(c(1, 2, 3, 4, 5, 6, 7, 8, NA, NA, 11, 12))
```

On this time series we can apply different imputation algorithms. We start with using `na.mean`, which substitutes the NAs with mean values.

```
# Impute the missing values with na.mean
na.mean(x)

[1] 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 5.9 5.9 11.0 12.0
```

Most of the functions also have additional options that provide further algorithms (of the same algorithm category). In the example below it can be seen that `na.mean` can also be called with `option="median"`, which substitutes the NAs with median values.

```
# Impute the missing values with na.mean using option median
na.mean(x, option="median")

[1] 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 5.5 5.5 11.0 12.0
```

While `na.interpolation` and all other imputation functions are used the same way, the results produced may be different. As can be seen below, for this series linear interpolation gives more reasonable results.

```
# Impute the missing values with na.interpolation
na.interpolation(x)

[1] 1 2 3 4 5 6 7 8 9 10 11 12
```

For longer and more complex time series (with trend and seasonality) than in this example it is always a good idea to try `na.kalman` and `na.seadec`, since these functions very often produce the best results. These functions are called the same easy way as all other imputation functions.

Here is a usage example for the `na.kalman` function applied on the `tsAirgap` (described in 28.2.4) time series. As can be seen in Figure 1, `na.kalman` provides really good results for this series, which contains a strong seasonality and a strong trend.

```
# Impute the missing values with na.kalman
# (tsAirgap is an example time series provided by the imputeTS package)
imp <- na.kalman(tsAirgap)

#Code for visualization
plotNA.imputations(tsAirgap, x.imp, tsAirgapComplete)
```

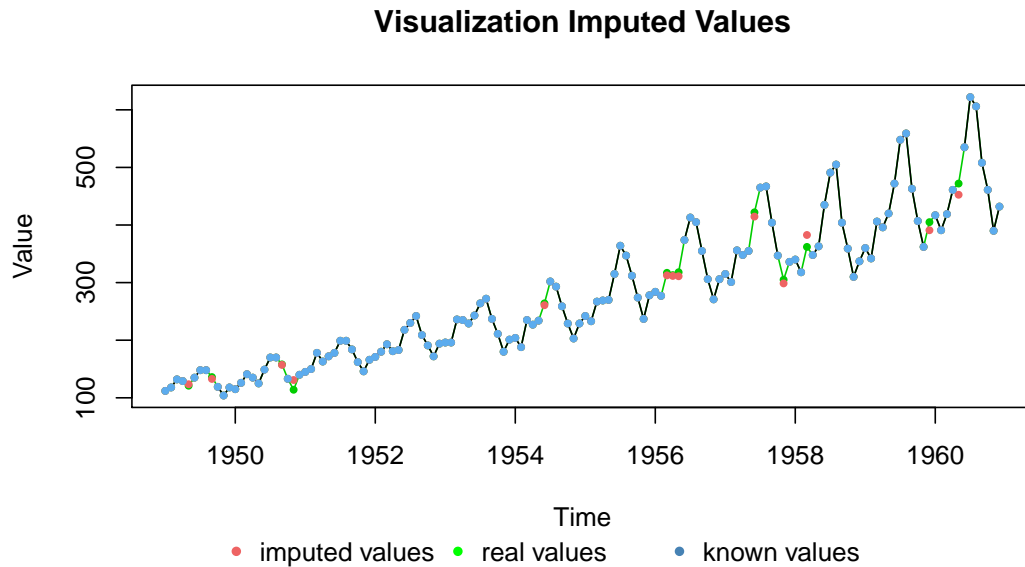


Figure 1: Results of imputation with `na.kalman` compared to real values

`plotNA.distribution`

This function visualizes the distribution of missing values within a time series. Therefore, the time series is plotted and whenever a value is NA the background is colored differently. This gives a nice overview, where in the time series most of the missing values occur. An example usage of the function can be seen below (for the plot see Figure 2).

```
# Example Code 'plotNA.distribution'
# (tsAirgap is an example time series provided by the imputeTS package)

# Visualize the missing values in this time series
plotNA.distribution(tsAirgap)
```

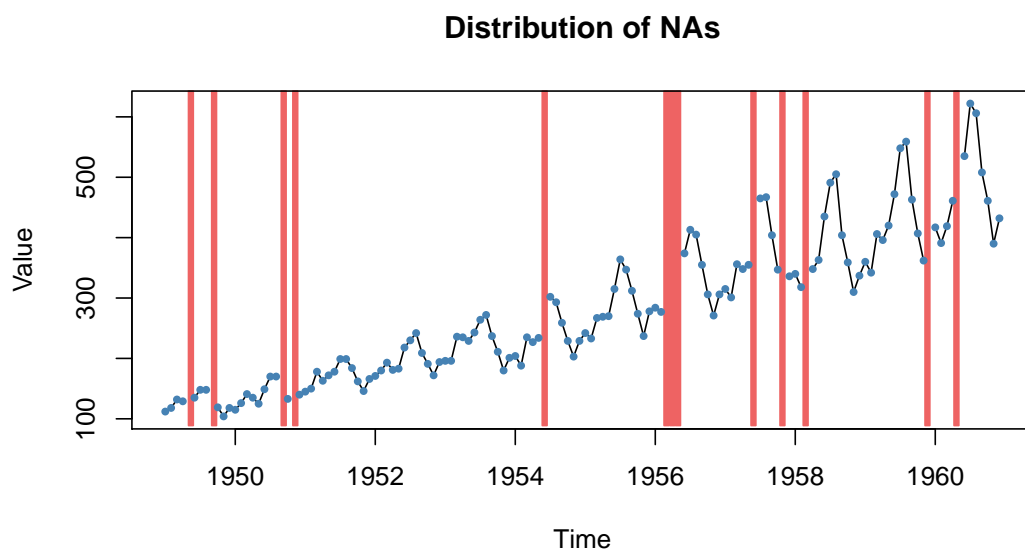


Figure 2: Example for `plotNA.distribution`

As can be seen in Figure 2, in areas with missing data the background is colored red. The whole plot is pretty much self-explanatory. The plotting function itself needs no further configuration parameters,

nevertheless it allows passing through of plot parameters (via ...).

plotNA.distributionBar

This function also visualizes the distribution of missing values within a time series. This is done as a barplot, which is especially useful if the time series would otherwise be too large to be plotted. Multiple observations for time intervals are grouped together and represented as bars. For these intervals, information about the amount of missing values are shown. An example usage of the function can be seen below (for the plot see Figure 3).

```
# Example Code 'plotNA.distributionBar'
# (tsHeating is an example time series provided by the imputeTS package)

# Visualize the missing values in this time series
plotNA.distributionBar(tsHeating, breaks = 20)
```

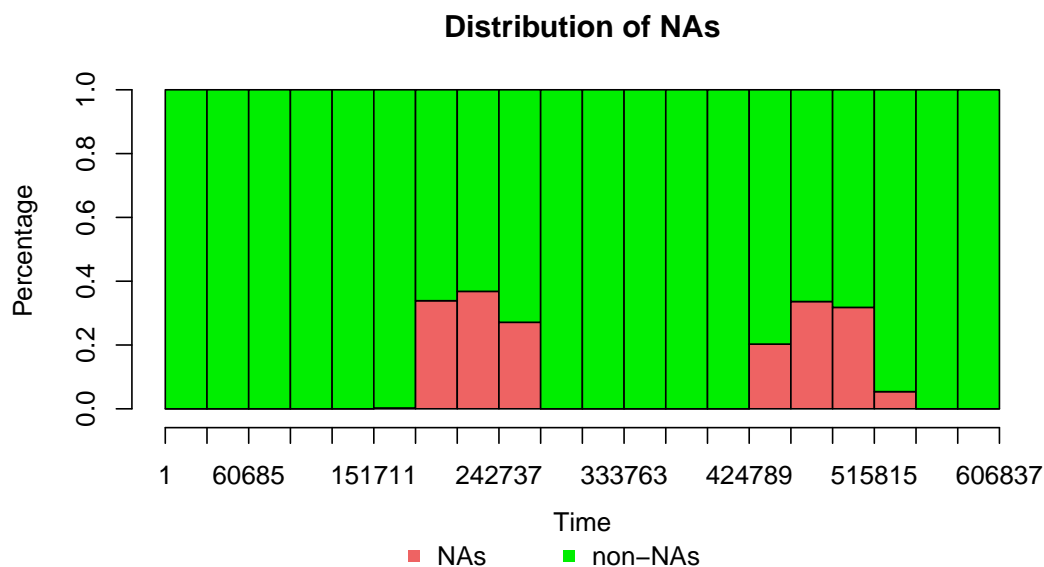


Figure 3: Example for plotNA.distributionBar

As can be seen in the x-axis of Figure 3, the `tsHeating` series is with over 600.000 observations a very large time series. While the missing values in the `tsAirgap` series (144 observations) can be visualized with `plotNA.distribution` like in Figure 2, this would for sure not work out for `tsHeating`. There just isn't enough space for 600.000 single consecutive observations/points in the plotting area. The `plotNA.distributionBar` function solves this problem. Multiple observations are grouped together in intervals. The 'breaks' parameter in the example defines that there should be 20 intervals used. This means every interval in Figure 3 represents approximately 30.000 observations. The first five intervals are completely green, which means there are no missing values present. This means from observation 1 up to observation 150.000 there are no missing values in the data. In the middle and at the end of the series there are several intervals each having around 40% of missing data. This means in these intervals 12.000 out of 30.000 observation are NA. All in all, the plot is able to give a nice but rough overview about the NA distribution in very large time series.

plotNA.gapsizes

This plotting function can be used to visualize how often different NA gaps (NAs in a row) occur in a time series. The function shows this information as a ranking. This ranking can be ordered by

total NAs gap sizes account for (number occurrence gap size * gap length) or just by the number of occurrences of gap sizes. In the end the results can be read like this: In time series x , 3 NAs in a row occur most often with 20 occurrences, 6 NAs in a row occur 2nd most with 5 occurrences, 2 NAs in a row occur 3rd most with 3 occurrences. An example usage of the function can be seen below (for the plot see Figure 4).

```
# Example Code 'plotNA.gapsizes'
# (tsNH4 is an example time series provided by the imputeTS package)

# Visualize the top gap sizes / NAs in a row
plotNA.gapsizes(tsNH4)
```

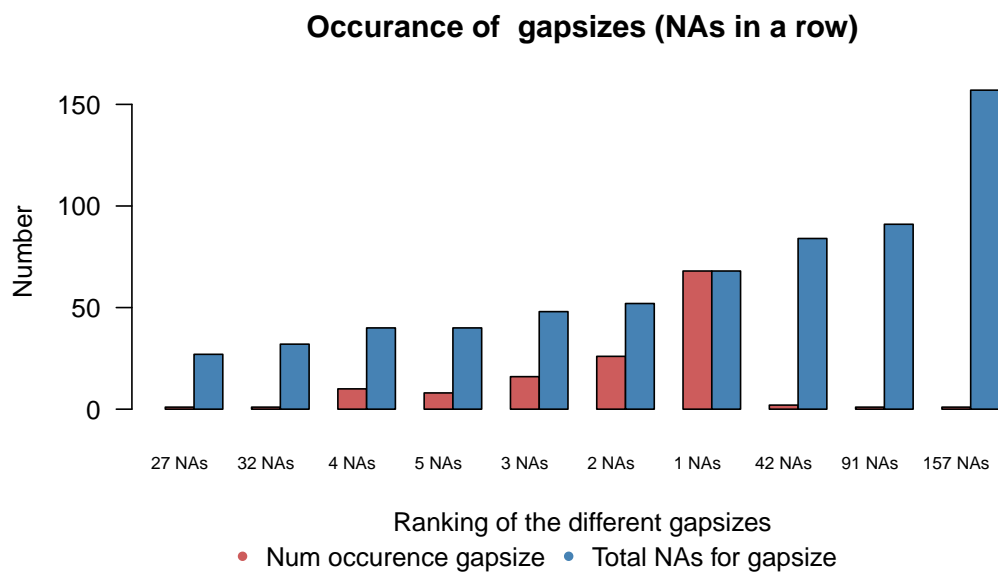


Figure 4: Example for printNA.gapsizes

The example plot (Figure 4) reads the following: In the time series $tsNH4$ gap size 157 occurs just 1 time, but makes up for most NAs of all gap sizes (157 NAs). A gap size of 91 (91 NAs in a row) also occurs just once, but makes up for 2nd most NAs (91 NAs). A gap size of 42 occurs two times in the time series, which leads to 3rd most overall (84 NAs). A gap size of one (no other NAs before or behind the NA) occurs 68 times, which makes this 4th in overall NAs (68 NAs).

plotNA.imputations

This plot can be used, to visualize the imputed values for a time series. Therefore, the imputed values (filled NA gaps) are shown in a different color than the other values. The function is used as below and Figure 5 shows the output.

```
# Example Code 'plotNA.imputations'
# (tsAirgap is an example time series provided by the imputeTS package)

# Step 1: Perform imputation for x using na.mean
tsAirgap.imp <- na.mean(tsAirgap)

# Step 2: Visualize the imputed values in the time series
plotNA.imputations(tsAirgap, tsAirgap.imp)
```

The visual inspection of Figure 5 indicates, that the imputed values (red) do not fit very well in the `tsAirgap` series. This is caused by `na.mean` being used for imputation of a series with a strong trend. The plotting function enables users to quickly detect such problems in the imputation results. If the ground truth is known for the imputed values, this information can also be added to the plot. The plotting function itself needs no further configuration parameters. Nevertheless, it allows passing through of plot parameters (via ...).

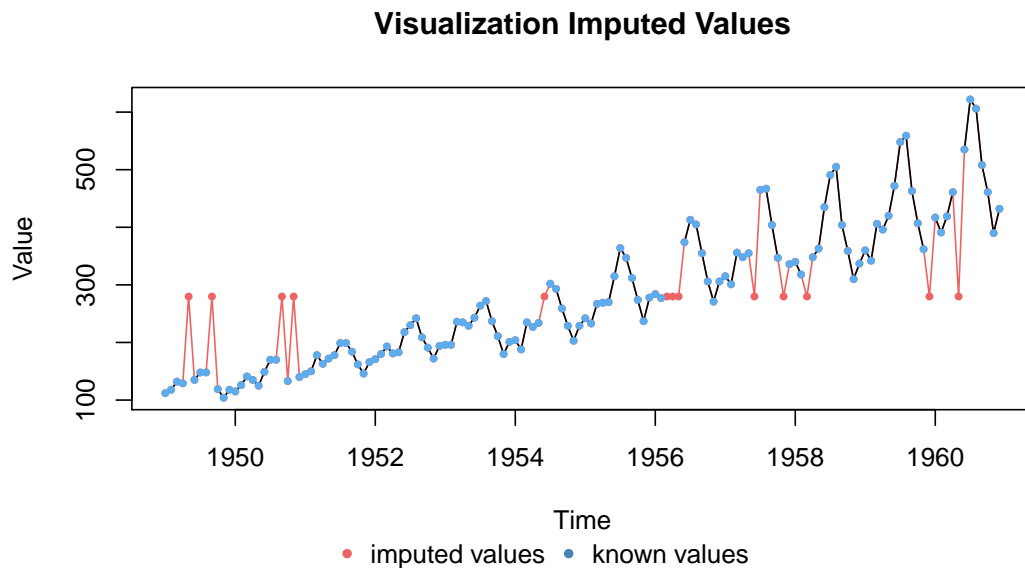


Figure 5: Example for `printNA.imputations`

statsNA

The `statsNA` function prints summary stats about the distribution of missing values in univariate time series. Here is a short explanation about the information it gives:

- Length of time series
Number of observations in the time series (including NAs)
- Number of Missing Values
Number of missing values in the time series
- Percentage of Missing Values
Percentage of missing values in the time series
- Stats for Bins
Number/percentage of missing values for the split into bins
- Longest NA gap
Longest series of consecutive missing values (NAs in a row) in the time series
- Most frequent gap size
Most frequent occurring series of missing values in the time series
- Gap size accounting for most NAs
The series of consecutive missing values that accounts for most missing values overall in the time series
- Overview NA series
Overview about how often each series of consecutive missing values occurs. Series occurring 0 times are skipped

The function is used as below and Figure 6 shows the output.

```
# Example Code 'statsNA'
# (tsNH4 is an example time series provided by the imputeTS package)

# Print stats about the missing data
statsNA(tsNH4)
```

```

> statsNA(x)
[1] "Length of time series:"
[1] 4552
[1] "-----"
[1] "Number of Missing Values:"
[1] 883
[1] "-----"
[1] "Percentage of Missing Values:"
[1] "19.4%"
[1] "-----"
[1] "Stats for Bins"
[1] " Bin 1 (1138 values from 1 to 1138) :      233 NAs (20.5%)"
[1] " Bin 2 (1138 values from 1139 to 2276) :      433 NAs (38%)"
[1] " Bin 3 (1138 values from 2277 to 3414) :      135 NAs (11.9%)"
[1] " Bin 4 (1138 values from 3415 to 4552) :       82 NAs (7.21%)"
[1] "-----"
[1] "Longest NA gap (series of consecutive NAs)"
[1] "157 in a row"
[1] "-----"
[1] "Most frequent gap size (series of consecutive NA series)"
[1] "1 NA in a row (occurring 68 times)"
[1] "-----"
[1] "Gap size accounting for most NAs"
[1] "157 NA in a row (occurring 1 times, making up for overall 157 NAs)"
[1] "-----"
[1] "Overview NA series"
[1] " 1 NA in a row: 68 times"
[1] " 2 NA in a row: 26 times"
[1] " 3 NA in a row: 16 times"
[1] " 4 NA in a row: 10 times"
[1] " 5 NA in a row: 8 times"
[1] " 6 NA in a row: 4 times"
[1] " 7 NA in a row: 2 times"

```

Figure 6: Excerpt of statsNA output

Datasets

Using the datasets is self-explanatory, after the package is loaded they are directly available and usable under their name. No call of `data()` is needed. For every dataset there is always a complete version (without NAs) and an incomplete version (containing NAs) available.

```

# Example Code to use tsAirgap dataset
library("imputeTS")
tsAirgap

```

```

> tsAirgap
      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129  NA 135 148 148  NA 119 104 118
1950 115 126 141 135 125 149 170 170  NA 133  NA 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234  NA 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277  NA  NA  NA 374 413 405 355 306 271 306
1957 315 301 356 348 355  NA 465 467 404 347  NA 336
1958 340 318  NA 348 363 435 491 505 404 359 310 337
1959 360 342 406 396 420 472 548 559 463 407 362  NA
1960 417 391 419 461  NA 535 622 606 508 461 390 432

```

Figure 7: Example tsAirgap time series

Conclusions

Missing data is a very common problem for all kinds of data. However, in case of univariate time series most standard algorithms and existing functions within R packages cannot be applied. This paper presented the **imputeTS** package that provides a collection of algorithms and tools especially tailored to this task. Using example time series, we illustrated the ease of use and the advantages of the provided functions. Simple algorithms as well as more complicated ones can be applied in the same simple and user-friendly manner.

The functionality provided makes the **imputeTS** package a good choice for preprocessing of time series ahead of further analysis steps that require complete absence of missing values.

Future research and development plans for forthcoming versions of the package include adding additional time series algorithm options to choose from.

Acknowledgment

Parts of this work have been developed in the project 'IMProVt: Intelligente Messverfahren zur Prozessoptimierung von Trinkwasserbereitstellung und -verteilung' (reference number: 03ET1387A). Kindly supported by the Federal Ministry of Economic Affairs and Energy of the Federal Republic of Germany.

Supported by:



Federal Ministry
for Economic Affairs
and Energy

on the basis of a decision
by the German Bundestag

Bibliography

- Z. Bar-Joseph, G. K. Gerber, D. K. Gifford, T. S. Jaakkola, and I. Simon. Continuous representations of time-series gene expression data. *Journal of Computational Biology*, 10(3-4):341–356, 2003. URL <https://doi.org/10.1089/10665270360688057>. [p207]
- R. Billinton, H. Chen, and R. Ghajar. Time-series models for reliability evaluation of power systems including wind energy. *Microelectronics Reliability*, 36(9):1253–1261, 1996. URL [https://doi.org/10.1016/0026-2714\(95\)00154-9](https://doi.org/10.1016/0026-2714(95)00154-9). [p207]
- G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 2015. [p210]
- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977. [p207]
- B. L. Ford. An Overview of Hot-Deck Procedures. *Incomplete data in sample surveys*, 2(Part IV):185–207, 1983. [p207]
- J. M. Gottman. *Time-Series Analysis: A Comprehensive Introduction for Social Scientists*, volume 400. Cambridge University Press Cambridge, 1981. [p207]
- J. Honaker, G. King, and M. Blackwell. Amelia II: A program for missing data. *Journal of Statistical Software*, 45(7):1–47, 2011. URL <https://doi.org/10.18637/jss.v045.i07>. [p207]
- R. J. Hyndman. *forecast: Forecasting Functions for Time Series and Linear Models*, 2017. URL <http://github.com/robjhyndman/forecast>. R package version 8.0. [p207]

- J. Josse and F. Husson. *missMDA: A package for handling missing values in multivariate data analysis*. *Journal of Statistical Software*, 70(1):1–31, 2016. URL <https://doi.org/10.18637/jss.v070.i01>. [p207]
- A. Kowarik and M. Templ. Imputation with the R package VIM. *Journal of Statistical Software*, 74(7): 1–16, 2016. URL <https://doi.org/10.18637/jss.v074.i07>. [p207]
- S. Moritz. *imputeTS: Time Series Missing Value Imputation*, 2017a. URL <http://CRAN.R-project.org/package=imputeTS>. R package version 2.3. [p207]
- S. Moritz. *Package imputeTS*, 2017b. URL <http://cran.r-project.org/web/packages/imputeTS/imputeTS.pdf>. R package version 2.3. [p209]
- S. Moritz, A. Sard á, T. Bartz-Beielstein, M. Zaefferer, and J. Stork. Comparison of different methods for univariate time series imputation in R. *ArXiv e-prints*, 2015. [p207]
- E. Pebesma. *spacetime: Spatio-temporal data in R*. *Journal of Statistical Software*, 51(7):1–30, 2012. URL <https://doi.org/10.18637/jss.v051.i07>. [p207]
- Rmetrics Core Team, D. Wuertz, T. Setz, and Y. Chalabi. *timeSeries: Rmetrics - Financial Time Series Objects*, 2015. URL <https://CRAN.R-project.org/package=timeSeries>. R package version 3022.101.2. [p207]
- D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. John Wiley & Sons, 1987. URL <https://doi.org/10.1002/9780470316696>. [p207]
- J. A. Ryan and J. M. Ulrich. *xts: eXtensible Time Series*, 2014. URL <https://CRAN.R-project.org/package=xts>. R package version 0.9-7. [p207]
- S. J. Taylor. *Modelling Financial Time Series (Second Edition)*. World Scientific Publishing, 2007. URL <https://doi.org/10.1142/9789812770851>. [p207]
- P. Vacek and T. Ashikaga. An examination of the nearest neighbor rule for imputing missing values. *Proc. Statist. Computing Sect., Amer. Statist. Ass*, pages 326–331, 1980. [p207]
- S. van Buuren and K. Groothuis-Oudshoorn. *mice: Multivariate imputation by chained equations in R*. *Journal of Statistical Software*, 45(3):1–67, 2011. URL <https://doi.org/10.18637/jss.v045.i03>. [p207]
- A. Zeileis and G. Grothendieck. *zoo: S3 infrastructure for regular and irregular time series*. *Journal of Statistical Software*, 14(6):1–27, 2005. URL <https://doi.org/10.18637/jss.v014.i06>. [p207]

Steffen Moritz
Cologne University of Applied Sciences
Cologne, Germany
steffen.moritz10@gmail.com

Thomas Bartz-Beielstein
Cologne University of Applied Sciences
Cologne, Germany
bartz.beielstein@fh-koeln.de

The NoiseFiltersR Package: Label Noise Preprocessing in R

by Pablo Morales, Julián Luengo, Luís P.F. Garcia, Ana C. Lorena, André C.P.L.F. de Carvalho and Francisco Herrera

Abstract In Data Mining, the value of extracted knowledge is directly related to the quality of the used data. This makes data preprocessing one of the most important steps in the knowledge discovery process. A common problem affecting data quality is the presence of noise. A training set with label noise can reduce the predictive performance of classification learning techniques and increase the overfitting of classification models. In this work we present the **NoiseFiltersR** package. It contains the first extensive R implementation of classical and state-of-the-art label noise filters, which are the most common techniques for preprocessing label noise. The algorithms used for the implementation of the label noise filters are appropriately documented and referenced. They can be called in a R-user-friendly manner, and their results are unified by means of the "filter" class, which also benefits from adapted print and summary methods.

Introduction

In the last years, the large quantity of data of many different kinds and from different sources has created numerous challenges in the Data Mining area. Not only their size, but their imperfections and varied formats are providing the researchers with plenty of new scenarios to be addressed. Consequently, Data Preprocessing (García et al., 2015) has become an important part of the *KDD* (*Knowledge Discovery from Databases*) process, and related software development is also essential to provide practitioners with the adequate tools.

Data Preprocessing intends to process the collected data appropriately so that subsequent learning algorithms can not only extract meaningful and relevant knowledge from the data, but also induce models with high predictive or descriptive performance. Data preprocessing is known as one of the most time-consuming steps in the whole KDD process. There exist several aspects involved in data preprocessing, like *feature selection*, dealing with *missing values* and detecting *noisy data*. Feature selection aims at extracting the most relevant attributes for the learning step, thus reducing the complexity of models and the computing time taken for their induction. The treatment of missing values is also essential to keep as much information as possible in the preprocessed dataset. Finally, noisy data refers to values that are either incorrect or clearly far from the general underlying data distribution.

All these tasks have associated software available. For instance, the KEEL tool (Alcalá et al., 2010) contains a broad collection of data preprocessing algorithms, which covers all the aforementioned topics. There exist many other general-purpose Data Mining software with data preprocessing functionalities, like WEKA (Witten and Frank, 2005), KNIME (Berthold et al., 2009), RapidMiner (Hofmann and Klinkenberg, 2013) or R.

Regarding the R statistical software, there are plenty of packages available in the *Comprehensive R Archive Network* (CRAN) repository to address preprocessing tasks. For example, **MICE** (van Buuren and Groothuis-Oudshoorn, 2011) and **Amelia** (Honaker et al., 2011) are very popular packages for handling missing values, whereas **caret** (Kuhn, 2008) or **FSelector** (Romanski and Kotthoff, 2014) provide a wide range of techniques for feature selection. There are also general-purpose packages for detecting outliers and anomalies, like **mvoutlier** (Filzmoser and Gschwandtner, 2015). If we examine software in CRAN developed to tackle label noise, there already exist *non-preprocessing* packages that provide label noise robust classifiers. For instance, **robustDA** implements a robust mixture discriminant analysis Bouveyron and Girard (2009), while **probFDA** package provides a probabilistic Fisher discriminant analysis related to the seminal work in Lawrence and Schölkopf (2001).

However, to the best of our knowledge, CRAN lacks an extensive collection of label noise preprocessing algorithms for classification (Sáez et al., 2016; Garcia et al., 2015), some of which are among the most influential preprocessing techniques (García et al., 2016). This is the gap we intend to fill with the release of the **NoiseFiltersR** package, whose taxonomy is inspired on the recent survey on label noise by B. Frénay and M. Verleysen (Frénay and Verleysen, 2014). Yet, it should be noted that there are other packages that include some isolated implementations of label noise filters, since they are sometimes needed as auxiliary functions. This is the case of the **unbalanced** (Pozzolo et al., 2015) package, which deals with imbalanced classification. It contains basic versions of classical filters, such as *Tomek-Links* (Tomek, 1976) or *ENN* (Wilson, 1972), which are typically applied after oversampling an imbalanced dataset (which is the main purpose of the **unbalanced** package).

In the following section we briefly introduce the problem of classification with label noise, as well as the most popular techniques to overcome this problem. Then, we show how to use the **NoiseFiltersR** package to apply these techniques in a unified and R-user-friendly manner. Finally, we present a general overview of this work and potential extensions.

Label noise preprocessing

Data collection and preparation processes are usually subject to errors in Data Mining applications (Wu and Zhu, 2008). Consequently, real-world datasets are commonly affected by imperfections or *noise*. In a classification problem, several effects of this noise can be observed by analyzing its spatial characteristics: noise may create small clusters of instances of a particular class in the instance space corresponding to another class, displace or remove instances located in key areas within a concrete class, or disrupt the boundaries of the classes resulting in an increased boundaries overlap. All these imperfections may harm interpretation of data, the design, size, building time, interpretability and accuracy of models, as well as the making of decisions (Zhong et al., 2004; Zhu and Wu, 2004).

In order to alleviate the effects of noise, we need first to identify and quantify the components of the data that can be affected. As described by Wang et al. (1995), from the large number of components that comprise a dataset, class labels and attribute values are two essential elements in classification datasets (Wu, 1996). Thus, two types of noise are commonly differentiated in the literature (Zhu and Wu, 2004; Wu, 1996):

- *Label noise*, also known as *class noise*, is when an example is wrongly labeled. Several causes may induce label noise, including subjectivity during the labeling process, data entry errors, or inadequacy of the information used to label each instance. Label noise includes contradictory examples (Hernández and Stolfo, 1998) (examples with identical input attribute values having different class labels) and misclassifications (examples which are incorrectly labeled Zhu and Wu, 2004). Since detecting contradictory examples is easier than identifying misclassifications (Zhu and Wu, 2004), most of the literature is focused on the study of misclassifications, and the term *label noise* usually refers to this type of noise (Teng, 1999; Sáez et al., 2014).
- *Attribute noise* refers to corruptions in the values of the input attributes. It includes erroneous attribute values, missing values and incomplete attributes or “do not care” values. Missing values are usually considered independently in the literature, so *attribute noise* is mainly used for erroneous values (Zhu and Wu, 2004).

The **NoiseFiltersR** package (and the rest of this manuscript) focuses on *label noise*, which is known to be the most disruptive one, since label quality is essential for the classifier training (Zhu and Wu, 2004). In Fréney and Verleysen (2014) the mechanisms that generate label noise are examined, relating them to the appropriate treatment procedures that can be safely applied. In the specialized literature there exist two main approaches to deal with label noise, and both are surveyed in Fréney and Verleysen (2014):

- On the one hand, *algorithm level* approaches attempt to create robust classification algorithms that are little influenced by the presence of noise. This includes approaches where existing algorithms are modified to cope with label noise by either modeling it in the classifier construction (Lawrence and Schölkopf, 2001; Li et al., 2007), by applying pruning strategies to avoid overfitting as in Quinlan (1993) or by diminishing the importance of noisy instances with respect to clean ones (Miao et al., 2016). There exist recent proposals that combine these two approaches, which model the noise and give less relevance to potentially noisy instances in the classifier building process (Bouveyron and Girard, 2009).
- On the other hand, *data level* approaches (also called *filters*) try to develop strategies to cleanse the dataset as a previous step to the fitting of the classifier, by either creating ensembles of classifiers (Brodley and Friedl, 1999), iteratively filtering noisy instances (Khoshgoftaar and Rebours, 2007), computing metrics on the data or even hybrid approaches that combine several of these strategies.

The **NoiseFiltersR** package follows the data level approaches, since this allows the data preprocessing to be carried out just once, and apply any classifier thereafter, whereas algorithm level approaches are specific for each classification algorithm¹. Regarding data-level handling of label noise, we take the aforementioned survey by Fréney and Verleysen (2014) as the basis for our **NoiseFiltersR** package.

¹Of course, in R there exist implementations of very popular label noise robust classifiers (the aforementioned algorithm-level approach), such as *C4.5* and *RIPPER*, which are called *J48* and *JRip* respectively in the **RWeka** package (Hornik et al., 2009), which is a R interface to WEKA software (Witten and Frank, 2005), or the method described in Bouveyron and Girard (2009) in its own package.

That work provides an overview and references for the most popular classical and state-of-the-art filters, which are organized and classified taking into account several aspects:

- Considering how noisy instances are identified, *ensemble* based, *similarity* based and *data complexity* based algorithms are distinguished. The first type makes use of predictions from ensembles of classifiers built over different partitions or resamples of training data. The second is based on label distribution from the nearest neighbors of each instance. And the third attempts to reduce complexity metrics which are related to the presence of noise. As we will explain in the next section (see Table 1), the **NoiseFiltersR** package contains implementations of all these types of algorithms, and the explicit distinction is indicated in the documentation page of each function.
- Regarding how to deal with the identified noise, *noise removal* and *noise reparation* strategies are considered. The first option removes the noisy instances, whereas the second relabels these instances with the most likely label on the basis of the available information. There also exist *hybrid* approaches, which only carry out relabelling when they have enough confidence on the new label. Otherwise, they remove the noisy instance. The discussion between noise removal, noise reparation and their possible synergies is an active and open field of research (Fréney and Verleysen, 2014, Section VI.H): most works agree on the potential damages of incorrect relabeling (Miranda et al., 2009), although other studies also point out the dangers of removing too many instances and advocate hybrid approaches (Teng, 2005). As we will see in the next section, the **NoiseFiltersR** package includes filters which implement all these possibilities, and the specific behaviour is explicitly indicated in the documentation page of the corresponding function.

The NoiseFiltersR package

The released package implements, documents, explains and provides references for a broad collection of label noise filters surveyed in (Fréney and Verleysen, 2014). To the best of our knowledge, it is the first comprehensive review and implementation of this topic for R, which has become an essential tool in Data Mining in the last years.

Namely, the **NoiseFiltersR** package includes a total of 30 filters, which were published in 24 research papers. Each one of these papers is referenced in the corresponding filter documentation page, as shown in the next *Documentation* section (and particularly in Figure 1). Regarding the noise detection strategy, 13 of them are ensemble based filters, 14 can be cataloged as similarity based, and the other 3 are based on data complexity measures. Taking into account the noise handling approach, 4 of them integrate the possibility of relabelling, whereas the other 26 only allow for removing (which clearly evidences a general preference for data removal in the literature). The full list of implemented filters and its distribution according to the two aforementioned criterions is displayed in Table 1, which provides a general overview of the package.

		Noise Identification		
		Ensemble	Similarity	Data Complexity
Noise Handling	Remove	C45robustFilter C45votingFilter C45iteratedVotingFilter CVCF dynamicCF edgeBoostFilter EF HARF INFFC IPF ORBoostFilter PF	AENN BBNR CNN DROP1 DROP2 DROP3 ENG ENN PRISM RNN TomekLinks	saturationFilter consensusSF classifSF
	Repair/ Hybrid	hybridRepairFilter	EWF GE ModeFilter	

Table 1: Names and taxonomy of available filters in the NoiseFiltersR package. Every filter is appropriately referenced in its documentation page, where the original paper is provided.

The rest of this section is organized as follows. First, a few lines are devoted to the installation process. Then, we present the documentation page for the filters, where further specific details can be

looked up. After that, we focus on the two implemented methods to call the filters (*default* and *formula*). Finally, the "filter" class, which unifies the return value of the filters in **NoiseFiltersR** package, is presented.

Installation

The **NoiseFiltersR** package is available at CRAN servers, so it can be downloaded and installed directly from the R command line by typing:

```
> install.packages("NoiseFiltersR")
```

In order to easily access all the package's functions, it must be attached in the usual way:

```
> library(NoiseFiltersR)
```

Documentation

Whereas this paper provides the user with an overview of the **NoiseFiltersR** package, it is also important to have access to specific information for each available filter. This information can be looked up in the corresponding documentation page, that in all cases includes the following essential items (see Figure 1 for an example):

- A *description* section, which indicates the type of filter according to the taxonomy explained in Table 1.
- A *details* section, which provides the user with a general explanation of the filter's behaviour and any other usage particularity or warning.
- A *references* section that points to the original contribution where the filter was proposed, where further details, motivations or contextualization can be found.

GE {NoiseFiltersR}	R Documentation
Generalized Edition	
Description	
Similarity-based filter for removing or repairing label noise from a dataset as a preprocessing step of classification. For more information, see 'Details' and 'References' sections.	
Details	
GE is a generalization of ENN that integrates the possibility of 'repairing' or 'relabeling' instances rather than only 'removing'. For each instance, GE considers its $k-1$ neighbors and the instance itself. If there are at least k examples from the same class, the instance is relabeled with that class (which could be its own). Otherwise, it is removed.	
References	
Koplowitz J., Brown T. A. (1981): On the relation of performance to editing in nearest neighbor rules. <i>Pattern Recognition</i> , 13(3), 251-255.	

Figure 1: Extract from GE filter's documentation page, showing the highlighted above aspects.

As usually in R, the function documentation pages can be either checked in the CRAN website for the package or loaded from the command line with the orders `?` or `help()`:

```
> ?GE
> help(GE)
```

Calling the filters

When one wants to use a label noise filter in Data Mining applications, all we need to know is the dataset to be filtered and its *class* variable (i.e. the one that contains the label for each available instance). The **NoiseFiltersR** package provides two standard ways for tagging the class variable when calling the implemented filters (see also Figure 2 and the example below):

- The *default* method receives the dataset to be filtered in the `x` argument, and the number for the class column through the `classColumn` argument. If the latter is not provided, the last column of the dataset is assumed to contain the labels.
- The *formula* method is intended for regular R users, who are used to this approach when fitting regression or classification models. It allows for indicating the class variable (along with the attributes to be used) by means of an expression like `Class~Attr1+...+AttrN` (recall that `Class~.` makes use of all attributes).

Next, we provide an example on how to use these two methods for filtering out the `iris` dataset with `edgeBoostFilter` (we did not change the default parameters of the filter):

```
# Checking the structure of the dataset (last variable is the class one)
> data(iris)
> str(iris)
'data.frame':      150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 ...

# Using the default method:
> out_Def <- edgeBoostFilter(iris, classColumn = 5)
# Using the formula method:
> out_For <- edgeBoostFilter(Species~., iris)
# Checking that the filtered datasets are identical:
> identical(out_Def$cleanData, out_For$cleanData)
[1] TRUE
```

edgeBoostFilter {NoiseFiltersR}
R Documentation

Edge Boosting Filter

Usage

```
## S3 method for class 'formula'
edgeBoostFilter(formula, data, ...)

## Default S3 method:
edgeBoostFilter(x, m = 15, percent = 0.05,
               threshold = 0, classColumn = ncol(x), ...)
```

Arguments

<code>formula</code>	A formula describing the classification variable and the attributes to be used.
<code>data, x</code>	Data frame containing the training dataset to be filtered.
<code>...</code>	Optional parameters to be passed to other methods.
<code>m</code>	Number of boosting iterations
<code>percent</code>	Real number between 0 and 1. It sets the percentage of instances to be removed (as long as their edge value exceeds the parameter <code>threshold</code>).
<code>threshold</code>	Real number between 0 and 1. It sets the minimum edge value required by an instance in order to be removed.
<code>classColumn</code>	Positive integer indicating the column which contains the (factor of) classes. By default, the last column is considered.

Figure 2: Extract from `edgeBoostFilter`'s documentation page, which shows the two methods for calling filters in **NoiseFiltersR** package. In both cases, the parameters of the filter can be tuned through additional arguments.

Notice that, in the last command of the example, we used the `$` operator to access the objects returned from the filter. In next section we explore the structure and contents of these objects.

The "filter" class

The S3 class "filter" is designed to unify the return value of the filters inside the **NoiseFiltersR** package. It is a list that encapsulates seven elements with the most relevant information of the process:

- `cleanData` is a `data.frame` containing the filtered dataset.
- `remIdx` is a vector of integers indicating the indexes of removed instances (i.e. their row number with respect to the original `data.frame`).
- `repIdx` is a vector of integers indicating the indexes of repaired/relabelled instances (i.e. their row number with respect to the original `data.frame`).
- `repLab` is a factor containing the new labels for repaired instances.
- `parameters` is a list that includes the adopted parameters for the filter.
- `call` is an expression that contains the original call to the filter.
- `extraInf` is a character vector including additional information not covered by previous items.

As an example, we can check the structure of the above `out_For` object, which was the return value of `edgeBoostFilter` function:

```
> str(out_For)
List of 7
 $ cleanData :'data.frame':      142 obs. of  5 variables:
  ..$ Sepal.Length: num [1:142] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
  ..$ Sepal.Width : num [1:142] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
  ..$ Petal.Length: num [1:142] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
  ..$ Petal.Width : num [1:142] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
  ..$ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 ...
 $ remIdx       : int [1:8] 58 78 84 107 120 130 134 139
 $ repIdx       : NULL
 $ repLab       : NULL
 $ parameters:List of 3
  ..$ m         : num 15
  ..$ percent   : num 0.05
  ..$ threshold : num 0
 $ call        : language edgeBoostFilter(formula = Species ~ ., data = iris)
 $ extraInf    : chr "Highest edge value kept: 0.0669358381115568"
 - attr(*, "class")= chr "filter"
```

In order to cleanly display this "filter" class in the R console, two specific print and summary methods were implemented. The appearance of the first one is as follows

```
> print(out_For)

Call:
edgeBoostFilter(formula = Species ~ ., data = iris)

Parameters:
m: 15
percent: 0.05
threshold: 0

Results:
Number of removed instances: 8 (5.333333 %)
Number of repaired instances: 0 (0 %)
```

and contains three main blocks:

- The original *call* to the filter.
- The *parameters* used for the filter.
- An overview of the *results*, with the absolute number (and percentage of the total) of removed and repaired instances.

The summary method displays some extra blocks:

- It always adds a title that summarizes the filter and dataset used.
- If there exists additional information in the `extraInf` component of the object, it is displayed under a homonymous block.
- If the argument `explicit` is set to `TRUE` (it defaults to `FALSE`), the explicit results (i.e. the indexes for removed and repaired instances and the new labels for the latter) are displayed.

In the case of the previous `out_For` object, the `summary` command gets the following format:

```
> summary(out_For, explicit = TRUE)

Filter edgeBoostFilter applied to dataset iris

Call:
edgeBoostFilter(formula = Species ~ ., data = iris)

Parameters:
m: 15
percent: 0.05
threshold: 0

Results:
Number of removed instances: 8 (5.333333 %)
Number of repaired instances: 0 (0 %)

Additional information:
Highest edge value kept: 0.0669358381115568

Explicit indexes for removed instances:
58 78 84 107 120 130 134 139
```

Summary

In this paper, we introduced the **NoiseFiltersR** package, which is the first R extensive implementation of classification-oriented label-noise filters. To set a context and motivation for this work, we presented the problem of label noise and the main approaches to deal with it inside data preprocessing, as well as the related software. As previously explained, the released package unifies the return value of the filters by means of the "filter" class, which benefits from specific print and summary methods. Moreover, it provides a R-user-friendly way to call the implemented filters, whose documentation is worth reading and points to the original reference where they were first published.

Regarding the potential extensions of this package, there exist several aspects which can be addressed in future releases. For instance, there exist some other label noise filters reviewed in the main reference (Frénay and Verleysen, 2014) whose noise identification strategy does not belong to the ones covered here: ensemble based, similarity based and data complexity based (as shown in Table 1). Other relevant extension would be the inclusion of some datasets with different levels of artificially introduced label noise, in order to ease the experimentation workflow².

Acknowledgements

This work was supported by the Spanish Research Project TIN2014-57251-P, the Andalusian Research Plan P11-TIC-7765, and the Brazilian grants CeMEAI-FAPESP 2013/07375-0 and FAPESP 2012/22608-8. Luís P. F. Garcia was supported by FAPESP 2011/14602-7.

Bibliography

- J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2010. [p219]
- M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel. KNIME - the Konstanz information miner: Version 2.0 and beyond. *SIGKDD Explorations Newsletter*, 11(1):26–31, 2009. URL <https://doi.org/10.1145/1656274.1656280>. [p219]
- C. Bouveyron and S. Girard. Robust supervised classification with mixture models: Learning from data with uncertain labels. *Pattern Recognition*, 42(11):2649–2658, 2009. URL <https://doi.org/10.1016/j.patcog.2009.03.027>. [p219, 220]

²A wide variety of such datasets can be downloaded from the KEEL dataset repository in the website <http://www.keel.es/>, and then loaded from R.

- C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999. URL <https://doi.org/10.1613/jair.606>. [p220]
- P. Filzmoser and M. Gschwandtner. *Mvoutlier: Multivariate Outlier Detection Based on Robust Methods*, 2015. URL <https://CRAN.R-project.org/package=mvoutlier>. R package version 2.0.6. [p219]
- B. Frénay and M. Verleysen. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869, 2014. URL <https://doi.org/10.1109/TNNLS.2013.2292894>. [p219, 220, 221, 225]
- L. P. Garcia, J. A. Sáez, J. Luengo, A. C. Lorena, A. C. de Carvalho, and F. Herrera. Using the One-vs-One decomposition to improve the performance of class noise filters via an aggregation strategy in multi-class classification problems. *Knowledge-Based Systems*, 90:153–164, 2015. URL <https://doi.org/10.1016/j.knsys.2015.09.023>. [p219]
- S. García, J. Luengo, and F. Herrera. *Data Preprocessing in Data Mining*. Springer-Verlag, 2015. URL <https://doi.org/10.1007/978-3-319-10247-4>. [p219]
- S. García, J. Luengo, and F. Herrera. Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. *Knowledge-Based Systems*, 98:1–29, 2016. URL <https://doi.org/10.1016/j.knsys.2015.12.006>. [p219]
- M. A. Hernández and S. J. Stolfo. Real-World Data Is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2:9–37, 1998. URL <https://doi.org/10.1023/A:1009761603038>. [p220]
- M. Hofmann and R. Klinkenberg. *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. Chapman & Hall/CRC, 2013. ISBN 1482205491, 9781482205497. [p219]
- J. Honaker, G. King, and M. Blackwell. Amelia II: A program for missing data. *Journal of Statistical Software*, 45(7):1–47, 2011. URL <https://doi.org/10.18637/jss.v045.i07>. [p219]
- K. Hornik, C. Buchta, and A. Zeileis. Open-source machine learning: R meets Weka. *Computational Statistics*, 24(2):225–232, 2009. URL <https://doi.org/10.1007/s00180-008-0119-7>. [p220]
- T. M. Khoshgoftaar and P. Rebour. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, 22(3):387–396, 2007. URL <https://doi.org/10.1007/s11390-007-9054-2>. [p220]
- M. Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5), 2008. URL <https://doi.org/10.18637/jss.v028.i05>. [p219]
- N. D. Lawrence and B. Schölkopf. Estimating a kernel Fisher discriminant in the presence of label noise. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 306–313, 2001. [p219, 220]
- Y. Li, L. F. A. Wessels, D. de Ridder, and M. J. T. Reinders. Classification in the presence of class noise using a probabilistic kernel Fisher method. *Pattern Recognition*, 40(12):3349–3357, 2007. URL <https://doi.org/10.1016/j.patcog.2007.05.006>. [p220]
- Q. Miao, Y. Cao, G. Xia, M. Gong, J. Liu, and J. Song. RBoost: Label noise-robust boosting algorithm based on a nonconvex loss function and the numerically stable base learners. *IEEE Transactions on Neural Networks and Learning Systems*, 27(11):2216–2228, 2016. URL <https://doi.org/10.1109/TNNLS.2015.2475750>. [p220]
- A. L. Miranda, L. P. F. Garcia, A. C. Carvalho, and A. C. Lorena. Use of classification algorithms in noise detection and elimination. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 417–424. Springer, 2009. URL https://doi.org/10.1007/978-3-642-02319-4_50. [p221]
- A. D. Pozzolo, O. Caelen, and G. Bontempi. *Unbalanced: Racing for Unbalanced Methods Selection*, 2015. URL <https://CRAN.R-project.org/package=unbalanced>. R package version 2.0. [p219]
- J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. [p220]
- P. Romanski and L. Kotthoff. *FSelector: Selecting Attributes*, 2014. URL <https://CRAN.R-project.org/package=FSelector>. R package version 0.20. [p219]
- J. A. Sáez, M. Galar, J. Luengo, and F. Herrera. Analyzing the Presence of Noise in Multi-Class Problems: Alleviating Its Influence with the One-vs-One Decomposition. *Knowledge and Information Systems*, 38(1):179–206, 2014. URL <https://doi.org/10.1007/s10115-012-0570-1>. [p220]

- J. A. Sáez, M. Galar, J. Luengo, and F. Herrera. INFFC: An iterative class noise filter based on the fusion of classifiers with noise sensitivity control. *Information Fusion*, 27:19–32, 2016. URL <https://doi.org/10.1016/j.inffus.2015.04.002>. [p219]
- C.-M. Teng. Correcting Noisy Data. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 239–248, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers. [p220]
- C. M. Teng. Dealing with data corruption in remote sensing. In *International Symposium on Intelligent Data Analysis*, pages 452–463. Springer, 2005. URL https://doi.org/10.1007/11552253_41. [p221]
- I. Tomek. Two modifications of CNN. *IEEE Trans. Systems, Man and Cybernetics*, 6:769–772, 1976. URL <https://doi.org/10.1109/TSMC.1976.4309452>. [p219]
- S. van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011. URL <https://doi.org/10.18637/jss.v045.i03>. [p219]
- R. Y. Wang, V. C. Storey, and C. P. Firth. A Framework for Analysis of Data Quality Research. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):623–640, 1995. URL <https://doi.org/10.1109/69.404034>. [p220]
- D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(3):408–421, 1972. URL <https://doi.org/10.1109/TSMC.1972.4309137>. [p219]
- I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005. [p219, 220]
- X. Wu. *Knowledge Acquisition from Databases*. Ablex Publishing Corp., Norwood, NJ, USA, 1996. [p220]
- X. Wu and X. Zhu. Mining with noise knowledge: Error-aware data mining. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(4):917–932, 2008. URL <https://doi.org/10.1109/TSMCA.2008.923034>. [p220]
- S. Zhong, T. M. Khoshgoftaar, and N. Seliya. Analyzing Software Measurement Data with Clustering Techniques. *IEEE Intelligent Systems*, 19(2):20–27, 2004. URL <https://doi.org/10.1109/MIS.2004.1274907>. [p220]
- X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22(3):177–210, 2004. URL <https://doi.org/10.1007/s10462-004-0751-8>. [p220]

Pablo Morales

Department of Computer Science and Artificial Intelligence, University of Granada
18071 Granada, Spain
pablmorales@decsai.ugr.es

Julián Luengo

Department of Computer Science and Artificial Intelligence, University of Granada
18071 Granada, Spain
julianlm@decsai.ugr.es

Luís P.F. Garcia

Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo
Trabalhador São-carlense Av. 400, São Carlos, São Paulo 13560-970, Brazil
lp Garcia@icmc.usp.br

Ana C. Lorena

Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo
Tálim St. 330, São José dos Campos, São Paulo 12231-280, Brazil
aclorena@unifesp.br

André C.P.L.F. de Carvalho

Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo
Trabalhador São-carlense Av. 400, São Carlos, São Paulo 13560-970, Brazil
andre@icmc.usp.br

Francisco Herrera
Department of Computer Science and Artificial Intelligence, University of Granada
18071 Granada, Spain
herrera@decsai.ugr.es

coxphMIC: An R Package for Sparse Estimation of Cox Proportional Hazards Models via Approximated Information Criteria

by Raziieh Nabi and Xiaogang Su

Abstract In this paper, we describe an R package named `coxphMIC`, which implements the sparse estimation method for Cox proportional hazards models via approximated information criterion (Su et al., 2016). The developed methodology is named MIC which stands for “Minimizing approximated Information Criteria”. A reparameterization step is introduced to enforce sparsity while at the same time keeping the objective function smooth. As a result, MIC is computationally fast with a superior performance in sparse estimation. Furthermore, the reparameterization tactic yields an additional advantage in terms of circumventing post-selection inference (Leeb and Pötscher, 2005). The MIC method and its R implementation are introduced and illustrated with the PBC data.

Introduction

Time to event (survival time) is often a primary outcome of interest in many research areas, especially in medical research such as time that takes to respond to a particular therapy, time to death, remission, or relapse. Survival times are typically right skewed and subject to censoring due to study termination, loss of follow ups, or withdrawals. Moreover, covariates may vary by time.

Cox Proportional Hazards (PH) model (Cox, 1972) is commonly used to model survival data. Given a typical survival data set that consists of $\{(T_i, \delta_i, \mathbf{z}_i) : i = 1, \dots, n\}$, where T_i is the observed event time, δ_i is the 0-1 binary censoring indicator, and $\mathbf{z}_i \in \mathbb{R}^p$ is the covariate vector associated with the i -th subject, the Cox PH model formulates the hazard function $h(t|\mathbf{z}_i)$ for the i th subject as

$$h(t|\mathbf{z}_i) = h_0(t) \exp(\boldsymbol{\beta}^T \mathbf{z}_i),$$

where $\mathbf{z}_i \in \mathbb{R}^p$ denotes the p -dimensional covariate vector associated with subject i , $\boldsymbol{\beta} = (\beta_j) \in \mathbb{R}^p$ is the unknown regression parameter vector, and $h_0(t)$ is the unspecified baseline hazard function. The vector of $\boldsymbol{\beta}$ can be estimated by maximizing the partial log-likelihood (Cox, 1975), which is given by

$$l(\boldsymbol{\beta}) = \sum_{i=1}^n \delta_i \left[\mathbf{z}_i^T \boldsymbol{\beta} - \log \sum_{i'=1}^n \left\{ I(T_{i'} \geq T_i) \exp(\mathbf{z}_{i'}^T \boldsymbol{\beta}) \right\} \right].$$

Let $\hat{\boldsymbol{\beta}}$ denote the resultant maximum partial likelihood estimator (MPLE).

Since the true $\boldsymbol{\beta}$ is often sparse, we need to look for methods that identify the zero components in $\boldsymbol{\beta}$ and at the same time estimate the nonzero ones. Best subset selection (BSS) and regularization are among two major algorithms used in survival analyses for variable selection. Both are derived from a penalized partial likelihood. Let $\text{pen}(\boldsymbol{\beta})$ and λ denote the penalty function and penalty parameter, respectively. The general objective function in both of the techniques is as follows:

$$\min_{\boldsymbol{\beta}} -2l(\boldsymbol{\beta}) + \lambda \cdot \text{pen}(\boldsymbol{\beta}).$$

In BSS, the penalty function is set to $\text{pen}(\boldsymbol{\beta}) = \sum_{j=1}^p I\{\beta_j \neq 0\}$ (number of nonzero coefficients), and the penalty parameter is fixed as $\lambda = 2$ for AIC (Akaike, 1974) or $\lambda = \ln(n_0)$, where n_0 is the total number of uncensored failures, with a slight modification of BIC (Vollins and Raftery, 2000). In regularization, the penalty function is set to $\text{pen}(\boldsymbol{\beta}) = \sum_{j=1}^p |\beta_j|$, and the penalty parameter is not fixed and is appropriately chosen. The sparse estimation is reformulated into a continuous convex optimization problem. The optimization of the two techniques is a two-step process. In BSS, one needs to fit every model with the maximum partial likelihood method and then compare the fitted models according to an information criterion such as AIC (Akaike, 1974) or BIC (Schwarz, 1978). This makes the BSS infeasible for moderately large p . In regularization, one need to solve the objective function for every fixed positive value of λ to obtain a regularization path $\{\tilde{\boldsymbol{\beta}}(\lambda) : \lambda > 0\}$, and then select the best λ according to an information criterion such as AIC or BIC along the path. Since such a search is only along the regularization path (a one-dimensional curve in \mathbb{R}^p), the search space is

much reduced and hence, it may not perform as well as the estimator obtained with BSS, if AIC or BIC is used as the yardstick. Beside the computational burden, both methods face the post-selection inference challenge. A new technique is developed by Su et al. (2016) on the basis of Su (2015) for conducting sparse estimation of Cox PH models to help address the aforementioned deficiencies.

The MIC method

A new method, named MIC for “Minimizing approximated Information Criteria”, is developed to conduct sparse estimation of Cox PH models. MIC borrows strength from both BSS and regularization. The main issue with BSS is the indicator function, $I(\beta \neq 0)$, involved in the ℓ_0 penalty function, leading to a discrete optimization problem. To overcome this difficulty, MIC proposes to approximate the indicator function by a continuous or smooth unit dent function. One reasonable approximation is the hyperbolic tangent function given by

$$w(\beta) = \tanh(a\beta^2) = \frac{\exp(a\beta^2) - \exp(-a\beta^2)}{\exp(a\beta^2) + \exp(-a\beta^2)},$$

where a is a nonnegative scale parameter that controls the sharpness of the approximation.

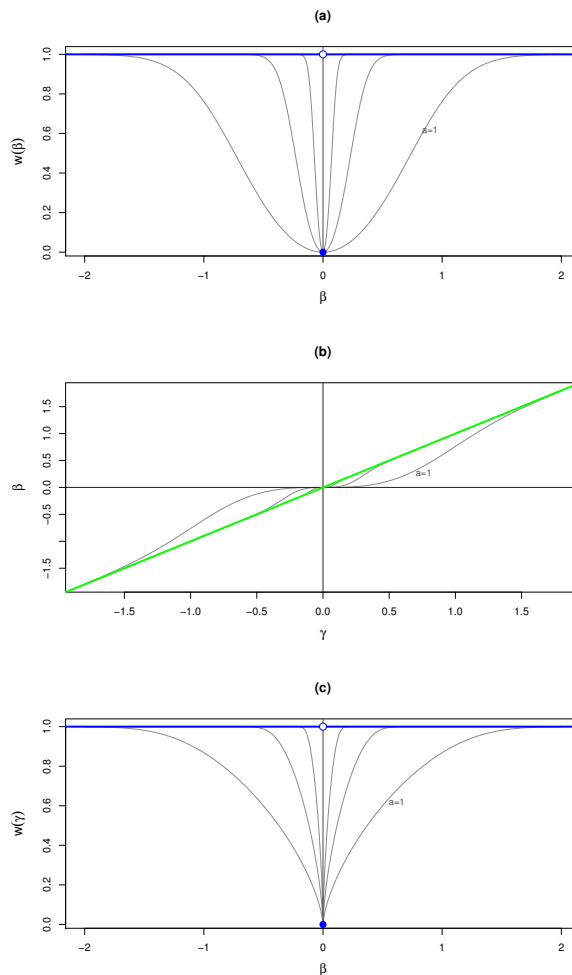


Figure 1: MIC penalty and the reparameterization step: (a) the hyperbolic tangent penalty $\tanh(a\beta^2)$ versus β ; (b) $\beta = \gamma \tanh(a\gamma^2)$ versus γ ; (c) $\tanh(a\gamma^2)$ as a penalty function of β . Three values of $a \in \{1, 10, 100\}$ are illustrated.

As shown in Figure 1(a), $w(\beta)$ provides a smooth approximation to the discrete function $I\{\beta \neq 0\}$. However, the curve does not have zero as a singular point. If we estimate β by minimizing $-2l(\beta) + \ln(n_0) \sum_{j=1}^p w(\beta_j)$, we will not obtain sparse estimates. To enforce sparsity, MIC devises a reparameterization step. The reparameterization is based on the decomposition $\beta = \beta I\{\beta \neq 0\}$. Set $\gamma = \beta$ and approximate $I\{\beta \neq 0\}$ with $w(\gamma) = \tanh(a\gamma^2)$. This leads to a reparameterization of

$\beta = \gamma w(\gamma)$. The objective function in MIC is given by

$$Q_n(\beta) = -2l(\mathbf{W}\gamma) + \lambda_0 \text{tr}(\mathbf{W}), \quad (1)$$

where the penalty parameter λ_0 is fixed as $\ln(n_0)$ for BIC (Vollins and Raftery, 2000) and matrix \mathbf{W} is $p \times p$ diagonal with diagonal elements $w_j = w(\gamma_j)$ and hence trace $\text{tr}(\mathbf{W}) = \sum_{j=1}^p \tanh(a\gamma_j^2)$. With this notation, it follows that $\beta = \mathbf{W}\gamma$.

The above reparameterization offers several important conveniences:

1. Sparsity now becomes achievable in estimating β . The penalty $w(\gamma)$ as a function of $\beta = \gamma w(\gamma) = \gamma \tanh(\gamma)$ is a unit dent function that is smooth everywhere except at $\beta = 0$, as shown in Figure 1(c). This is a necessary condition to ensure sparsity as indicated by Fan and Li (2002). On this basis, the oracle properties of the MIC estimator $\tilde{\beta}$ obtained by minimizing $Q_n(\beta)$ in (1)

$$\tilde{\beta} = \arg \min_{\beta} Q_n(\beta) = \arg \min_{\beta} -2l(\beta) + \ln(n_0) \sum_{j=1}^p w(\gamma_j)$$

can be established under regularity conditions. The asymptotic results entails $a_n = O(n)$. For this reason, we fix $a_n = n_0$, the number of non-censored failures. In practice, the empirical performance of MIC is large stable with respect to the choice of a , as demonstrated in Su (2015). Thus simply fixing a at a reasonably large value (say, $a \geq 10$) could do as well practically.

2. In terms of practical optimization, it is preferable to consider γ as the decision vector. Namely, we minimize $Q_n(\gamma)$ with respect to γ by treating it as a function of γ . Let $\tilde{\gamma}$ be the resultant MIC estimator of γ

$$\tilde{\gamma} = \arg \min_{\gamma} Q_n(\gamma) = \arg \min_{\gamma} -2l(\mathbf{W}\gamma) + \ln(n_0) \sum_{j=1}^p w(\gamma_j). \quad (2)$$

One immediate advantage of doing so is that $Q_n(\gamma)$ is smooth in γ and hence many optimization routines can be applied directly. Since no selection of tuning parameters is involved, MIC is computationally efficient.

3. One consequence of post-selection inference is that no standard error formula is available for zero estimates of β_j . As depicted in Figure 1(b), β_j and γ_j have a one-to-one correspondence with $\beta = 0$ iff $\gamma = 0$. This motivates us to test $H_0: \beta_j = 0$ by equivalently testing $H_0: \gamma_j = 0$. The MIC estimator $\tilde{\gamma}$ can be viewed as an M-estimator with smooth objective function $Q_n(\gamma)$ and hence standard arguments can be used to make inference.

Implementation in R

The R package **coxphMIC** implements MIC on the basis of R package **survival** (Therneau and Grambsch, 2000) and is hosted at CRAN. Type the following command in R console in order to install the package:

```
> install.packages("coxphMIC")
```

To summarize, MIC can be simply formulated as the following optimization problem

$$\min_{\gamma} -2l(\mathbf{W}\gamma) + \ln(n_0) \sum_{j=1}^p \tanh(n_0\gamma_j^2). \quad (3)$$

Owing to the non-convex nature, a global optimization method is helpful in solving (3). While other R routines (Mullen, 2014) are available, we have found using the SANN method combined with the BFGS method in R function `optim()` is fast and quite effective. The simulated annealing (SA) implemented by SANN helps locate a nearly minimum point globally. Then the quasi-Newton BFGS method makes sure that the algorithm stops at a critical point.

There are two functions included in the **coxphMIC** package: an internal function `LoglikPen()` that computes the partial log-likelihood and a wrapper function `coxphMIC()` that does the MIC sparse estimation. The function `coxphMIC()` has the following usage:

```
coxphMIC(formula = Surv(time, status) ~ ., data, method.beta0 = "MPL",
         beta0 = NULL, theta0 = 1, method = "BIC", lambda0 = NULL, a0 = NULL,
         scale.x = TRUE, maxit.global = 300, maxit.local = 100,
         rounding.digits = 4, zero = sqrt(.Machine$double.eps),
```

n	p	Censoring		Method				
		Rate	Full	Stepwise	MIC	LASSO	ALASSO	SCAD
200	10	25%	0.007	0.307	0.067	0.157	0.163	5.923
		40%	0.000	0.320	0.060	0.150	0.170	4.900
	50	25%	0.027	18.957	0.063	0.397	0.417	5.587
		40%	0.027	18.107	0.057	0.453	0.480	5.480
	100	25%	0.060	189.040	0.057	1.450	1.387	—
		40%	0.067	181.147	0.057	2.053	1.897	—
2000	10	25%	0.020	0.907	0.243	0.903	0.837	415.097
		40%	0.017	0.880	0.240	0.893	0.823	328.150
	50	25%	0.110	81.380	0.243	1.590	1.153	—
		40%	0.093	72.887	0.237	1.613	1.163	—
	100	25%	0.333	894.607	0.223	2.383	2.103	—
		40%	0.240	673.503	0.187	2.073	1.357	—

Table 1: Comparison of computation time: CPU time (in seconds) averaged over three runs.

```
compute.se.gamma = TRUE, compute.se.beta = TRUE,
CI.gamma = TRUE, conf.level = 0.95,
details = FALSE)
```

We briefly explain some of the important options. The formula argument is a formula object similar to that in `survival`, with the response on the left of the `~` operator being a survival object as returned by the `Surv` function, and the terms on the right being predictors. The arguments `method`, `beta0`, `beta0`, and `theta0` pertains to the initial starting values. By default, the maximum partial likelihood estimator with the option `MPL` is used. Otherwise, one can use the ridge estimator with option `ridge`. The `theta0` corresponds to the tuning parameter in ridge estimation. User defined starting values can also be used such as $\beta = \gamma = \mathbf{0}$ by specifying `beta0`. By default, the approximated BIC (Vollins and Raftery, 2000) is recommended. However, one can use AIC (Akaike, 1974). Alternatively, user-specified penalty is allowed by specifying `lambda0`. The default value for `a` is `n0`. The option `maxit.global` allows for specification of the maximal iteration steps in SANN while `maxit.local` specifies the maximal iteration steps for BFGS. MIC computes the standard errors (SE) for both $\tilde{\beta}$ and $\tilde{\gamma}$. For $\tilde{\beta}$, the SE computation is only applicable for its nonzero components. The option `maxit.global` asks whether the user wants to output the confidence intervals for γ_j at the confidence level specified by `conf.level` (with 95% as default).

The output of Function `coxphMIC()` is an object of S3 class `coxphMIC`, which is essentially a list of detailed objects that can be used for other purposes. In particular, the item `result` presents the most important results, where one can see the selected model and inference based on testing γ . Two generic functions, `print` and `plot`, are made available for exploring a `coxphMIC` object.

Other R packages for variable selection in Cox PH models

Several other R packages are available for variable selection of Cox PH models. The best subset selection (BSS) is available in the R Package `glmulti` (Calcagno and de Mazancourt, 2010) with AIC only, but it is very slow owing to the intensive computation involved. For large p , a stepwise selection procedure could be used as a surrogate. LASSO (Tibshirani, 1997) can be computed via R Package `glmnet` (Friedman et al., 2010). Zhang and Lu (2007) have made their R codes for implementing ALASSO for Cox models available at http://www4.stat.ncsu.edu/~hzhang/paper/cox_new.tar. But the program was written without resorting well to available R routines and it takes an unnecessarily long running time. One alternative way to compute ALASSO is first transform the design matrix $Z := Z \text{diag}(|\hat{\beta}|)$ so that LASSO could be applied and then transform the resultant estimates back. SCAD for Cox PH models (Fan and Li, 2002; Fan et al., 2010) can be computed with an earlier version of the R package `SIS` (Saldana and Feng, 2016), but it is no longer available in its current version. One is referred to Table 1, which is presented as Table B1 in Su et al. (2016), for a comparison study of these above-mentioned methods. MIC clearly stands out as the top or among-the-top performer in both sparse estimation and computing time.

Examples

We illustrate the usage of `coxphMIC` via an analysis of the PBC (primary biliary cirrhosis) data, available from the `survival` package (Therneau and Grambsch, 2000).

Data preparation

To proceed, some minor data preparation is needed. First of all, we want to make sure that the censoring indicator is 0-1 binary.

```
> library(survival); data(pbc);
> dat <- pbc; dim(dat);
[1] 418 20
> dat$status <- ifelse(pbc$status == 2, 1, 0)
```

Next, we explicitly created dummy variable for categorical variables. The `factor()` function could be used instead. Also, grouped sparsity could be used to handle these dummy variables so that they are either all selected or all excluded. We plan to explore this possibility in future research.

```
> dat$sex <- ifelse(pbc$sex == "f", 1, 0)
```

Another necessary step is to handle missing values. This current version does not automatically treat missings. Here, the listwise deletion is used so that only the 276 subjects with complete records are used for further analysis.

```
> dat <- na.omit(dat);
> dim(dat);
[1] 276 20
> head(dat)
  id time status trt      age sex ascites hepato spiders edema bili chol
1  1  400      1  1 58.76523  1     1     1      1  1.0 14.5 261
2  2 4500      0  1 56.44627  1     0     1      1  0.0  1.1 302
3  3 1012      1  1 70.07255  0     0     0      0  0.5  1.4 176
4  4 1925      1  1 54.74059  1     0     1      1  0.5  1.8 244
5  5 1504      0  2 38.10541  1     0     1      1  0.0  3.4 279
7  7 1832      0  2 55.53457  1     0     1      0  0.0  1.0 322
  albumin copper  alk.phos   ast trig platelet protime stage
1    2.60   156  1718.0 137.95 172     190   12.2    4
2    4.14    54  7394.8 113.52  88     221   10.6    3
3    3.48   210   516.0  96.10  55     151   12.0    4
4    2.54    64  6121.8  60.63  92     183   10.3    4
5    3.53   143   671.0 113.15  72     136   10.9    3
7    4.09    52   824.0  60.45 213     204    9.7    3
```

The data set now contains 20 variables. Except `id`, `time`, and `status`, there are a total of 17 predictors.

MIC starting with MPLE

To apply `coxphMIC`, one simply proceeds in the usual way of using `coxph` formula. By default, all predictors are standardized; the approximated BIC ($\lambda_0 = \ln(n_0)$) is used with $a = n_0$; and the MPLE is used as the starting point.

```
> fit.mic <- coxphMIC(formula = Surv(time, status)~.-id, data = dat, CI.gamma = FALSE)
> names(fit.mic)
[1] "opt.global" "opt.local" "min.Q"      "gamma"      "beta"      "VCOV.gamma"
[7] "se.gamma"  "se.beta"   "BIC"       "result"    "call"
```

The output of `coxphMIC` contains the minimized Q_n value, the final estimates of γ and β , the variance-covariance matrix and SE for $\tilde{\gamma}$, SE for nonzero $\tilde{\beta}$, BIC value for the final model, and a summary table `result`. In order for the user to be able to inspect the convergence and other detailed info of the optimization algorithms, we also output two objects `opt.global` and `opt.local`, which result from the global (SANN by default) and local optimization (BFGS by default) algorithms.

The output `fit.mic` is a S3 object of `coxphMIC` class. Two generic functions, `print` and `plot`, are available. The `print` function provides a summary table as below:

```
> print(fit.mic)
      beta0  gamma se.gamma  z.stat p.value beta.MIC se.beta.MIC
trt      -0.0622 0.0000  0.1071  0.0000 1.0000  0.0000      NA
age       0.3041 0.3309  0.1219  2.7138 0.0067  0.3309      0.1074
sex      -0.1204 0.0000  0.1086 -0.0002 0.9998  0.0000      NA
ascites   0.0224 0.0000  0.0991  0.0000 1.0000  0.0000      NA
hepato    0.0128 0.0000  0.1259  0.0000 1.0000  0.0000      NA
spiders   0.0460 0.0000  0.1118 -0.0001 1.0000  0.0000      NA
edema     0.2733 0.2224  0.1066  2.0861 0.0370  0.2224      0.0939
bili      0.3681 0.3909  0.1142  3.4237 0.0006  0.3909      0.0890
chol      0.1155 0.0000  0.1181  0.0002 0.9999  0.0000      NA
albumin  -0.2999 -0.2901  0.1248 -2.3239 0.0201 -0.2901     0.1103
copper    0.2198 0.2518  0.1050  2.3986 0.0165  0.2518      0.0868
alk.phos  0.0222 0.0000  0.0837  0.0000 1.0000  0.0000      NA
ast       0.2308 0.2484  0.1128  2.2023 0.0276  0.2484      0.1025
trig     -0.0637 0.0000  0.0858  0.0000 1.0000  0.0000      NA
platelet  0.0840 0.0000  0.1129  0.0000 1.0000  0.0000      NA
protime   0.2344 0.2293  0.1046  2.1917 0.0284  0.2293      0.1022
stage     0.3881 0.3692  0.1476  2.5007 0.0124  0.3692      0.1243
```

The above results are presented as Table 4 in [Su et al. \(2016\)](#). In this example, MIC started with MPLE given by the first column named beta0. Columns 2–5 present estimation of γ and the hypothesis testing results on $H_0 : \gamma_j = 0$. The estimates of β are given in the last two columns. It can be seen that eight variables are selected in the final model, which are age, edema, bili, albumin, copper, ast, protime, and stage.

The plot function provides error bar plots based on the MIC estimator of both β and the reparameterized γ :

```
> plot(fit.mic, conf.level = 0.95)
```

as shown in Figure 2. Essentially, the 95% confidence intervals (CI) are plotted. One can modify the confidence level with the `conf.level` option. To compare two plots conveniently, they are made with the same range on the vertical y-axis. Note that CI is not available for any zero β_j estimate in Panel (b), which corresponds to an unselected variable. Those selected variables are highlighted in green color in Panel.

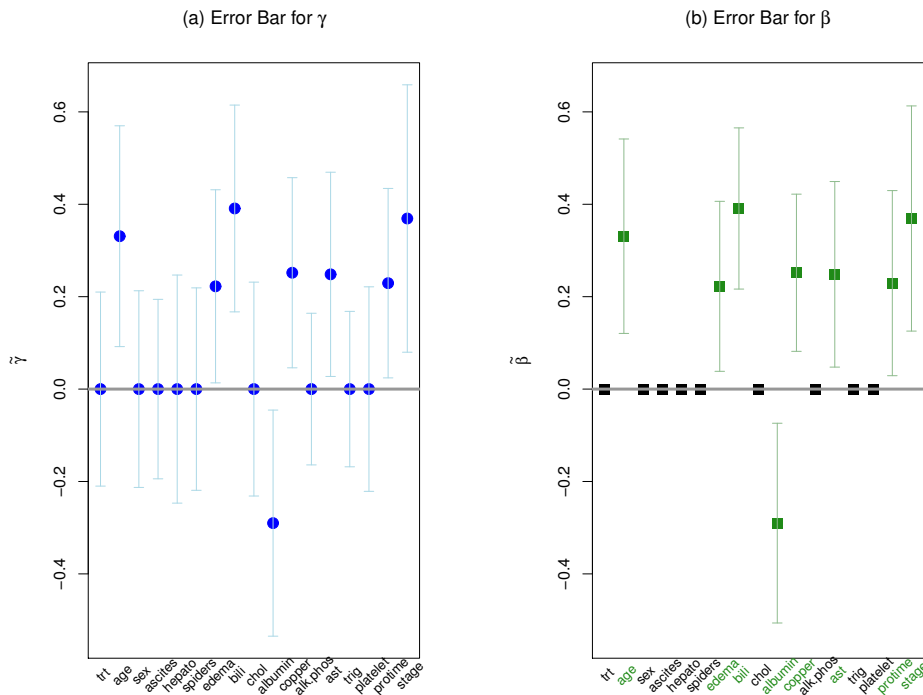


Figure 2: Error bar plots for MIC estimates of γ in (a) and β in (b). The 95% confidence intervals (CI) are plotted. The selected variables are highlighted in green in Panel (b).

Multiple starting points

Trying out multiple starting point is a common strategy in facing global optimization problems. We may consider starting with the $\mathbf{0}$ vector, which corresponds to the null model. Having $\text{beta0} = \mathbf{0}$ is actually the default option if `method.beta0` is neither 'MPLE' nor 'ridge' and a specific value for `beta0` is not given, i.e., setting `beta0 = NULL`.

```
> fit0.mic <- coxphMIC(formula = Surv(time, status)~.-id, data = dat,
+ method = "BIC", scale.x = TRUE, method.beta0 = "zero")

> c(fit.mic$min.Q, fit0.mic$min.Q)
[1] 974.3340 978.1232
```

We can compare the minimized objective function `min.Q` to decide which fitting result is preferable (i.e., the smaller one). The above result suggests that the fit with MPLE as starting point remains preferable.

Concerning sparse estimation, the vectors with 0/+1/-1 values obtained by applying a threshold to the MPLE $|\hat{\beta}|$ could be reasonable choices for the starting point too, i.e.,

$$\beta_{0j} := \text{sgn}(\hat{\beta}_j) I\{|\hat{\beta}_j| > c_0\},$$

where $c_0 > 0$ is a threshold close to 0. For example, setting $c = 0.06$ yields

```
> beta.MPLE <- fit.mic$result[, 1]
> beta0 <- sign(beta.MPLE)*sign(abs(beta.MPLE) > .06);
> cbind(beta.MPLE, beta0)
  beta.MPLE beta0
[1,] -0.0622  -1
[2,]  0.3041   1
[3,] -0.1204  -1
[4,]  0.0224   0
[5,]  0.0128   0
[6,]  0.0460   0
[7,]  0.2733   1
[8,]  0.3681   1
[9,]  0.1155   1
[10,] -0.2999  -1
[11,]  0.2198   1
[12,]  0.0022   0
[13,]  0.2308   1
[14,] -0.0637  -1
[15,]  0.0840   1
[16,]  0.2344   1
[17,]  0.3881   1
```

In the above example, we applied a threshold of 0.06 to the MPLE to obtain a 0/+1/-1 valued vector. To start with this user-supplied starting point, one proceeds as follows.

```
> fit1.mic <- coxphMIC(formula = Surv(time, status)~.-id, data = dat,
+ method = "BIC", scale.x = TRUE, method.beta0 = "user-supplied", beta0 = beta0)
> c(fit.mic$min.Q, fit0.mic$min.Q, fit1.mic$min.Q)
[1] 974.3340 978.1232 979.6826
```

Again, the fitting starting at MPLE seems the best in this example, by giving the smallest minimized value.

Different a values

We may consider obtaining the regularization path with respect to a . According to asymptotic results, $a = O(n)$ is desirable and the recommended value is $a = n_0$ the number of uncensored deaths, which is $n_0 = 111$ in the PBC data under study.

We try out a spread of a values that range from 10 to 200, as prescribed by the R object `A0`.

```
> set.seed(818)
> n <- NROW(dat); n0 <- sum(dat$status == 1)
> A0 <- 10:200
```

```

> p <- NCOL(dat)-3
> BETA <- matrix(0, nrow = length(A0), ncol = p) # USE ARRAY
> for (j in 1:length(A0)){
  su.fit <- coxphMIC(formula = Surv(time, status)~.-id, data = dat, a0 = A0[j],
    method = "BIC", scale.x = TRUE)
  BETA[j, ] <- su.fit$beta
}
> BETA <- as.data.frame(BETA)
> colnames(BETA) <- colnames(dat)[- (1:3)]
> row.names(BETA) <- A0
> head(BETA, n = 5)
  trt   age sex ascites hepato spiders edema  bili chol albumin copper alk.phos
10  0 0.2983  0      0      0      0 0.2024 0.4135  0 -0.2799 0.2495      0
11  0 0.2987  0      0      0      0 0.2015 0.4159  0 -0.2799 0.2491      0
12  0 0.2992  0      0      0      0 0.2006 0.4181  0 -0.2799 0.2487      0
13  0 0.3000  0      0      0      0 0.1998 0.4200  0 -0.2801 0.2482      0
14  0 0.3009  0      0      0      0 0.1992 0.4216  0 -0.2804 0.2478      0

  ast trig platelet protime stage
10 0.1937  0      0 0.1912 0.3583
11 0.1924  0      0 0.1895 0.3612
12 0.1914  0      0 0.1878 0.3642
13 0.1906  0      0 0.1862 0.3672
14 0.1901  0      0 0.1847 0.3701

```

A plot of the regularization path with respect to a , as shown in Figure 3, can be obtained as follows:

```

> par(mar = rep(5,4), mfrow = c(1,1))
> x.min <- min(A0); x.max <- max(A0)
> plot(x = c(x.min, x.max), y = c(min(BETA), max(BETA)), type = "n",
+   xlab = "a", cex.lab = 1.2, las = 1, ylab = expression(tilde(beta)))
> for (j in 1:ncol(BETA)){
+   lines(x = A0, y = BETA[,j], col = "red", lty = 1, lwd = 1)
+   points(x = A0, y = BETA[,j], col = "red", pch = j, cex = .3)
+   vname <- colnames(BETA)[j]
+   if (abs(BETA[nrow(BETA),j]) > .00001) {
+     # text(x.max+5, BETA[nrow(BETA),j], labels = vname, cex = 1, col = "blue")
+     mtext(text = vname, side = 4, line = 0.5, at = BETA[nrow(BETA),j], las = 1,
+       cex = 1, col = "blue", font = 1)
+   }
+ }
> abline(h = 0, col = "gray25", lwd = 2)
> abline(v = n0, col = "gray45", lwd = 1.5)
> text(n0+5, -0.2, expression(paste("a = ", n[0], " = ", 111, sep = "")), cex = 1.2,
+ col = "gray35")

```

From Figure 3, it can be seen that the regularization path is essentially flat with respect to a , especially for relatively large a values. This indicates that treating a as a tuning parameter is unnecessary.

Summary

The paper presents the `coxphMIC` package to implement the MIC method for Cox proportional hazards models. Compared to several other competitive methods, MIC has three main advantages by offering a superior empirical performance for it aims to minimize BIC (albeit approximated) without reducing the search space, great computational efficiency since it does not involve selection of any tuning parameter, and a leeway to perform significance testing that is free of the post-selection inference.

Bibliography

- H. Akaike. A new look at model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. [p229, 232]
- V. Calcagno and C. de Mazancourt. `Glmulti`: An R package for easy automated model selection with (generalized) linear models. *Journal of Statistical Software*, 34(12), 2010. [p232]

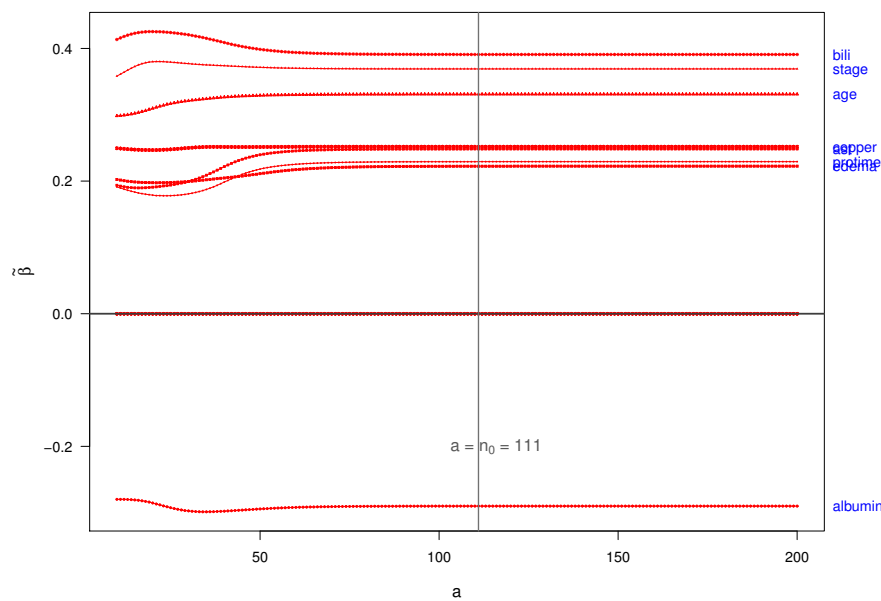


Figure 3: The regularization path with respect to a with the PBC data. The a value varies from 10 to 200. The recommended value is $a = n_0 = 111$.

- D. R. Cox. Regression models and life tables (with discussion). *Journal of the Royal Statistical Society B*, 34(2):187–220, 1972. [p229]
- D. R. Cox. Partial likelihood. *Biometrika*, 62(2):269–276, 1975. [p229]
- J. Fan and R. Li. Variable selection for cox’s proportional hazards model and frailty model. *The Annals of Statistics*, 30(1):74–99, 2002. [p231, 232]
- J. Fan, Y. Feng, and Y. Wu. High-dimensional variable selection for cox’s proportional hazards model. *The Institute of Mathematical Statistics*, 6:70–86, 2010. [p232]
- J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 2010. [p232]
- H. Leeb and B. M. Pötscher. Model selection and inference: Facts and fiction. *Econometric Theory*, 21(1): 21–59, 2005. [p229]
- K. M. Mullen. Continuous global optimization in R. *Journal of Statistical Software*, 60(6), 2014. URL <https://www.jstatsoft.org/article/view/v060i06/v60i06.pdf>. [p231]
- D. F. Saldana and Y. Feng. SIS: An R package for sure independence screening in ultrahigh-dimensional statistical models. *Journal of Statistical Software*, 2016. URL <http://www.stat.columbia.edu/~diego/PapersandDraft/SIS.pdf>. In press. [p232]
- G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. [p229]
- X. Su. Variable selection via subtle uprooting. *Journal of Computational and Graphical Statistics*, 24(4): 1092–1113, 2015. [p230, 231]
- X. Su, C. S. Wijayasinghe, J. Fan, and Y. Zhang. Sparse estimation of cox proportional hazards models via approximated information criteria. *Biometrics*, 72:751–759, 2016. [p229, 230, 232, 234]
- T. Therneau and P. Grambsch. Modeling survival data: Extending the cox model. *Springer-Verlag*, 2000. [p231, 233]
- R. Tibshirani. The LASSO method for variable selection in the Cox model. *Statistics in Medicine*, 16(4): 385–395, 1997. [p232]
- C. T. Vollinsk and A. E. Raftery. Bayesian information criterion for censored survival models. *Biometrics*, 56(1):256–262, 2000. [p229, 231, 232]

H. Zhang and W. Lu. Adaptive lasso for Cox's proportional hazards model. *Biometrika*, 94(3):691–703, 2007. [p232]

Razieh Nabi
Department of Computer Science
Johns Hopkins University
Maryland 21218, USA
rnabiab1@jhu.edu

Xiaogang Su
Department of Mathematical Sciences
University of Texas at El Paso
Texas 79968, USA
xsu@utep.edu

Update of the nlme Package to Allow a Fixed Standard Deviation of the Residual Error

by Simon H. Heisterkamp, Engelbertus van Willigen, Paul-Matthias Diderichsen, John Maringwa

Abstract The use of linear and non-linear mixed models in the life sciences and pharmacometrics is common practice. Estimation of the parameters of models not involving a system of differential equations is often done by the R or S-Plus software with the nonlinear mixed effects **nlme** package. The estimated residual error may be used for diagnosis of the fitted model, but not whether the model correctly describes the relation between response and included variables including the true covariance structure. The latter is only true if the residual error is known in advance. Therefore, it may be necessary or more appropriate to fix the residual error a priori instead of estimate its value. This can be the case if one wants to include evidence from past studies or a theoretical derivation; e.g., when using a binomial model. S-Plus has an option to fix this residual error to a constant, in contrast to R. For convenience, the **nlme** package was customized to offer this option as well. In this paper, we derived the log-likelihoods for the mixed models using a fixed residual error. By using some well-known examples from mixed models, we demonstrated the equivalence of R and S-Plus with respect to the estimates. The updated package has been accepted by the Comprehensive R Archive Network (CRAN) team and will be available at the CRAN website.

Introduction

Life science and pharmacometric studies almost always involve an application of linear (Laird and Ware, 1982) and non-linear (Davidian and Giltinan, 1995) mixed models. In pharmacometrics, the use of nonlinear mixed effects modelling (NONMEM) software (Beal and Sheiner, 1988) for models with systems of differential equations is predominant. Simpler pharmacokinetic (PK) or pharmacodynamic (PD) models may be based on curve fitting, for which software such as S-Plus and R are preferred. Both programs use a similar mixed model package, respectively, **nlme** and nlme library, both of which were originally developed by the authors Pinheiro and Bates (Pinheiro and Bates, 2001). In most cases, the estimates of the parameters are the same at least to the third significant digit, although the same model and data might occasionally lead to convergence problems (e.g., in R and not in S-Plus, and vice versa), although larger differences may occur e.g., when performing an analysis of variance with the function `anova`, as shown in 34.6. Typically, the estimated fixed and random coefficients have a practical interpretation. The fixed coefficients usually refer to the parameters of interest for the study; i.e. treatment, gender differences, half-life of the drug substance etc., while the random parameters are interpreted as variation in the population represented by the study. The residual error of the fitted model is an important indicator for the goodness of fit. Whether the goodness of fit is assessed by the log-likelihood ratio, the Akaike information criterion (AIC) (Akaike, 1980) or the Bayesian information criterion (BIC) (Schwartz, 1978) criteria, all criteria implicitly or explicitly use the residual error of the fit.

For some applications, it is not appropriate to estimate the residual error, especially when it is known in advance based on evidence from past studies, or a theoretically derived scaling parameter. An example of the former are meta-analyses where the standard deviation of individual patient outcomes in each study is reported. An example of the latter is a data transformation, implying a fixed scaling parameter; e.g., the square root arcsine or logit transformation may be used as an approximation of the binomial distribution.

As pointed out by an anonymous reviewer, other applications may be considered as well (Littel et al., 2006). For example simulation of new observations from the fitted model while keeping sigma constant or performing power calculations where commonly the expected error is assumed to be known and kept constant. In both cases one must bear in mind that the simulated data will not cover the whole range of possible outcomes, and in case of power calculation the actual power may be well below the nominal power one aimed at.

The fitting functions from the nlme library in S-Plus allow the residual standard error to be fixed and kept constant during the fitting process. This feature was not described in Pinheiro and Bates (Pinheiro and Bates, 2001) but has been added as an argument to the respective control functions. This feature was not available in the R package **nlme** but is included from version 3.1.123 (released 2016-01-17) onward.

We present this feature of the control functions used in linear mixed effects `lme`, non-linear mixed

effects `nlme`, generalized linear least squares `gls` and generalized non-linear least squares `gnls`. Utilities `summary`, `anova`, `print` and `intervals` have been adapted as well.

From a coding point of view these changes were significant, as they required, among others, a change of the likelihood functions in both the R and C source codes. This paper discusses the statistical background in Section 2. In Section 3 the similarities and differences of the output of S-Plus version 8.2 are given for selected examples from Pinheiro and Bates (Pinheiro and Bates, 2001) as well as a case study describing a model-based meta-analysis of outcomes in rheumatoid arthritis. In the last Section proposals for change in `nlme` are put forward to address some of these similarities and differences.

The general linear and linear mixed model

The mixed model is an extension of the general linear model. The extension to a mixed model allows other parameters of the general linear model to have a statistical normal distribution (Laird and Ware, 1982). These models are also known as multilevel models in different fields of application, as they describe a hierarchy of levels in the data. For example, in a clinical trial, the first level might be the trial centre, the second might be the patients within the trial centre, and the third might be the time points at which the observations are acquired for an individual patient.

The reason for such a break-down of the random variation is that, within a centre, the variation between subjects may be different than between centres; e.g., differences in environment, skills of the operators, measurement devices, or the mixture of ethnicity of the patients. Variation within patients is likely to be different and correlation of longitudinal observations within a patient may be present.

The expected outcome may be linear in the design parameters, including linear combinations of polynomials and general additive models (Wood, 2006). Similarly parameters of the non-linear models are allowed to be linear combinations (Davidian and Giltinan, 1995). General linear and mixed linear models are fitted with `gls` and `lme` respectively. The corresponding non-linear models (e.g. a Michaelis-Menten model curve) are fitted with `gnls` and `nlme`.

Mixed models have two sets of parameters. The fixed parameters are often of primary interest of the study. For example, differences in treatment or gender, the half-life of a drug or factors for which one wants to control like co-medication. The random parameters are used to describe variation within respective levels of the study and are generally not controllable. For example, variation between and within a population of patients. The random parameters might be dependent on some or all of the fixed parameters or perhaps a function of the fitted values. Common statistical analysis seeks to estimate all fixed and random parameters simultaneously using the same data without prior knowledge.

In the next subsections, we describe the mathematical model extending from the general linear to the linear mixed and non-linear mixed model with knowledge of the residual error. The notation and the terminology with regard to the number of hierarchical levels follows closely to that in Pinheiro and Bates (Pinheiro and Bates, 2001). Throughout the paper we will use Greek characters to indicate the true but unknown values of the model parameters, while the corresponding character superscripted by $\hat{\cdot}$ denotes its estimate.

General linear model

The general linear model with only the basic random level of the residuals is presented here. A comparison is made between the likelihood for the usual model with an estimated residual error and the one with a fixed value of the latter. For further details of the derivation see (Pinheiro and Bates, 2001). If there were M groups (e.g. subjects) with n_i possibly correlated observations each, the total number of observations is: $N = \sum_i^M n_i$. The model is expressed for each group by the following formula:

$$y_i = \mathbf{X}_i \beta + \varepsilon_i \quad \varepsilon_i \sim N(0, \sigma^2 \Lambda_i) \quad (1)$$

Here, \mathbf{X} is the $n_i \times p$ design matrix and β is a column vector of length p . The matrix Λ_i , the covariance matrix of each of the subjects, is a $n_i \times n_i$ positive-definite matrix is multiplied by σ^2 . Note that in this model, we do not assume any between-groups variance; σ^2 is the same for each of the M groups. Assuming that the M groups are all statistically independent; e.g., subjects in a trial the total $N \times N$ covariance matrix Λ is block-diagonal and each block represents the matrix Λ_i . If all observations within a group were statistically independent as well Λ_i reduced to identity matrices \mathbf{I} of the same dimensions, and the model becomes an ordinary (multiple) regression model with covariance matrix \mathbf{I} multiplied by σ^2 and can be solved by standard multiple regression software.

To solve model (1), one may reduce the model to an ordinary least squares model by transforming both sides of the equation such that the covariance matrices are again diagonal and equal to $\sigma^2 \mathbf{I}$. For

details, see Appendix .1, where it is shown that formula (1) is equivalent to the least squares model following expression:

$$y_i^* = \mathbf{X}_i^* \beta + \varepsilon_i^* \quad \varepsilon_i^* \sim N(0, \sigma^2 \mathbf{I}) \tag{2}$$

The superscript * indicates the corresponding transformed vectors and matrices. This model appears to be an ordinary least squares model, however, its solution depends on unknown parameters in Λ_i , not explicitly in the equation. Thus, conditionally on the values of Λ_i , the fixed coefficients β of the model—indicated by $\hat{\beta}$ —with the usual least squares equations are estimated, yielding:

$$\hat{\beta} = [(\mathbf{X}^*)^T \mathbf{X}^*]^{-1} (\mathbf{X}^*)^T y^*$$

The matrices Λ_i usually depend on a set of only a few parameters with unknown values, which we denote as λ . This means that the estimate $\hat{\beta}$ depends on the unknown λ as well, but to simplify the notation we write $\hat{\beta}(\lambda)$, instead of $\hat{\beta}(\hat{\lambda})$. Estimation of $\hat{\beta}(\lambda)$, implies estimation of λ as well. The method employed in the package **nlme** maximizes the profiled (minus) log-likelihood which includes λ and $\hat{\beta}$ (not β) (Pinheiro and Bates, 2001). The maximization is done for both β and λ in a stepwise manner and at each step the residual variance is estimated as:

$$\hat{\sigma}^2 = \frac{\|y^* - \mathbf{X}^* \hat{\beta}(\lambda)\|^2}{N_p}$$

The number N_p is the degrees of freedom left for this residual variance. It equals N for the maximum likelihood method of estimation and $N - p$ for the restricted maximum likelihood and the least squares method. The number p is the sum of the numbers of fixed parameters and the numbers of parameters used in the function of the covariance matrix Λ . The estimate $\hat{\sigma}^2$ is plugged into all other expressions; e.g., the standard errors of the estimates and the student tests to compare the fixed parameters and the goodness of fit criteria.

The estimation of λ , β and σ is iterative because $\hat{\lambda}$, $\hat{\beta}$ and $\hat{\sigma}^2$ are interdependent. The (minus) profiled log-likelihood of the general linear model equals :

$$lik(\lambda | y, \hat{\sigma}, \hat{\beta}) = \frac{N_p}{2} \left[\log(N_p) - \log(2\pi) - 1 - \log\left(\|y^* - \mathbf{X}^* \hat{\beta}(\lambda)\|^2\right) \right] + \frac{1}{2} \sum_{i=1}^M \log |\Lambda_i| \tag{3}$$

However, if the residual variance is fixed, the profiled likelihood will be different. To see this, one has to derive the original log likelihood function without substitution of $\hat{\sigma}^2$. This profiled likelihood now depends on σ^2 instead of $\hat{\sigma}^2$, as the former is a fixed scalar which does not enter the equations.

$$llik(\lambda | y, \sigma, \hat{\beta}) = \frac{N_p}{2} \left[-\log(2\pi) - \log(\sigma^2) \right] - \frac{\|y^* - \mathbf{X}^* \hat{\beta}(\lambda)\|^2}{2\sigma^2} + \frac{1}{2} \sum_{i=1}^M \log |\Lambda_i| \tag{4}$$

Note that if $\hat{\sigma}^2$ is substituted in (4), the expression is exactly the same as (3). The striking difference between both profiled likelihoods is that, for known σ , the function (4) is linear in the residual sum of squares, while for unknown σ , (3) is linear in the log of the residual sum of squares. The estimated parameters $\hat{\lambda}$ and $\hat{\beta}$ will change accordingly. If the fixed scalar σ and the estimated $\hat{\sigma}$ are close to each other, the differences between the estimates will be quite small. Least squares estimation uses (4) with the appropriate N_p . Neither $\hat{\lambda}$ or $\hat{\beta}$ changes compared to maximum likelihood, although N rather than N_p is used in (4), as the first term in (4) is constant relative to λ . In contrast, when using the restricted maximum likelihood, the profiled likelihood is modified by adding $-\frac{1}{2} \sum_{i=1}^M \log \|(\mathbf{X}^*)^T \mathbf{X}^*\|$ to (4), which will change the dependency from λ and the parameters $\hat{\lambda}$ and $\hat{\beta}$ as well. For general and mixed linear models, the **nlme** package uses, by default, the restricted maximum likelihood method.

If the co-variance matrix only depends on the variance σ^2 because Λ is a constant matrix, the equations (3) and (4) will attain their maximum at the same values of β . Thus, the estimates of the fixed parameters β will be identical regardless of whether σ is kept fixed or estimated and only the standard errors of the fixed coefficients will change.

The linear mixed model

The extension of the general linear model to a mixed linear model complicates the likelihood and subsequently, the estimation of its parameters. The model is known as the Laird-Ware model (Laird and Ware, 1982), but the original proposal for solving the equations is credited to Harville (Harville, 1977). If M independent groups and Q hierarchical levels are above the basic null level, a general mixed model complicates the presentation of its expression. As an illustration, a linear mixed model with one hierarchical random level only is presented here:

$$y_i = \mathbf{X}_i\beta + \mathbf{Z}_ib_i + \varepsilon_i \quad b_i \sim N(0, \Psi) \quad \varepsilon_i \sim N(0, \sigma^2\Lambda_i) \tag{5}$$

The matrices \mathbf{X}_i and \mathbf{Z}_i are the design matrices for p fixed parameters and q_1 random effects, respectively. The fixed parameters and random effects are vectors of length p and q_1 respectively, while y is a column vector of length n_i . A model with Q random levels would have $[q_j, j = 1, \dots, Q]$ random effects in total. It is assumed that b_i and ε_i are independently normally distributed with a mean of zero. The covariance matrix between each of the separate observations - sometimes called the basic random level - is similar to the one in (1), allowing different dependencies within this basic random level; e.g., correlations over time of observations grouped by subjects. The covariance matrix Ψ is the same for all groups. The latter may depend on a small number of parameters θ similar to the covariance matrices Λ_i , which depend on parameters λ . The parameters θ are called random coefficients. In (5) the expectation of b_i is assumed to be zero. The latter presents no restriction, as the expectations can be part of the fixed model parameters β_i . The expression (5) is redundant in the sense that the matrices Ψ and Λ_i could have been combined, but the previous formulation is mathematically more convenient. Likewise, in equation (2), an appropriate transformation of both sides of the expression will change the model into a standard linear mixed model with independent errors conditionally on the unknown parameters λ and θ .

A particularity of model (5) is that the vectors b_i are neither observed nor are parameters of the model itself. In analogy to Bayesian modelling, the effects b_i are integrated out of the formula (5). The resulting likelihood depends only on $\beta, \sigma^2, \theta,$ and λ . The mathematical difficulty is that the random parameters θ are unknown and moreover the integration is multidimensional. Once the random parameters including σ^2 are estimated, these are plugged into the model to estimate the fixed parameters, similar to the general linear model. This approach is also known as empirical Bayesian estimation of a mixed model. In contrast to other approaches used in PROC MIXED and PROC GLIMMIX from SAS, the algorithm in **nlme** does not perform the integration numerically. In contrast, Pinheiro uses an analytical approach and integrates ‘manually’, which results in a solution exclusively in terms of ordinary matrix algebra, as in the general linear model (Pinheiro and Bates, 2001). In theory, this algorithm should be faster and more stable than that of PROC MIXED or PROC GLIMMIX from statistical analysis software (SAS). Stability in the sense of convergence depends on the model itself and is changeable. The drawback of the algorithm used by Pinheiro is that its derivation depends completely on the use of normal distributions, while the SAS algorithm can more easily be adapted for other distributions (such as binomial or Poisson). Both of the SAS procedures allow a full Bayesian solution by Monte Carlo Markov Chain (MCMC). Although b_i are not formally part of the model, these may be estimated and are called the random residuals of the model for grouped observations (e.g., subjects) and might be used to detect outlying groups.

In the following example, the (minus) log-likelihood of the standard mixed model for one hierarchical level ($Q = 1$) is compared to its equivalent of the model with known σ . For clarity of notation, the superscript * is excluded from all design matrices, but one must bear in mind that the latter matrices now depend on the random parameters $\hat{\theta}$ as well as $\hat{\lambda}$. Thus changing the estimates of the latter parameters will change all the others as well. Similar to the general linear model, the use of the normal or restricted maximum likelihood will change the parameters as well, but for mixed models also the fixed parameters. As noted in Pinheiro and Bates (Pinheiro and Bates, 2001), the profiled (minus) log-likelihood is formulated as:

$$llik(\lambda, \theta | y, \sigma) = \frac{N_p}{2} \left[\log(N_p) - \log(2\pi) - 1 - \sum_{i=1}^M \log \left(\|\mathbf{c}_{-1}^i\|^2 \right) \right] + \sum_{i=1}^M \log \left| \frac{\Delta}{\mathbf{R}_i} \right|$$

The vectors \mathbf{c}_{-1}^i and the matrices \mathbf{R}_i are the result of a so-called QR decomposition of the design matrices \mathbf{X}_i . See Appendix .2 for details of the properties of QR decomposition. The matrices Δ are relative precision factors, which may actually be a vector of scalars. In Pinheiro and Bates (Pinheiro and Bates, 2001) the precision factors are defined as any matrix Δ that satisfies

$$\sigma^2 \Psi^{-1} = \Delta^T \Delta \tag{6}$$

Analogous to (4) the profiled (minus) log-likelihood can be expressed as:

$$llik(\lambda, \theta | y, \sigma) = \frac{N_p}{2} \left[-\log(2\pi) - \log(\sigma^2) \right] - \frac{\sum_{i=1}^M \|\mathbf{c}_{-1}^i\|^2}{2\sigma^2} + \sum_{i=1}^M \log \left| \frac{\Delta}{\mathbf{R}_i} \right| \tag{7}$$

The profiled log-likelihood with fixed σ depends linearly on a term that is similar to the residual sum of squares, while the equation of the usual mixed model depends on the log of the latter quantity. Again, N_p with the maximum likelihood of estimation equals N , while for restricted maximum likelihood (REML), $N_p = N - p$. In case of REML-estimation equation (7) is augmented by: $\log(|\mathbf{R}_{00}|) = 0.5 \log \left(\left| \sum_{i=1}^M \mathbf{X}_i^T \Sigma_i^{-1} \mathbf{X}_i \right| \right)$ where the matrix \mathbf{R}_{00} is part of the QR decomposition. Maximizing the augmented equation causes different estimates of the random parameters especially as compared to those from the maximum likelihood method (ML). The method of estimation by lme is, by default, the restricted maximum likelihood.

The non-linear mixed model

The non-linear mixed model is not only more complicated, its model structure is also different. In a linear model, the expectation of response is directly modelled as a linear combination of the row vectors of the design matrix \mathbf{X} . The fixed coefficients of the model define these linear combinations, while the random coefficients weight the latter for each group of the hierarchical level. By contrast, in the non-linear models, the expectation of the response is a non-linear function and each of its arguments is given by a mixed model. The non-linear model is described in detail by Davidian and Giltinan (Davidian and Giltinan, 1995) and its algorithms in Bates and Chambers (Bates and Chambers, 1988), (Bates and Watts, 1992), (Lindstrom and Bates, 1990) and Pinheiro and Bates (Pinheiro and Bates, 2001). The inclusion of the random parameters considerably complicates maximizing the log likelihood. In this section, the model and the profiled likelihood are outlined. For details and alternative algorithms for the same model, see Pinheiro and Bates (Pinheiro and Bates, 2001). As a point of departure, a single hierarchical level, repeated measures, non-linear model is explained next. For each of $i = 1, \dots, M$ groups and $j = 1, \dots, n_{ij}$ observations within a group, we have:

$$\begin{aligned} y_{ij} &= f(\phi_{ij}, v_{ij}) + \varepsilon_{ij} \quad \varepsilon_{ij} \sim N(0, \sigma^2 \mathbf{I}) \\ \phi_{ij} &= \mathbf{A}_{ij} \beta + \mathbf{B}_{ij} b_i \quad b_i \sim N(0, \Psi) \end{aligned}$$

Again, the vector β are fixed parameters and the vectors b_i are the random effects. The matrices \mathbf{A}_{ij} and \mathbf{B}_{ij} play a similar role as \mathbf{X}_i and \mathbf{Z}_i , respectively, in (5). The symbol f stands for a continuous non-linear differentiable function of one dimension, and v_{ij} is a vector of covariates. The model assumes that for each group (e.g., subjects), the arguments of the function may be different but its form is the same for all groups. The model can be straightforwardly extended for more levels of the hierarchy, or even by omitting the random terms as in gnls.

The nlme algorithm consists of alternating between a penalized non-linear least squares step (PNLS) and the linear mixed model step (NLM), which is similar and explained in the linear mixed model section. Therefore, only the consequences of fixing σ in the PNLS step have to be considered. The PNLS step minimizes:

$$\sum_{i=1}^M \left[\|y_i - f_i(\beta, b_i)\|^2 + \|\Delta b_i\|^2 \right] \tag{8}$$

The minimum is found by keeping in each step the precision factor (6) and the random coefficients b_i fixed. A convenient technique is to transform (8) into an ordinary least squares problem by augmenting the response for each group with a vector of 0's and the vector of function values by the column vector Δb_i . The sum of squares (8) is then identical to

$\sum_{i=1}^M \left[\|\tilde{y}_i - \tilde{f}_i(\beta, b_i)\|^2 \right]$ with \tilde{y}_i and \tilde{f}_i are the augmented vectors. Thus in the PNLS step, the sum of squares is conditional on the value of σ and either estimation in the mixed model step or substitution with a prior known value will yield the same solution. Only the likelihood function in the NLM step is affected by the choice of substitution of a known σ or one that is estimated. For the non-linear mixed model algorithm within each PNLS step, the same correction can be used as in the linear mixed model. A strong point of the algorithm devised by Pinheiro for the (non) linear mixed model is the strategy of getting solutions, not only by the classical root finding of the derivatives of the likelihood, but also by means of the expectation-maximization (EM)-algorithm. The latter does not require any derivatives. The EM algorithm is powerful because starting values of the parameters may be far from optimal

Model type	nlme Function	Data set	Model name	Model function	Reference
Generalized linear	gls	Orthodont	fm1Orth.gls		p. 251
Non-linear	gnls	Dialyzer	fm1Dial.gnls	SSasymptOff	p. 402
Mixed linear	lme	Orthodont	fm1Orth.lme		p. 147
	lme	Orthodont	fm3Orth.lme		p. 177
	lme	Wafer	fm1Wafer.lme		p. 172
Mixed non-linear	n1me	Theophylline	fm1Theo.nlme	SSfol	p. 363, 516
	n1me	Theophylline	fm2Theo.nlme	SSfol	p. 364, 516
	n1me	Theophylline	fm3Theo.nlme	SSfol	p. 365, 516
	n1me	Orange	fm1Oran.nlme	SSlogis	p. 358, 519

Table 1: Examples used for comparison of R and S-Plus output with and without fixed σ . References are to [Pinheiro and Bates \(2001\)](#).

values. As on the other hand convergence slows when actually approaching the optimal values, the root finding takes over. Again, this property makes the algorithm quite robust in theory, though in practice, different starting values may cause differences in estimated values. Another difference between the algorithm of R and SAS is that in the former, the likelihood is re-parameterized using $\log(\sigma)$ rather than in σ , thus preventing the algorithm to crash if $\hat{\sigma}$ is close to zero. This problem is encountered frequently for over-parameterized models when using SAS. The **nlme** package contains a function `gnls` for the general non-linear model analogous to the general linear model function `gls`. It uses basically the same algorithm as `n1me` by stripping the random component.

Implementation and examples

From the viewpoint of the user, the only change in a function call of `gls`, is an extra argument in the control functions `glsControl`, `gnlsControl`, `lmeControl` and `n1meControl` respectively. The option is implemented exactly as the corresponding control functions in S-Plus; for example, using a simple model and data from the S-Plus and R libraries. The output object of the functions `lme`, `n1me`, `gls`, and `gnls` contained a logical flag as component to signal whether σ is either fixed or estimated. The flag is used in the utility functions `print`, `summary`, `anova`, and `intervals`. Table 1 gives an overview of the examples used to test the new feature. The former are taken from [Pinheiro and Bates \(2001\)](#). All examples ran both in R and S-Plus.

In the next subsections the output from package **nlme** of R and library `nlme` of S-Plus are compared. Striking differences are given in the discussion. All scripts, output and difference files- in HTML format- are made available to the editors of the journal. Not all models converged. They were either in both R and S-Plus or in R or S-Plus only. The package **nlme** has a choice for two different optimizing algorithms but only the default has been used as this is the only one available in S-Plus. Thus, non-convergence in R might be caused by the use of the default algorithm. However, this has not been tested systematically. The same starting values have been used in both R and S-Plus, except in the examples with the self-starting functions in **nlme**. For the latter, starting values had to be supplied. The same examples for estimated σ have been run both for the original **nlme** version and the updated version. As the output was completely the same, eventual differences between S-Plus and R in case of estimated σ cannot be attributed to the update of the package. We have chosen to compare the R-output and the actual output of S-Plus 8.2 rather than the published one ([Pinheiro and Bates, 2001](#)), as in some cases, differences in the case of estimated σ were found between the latter and the actual version of S-Plus 8.2. At the time of its publication, fixed σ was not yet implemented.

General linear and general non-linear models

As an example, the code for the general linear and general non-linear model are given below for fixed σ .

```
fm1Orth.gls <- gls( distance ~ Sex * I( age - 11), Orthodont)
```

While the same model with a fixed $\sigma = 2$ is declared as:

```
fm1Orth_fix.gls <- gls( distance ~ Sex * I( age - 11), Orthodont,
                      control = glsControl(sigma = 2)).
```

The default estimates σ and the argument `sigma` to `glsControl` or `gnlsControl` (below) can be omitted or set to 0 or NULL. For both estimated and fixed σ , the results from the R and S-Plus estimates of the

coefficients were identical to the third decimal using either the estimation method of the maximum likelihood (ML) or the restricted maximum likelihood method (REML). However, for both the default of estimated σ and fixed σ , the equivalence of the correlation structure and the confidence limits of the coefficients might be limited only to the second decimal. An example of the general non-linear model is scripted as:

```
fm1Dial.gnls <- gnls( rate ~ SSasymOff( pressure, Asym, lrc, c0), Dialyzer,
  params = list( Asym + lrc ~ QB, c0 ~ 1),
  start = c(53.6, 8.6, 0.51, -0.26, 0.225),
  control = gnlsControl(sigma = NULL))
```

For fixed σ the argument `sigma` to the `gnlsControl` function should be changed to 1. The function `gnls` can only produce a least squares solution and rarely uses the C-library. Note that although the function `SSasymOff` is a self-starting function, starting values had to be supplied in R, but not in S-Plus.

Mixed linear models

The `gls` example from above was also fitted as a mixed model with the appropriate changes in the script:

```
fm10rth.lme <- lme( distance ~ I(age-11), Orthodont)
```

and

```
fm10rth_fix.lme <- lme( distance ~ I(age-11), Orthodont,
  control = lmeControl(sigma = 1))
```

Similar results as for the `gls` model were found for the ML method. However, both in R and S-Plus the REML estimation method for both estimated and fixed σ stopped due to a convergence error. The latter is remarkable as the estimates for the REML method are given in Pinheiro and Bates (Pinheiro and Bates, 2001)). The third example uses a covariance structure and is given as:

```
fm30rth.lme <- lme( distance ~ Sex * I( age - 11), Orthodont,
  weights = varIdent(form = ~ 1 | Sex))
```

While the same model with a fixed $\sigma = 2$ is declared as:

```
fm30rth_fix.lme <- lme( distance ~ Sex * I( age - 11), Orthodont,
  weights = varIdent(form = ~ 1 | Sex),
  control = list(sigma = 2)).
```

The ML method of estimation for both estimated and fixed σ were equal in the third digit, with the exception of the random effects and the interval estimates, in both cases of estimated and fixed σ . Differences were sometimes found to the second decimal. The same applies to the method of REML estimation. Finally, the corresponding R code for the linear mixed model example:

```
fm1Wafer.lme <- lme( current ~ voltage + I( voltage^2 ), Wafer,
  random = list( Wafer = pdDiag( ~ voltage + I( voltage^2 ) ),
  Site = pdDiag( ~ voltage + I( voltage^2 ) ) ),
  control = lmeControl( sigma = NULL))
```

For fixed σ , one has to substitute NULL by 1 in the argument `sigma` of `lmeControl`. Similar results are found in the two previous examples. But notable differences in the confidence limits of the random effects were seen when using the REML method with fixed σ , although the point estimates of the coefficients themselves were the same. For the interpretation of the correlation between the random effects, this might have consequences, as in R the latter did not contain 0 in the interval as opposed to the result from S-Plus 8.2. The latter could be caused by the difference in the degrees of freedom of the t-distribution, as our implementation of these might have changed if σ is kept fixed.

Non-linear mixed models

The `nlme` function for solving a non-linear problem defaults to the ML method as opposed to `lme`. During testing it was discovered that the option of the restricted maximum produced exactly the same estimates as for ML, with and without estimation of σ . In our opinion, this is not correct, but as far as we know, has never been reported. The example codes as:

```
fm1Theo.nlme<-nlme(conc ~ SSfol(Dose, Time, lKe, lKa, lCl),
  Theoph, fixed = lKe + lKa + lCl ~ 1,
  start=c( -2.4, 0.45, -3.2),
  control = nlmeControl( sigma = NULL))
```

Fitting σ requires the same change as in the above examples. The ML estimates using estimated σ were the same for R and S-Plus to the third digit. However, for fixed σ the example in S-Plus did not converge, and no comparison can be made. For the REML method, neither of the estimations (free or fixed) converged. R required starting values, even if SSfol was supposed to be self-starting. The next example from Table 1 is an extension of the above and the code is:

```
fm2Theo.nlme<-nlme(conc ~ SSfol(Dose, Time, lKe, lKa, lCl),
  Theoph, fixed = lKe + lKa + lCl ~ 1,
  start=c( -2.4, 0.45, -3.2),
  control = nlmeControl( sigma = NULL),
  random = pdDiag( lKe + lKa + lCl ~ 1))
```

The estimates of the fixed coefficients are equal to the second or third place in all cases. In contrast differences between the random parameters for ML, REML, and estimated and fixed σ are larger. Also, the approximate variance-covariance matrix of the random effects is nearly singular, which causes the 95% lower and upper limits for lKe to be near infinity. In S-Plus, the covariance matrix of the REML random estimates was not estimated at all due to singularity of the Hessian matrix. This may indicate that the parametrization of model is not correct, probably for parameter lKe. The next example uses the same data set but omits lKe from the random parameter list:

```
fm2Theo.nlme<-nlme(conc ~ SSfol(Dose, Time, lKe, lKa, lCl),
  Theoph, fixed = lKe + lKa + lCl ~ 1,
  start=c( -2.4, 0.45, -3.2),
  control = nlmeControl( sigma = NULL),
  random = pdDiag( lKa + lCl ~ 1))
```

Now the outcomes of all the models are in better agreement, and the covariance of the random effects is available for S-Plus. There are still differences from the third digit onwards for the random effects confidence limits, especially for the REML method.

Note that for all of the above models we have changed the degrees of freedom of the log likelihood in case σ is held fixed, which coincides with the output of S-Plus. This may cause differences in the confidence limits especially when the degrees of freedom of the t-distribution is small.

A case study illustrating the application of a non-linear mixed model with a binomial response

A summary clinical outcome dataset including clinical outcome data from 14 randomized controlled trials of 4 drugs was extracted from the Quantify RA clinical database developed by Quantitative Solutions (drug and trial names blinded). Three endpoints were included describing the proportion of patients with ACR20, ACR50, and ACR70 responses. These endpoints are based on the American College of Rheumatology (ACR) criteria used to assess improvement in tender or swollen joint counts and improvement in three of the following five parameters: acute phase reactant (such as sedimentation rate), patient and physician assessment, pain scale, and a disability/functional questionnaire. For ACR20 response, a 20 percent improvement in tender or swollen joint counts as well as 20 percent improvement in three of the other five criteria is required. Similarly, for ACR50 and ACR70, a 50 and 70 percent improvement is required, respectively. Outcomes were available up to 6 months (24 weeks) after the start of treatment with one of the drugs, up to 12 weeks with the two other drugs, while the last drug was only observed in a 4-week trial. The analysis dataset was analyzed using two non-linear regression models and a mixed effects model with random effects on the level of trial assuming binomial distributed data. The probability of ACR20/50/70 response is confined to the interval between zero and one. In this example, the logit transformation was used as a link function between the probability of response p and a hidden model variable X :

$$p = \frac{e^X}{1 + e^X}$$

Parameter	Estimate (%RSE)	
	Estimated σ	Fixed $\sigma = 1$
E_0	-1.06 (10%)	-1.11 (4%)
α_{ACR50}	-1.05 (5%)	-1.07 (3%)
α_{ACR70}	-1.97 (3%)	-2.00 (2%)
E_{MAX}	2.02 (10%)	2.06 (4%)
$\log ED_{50,drug1}$	2.22 (63%)	2.14 (31%)
$\log ED_{50,drug2}$	1.75 (20%)	1.68 (9%)
$\log ED_{50,drug3}$	4.11 (16%)	4.04 (8%)
$\log ED_{50,drug4}$	1.22 (36%)	1.22 (15%)
Res. error	1.924	1 (fixed)

Table 2: Parameter estimates with uncertainty based on a gnls model with constant placebo. Results for estimated and fixed σ in left and right column respectively.

A convenient property of the logit transformation is that the log odds ratio (OR) of two probabilities translates into the difference between the corresponding hidden model variables:

$$\log OR = \log \left(\frac{\frac{p_1}{1-p_1}}{\frac{p_2}{1-p_2}} \right) = X_1 - X_2$$

If X is described by a linear model, correction for any predictor can be done by subtraction. Specifically, if the probability of response is modelled as the inverse logit of the sum of a placebo (E_0) and a drug effect (g), the log OR of the treated versus the untreated response is simply given by function g ; (see 11). The logit transformed observed proportion of ACR20/50/70 responders was described as the sum of an unstructured placebo response ($E_{0,it}$) and drug-dependent drug effects described by EMAX models with common E_{MAX} and drug-specific ED_{50} estimates.

$$N_{response,kijt} \sim \text{binomial} \left(P(response)_{kijt}, N_{kijt} \right) \tag{9}$$

$$P(response)_{kijt} = \text{logit}^{-1} \left(E_{0,it} + g_k \left(Drug_{ij}, Dose_{ij} \right) + \alpha_k \right) \tag{10}$$

$$g_k \left(Drug_{ij}, Dose_{ij} \right) = \frac{E_{MAX} \cdot Dose_{ij}}{ED_{50,Drug_{ij}} + Dose_{ij}} \tag{11}$$

Where $N_{response,kijt}$ is the number of subjects in treatment arm j of trial i with ACR_k response at the t_{th} visit out of N_{kijt} subjects. The coefficient α_k estimated for ACR50 and ACR70 describes the constant log OR of these endpoints relative to ACR20. Three variations of the model described in (11) were investigated using fixed and estimated residual standard error. Model 1 was a gnls model including a constant placebo model across trials and visits; i.e. $E_{0,it} = E_0$. Model 2 was a gnls model including an unstructured placebo model estimating a non-parametric placebo response at each visit in each trial; $E_{0,it}$ as shown in (11). Model 3 was a non-linear mixed effects model including an unstructured placebo model with uncorrelated random effects on E_{MAX} and α_k on the level of trial. All three models included a compound symmetry model correlating outcomes from different endpoints and visits within a trial. An overview of the parameter estimates for the three models are in Tables 2, 3, and 4 respectively.

It is interesting to note that comparison with anova showed significant differences between free estimated and fixed σ for models GNLS models 1 and 2, whereas the same comparison for the nlme model 3, did not show a significant difference. The (lack of) differences appeared for the criteria AIC and BIC, as well for the likelihood ratio test. For the former criteria the usual difference of 2 was used as a threshold. See Tables 5, 6 and 7. Comparison between models 1, 2 and 3 are not shown but from the tables it is clear that these models have an increasingly lower AIC and BIC, and better log likelihood ratio's either for free or fixed σ .

Conclusions and discussion

The estimates of the fixed coefficients of S-Plus and R are often identical to three significant digits, except in the case of over-parameterized models, which may cause non-convergence and inaccurate estimation of the covariance matrix of the random parameters. Differences might occur even for the default model with estimation of σ . For the non-linear mixed models differences between R and

Parameter	Estimate (%RSE) Estimated σ	Estimate (%RSE) Fixed $\sigma = 1$
Mean* E_0	-1.16	-1.22
α_{ACR50}	-1.08 (3%)	-1.09 (3%)
α_{ACR70}	-2.05 (2%)	-2.06 (2%)
E_{MAX}	2.24 (8%)	2.26 (5%)
$\log ED_{50,drug1}$	2.87 (69%)	2.80 (50%)
$\log ED_{50,drug2}$	1.94 (13%)	1.92 (9%)
$\log ED_{50,drug3}$	4.47 (21%)	4.42 (15%)
$\log ED_{50,drug4}$	1.66 (28%)	1.44 (22%)
Res. error	1.313	1 (fixed)

E_0^* : Mean of 34 unstructured placebo parameters

Table 3: Parameter estimates with uncertainty based on a gnl model with unstructured placebo. Results for estimated and fixed σ in left and right column respectively.

Parameter	Estimate (%RSE) Estimated σ	Estimate (%RSE) Fixed $\sigma = 1$
Mean* E_0	-1.28	-1.26
α_{ACR50}	-1.08 (8%)	-1.08 (8%)
α_{ACR70}	-2.08 (5%)	-2.08 (5%)
E_{MAX}	2.34 (11%)	2.33 (11%)
$\log ED_{50,drug1}$	2.97 (50%)	2.99 (54%)
$\log ED_{50,drug2}$	1.78 (10%)	1.76 (11%)
$\log ED_{50,drug3}$	4.66 (16%)	4.67 (17%)
$\log ED_{50,drug4}$	1.59 (33%)	1.71 (32%)
$sd(\alpha_{ACR50})$	0.260	0.257
$sd(\alpha_{ACR70})$	0.351	0.344
$sd(E_{MAX})$	0.761	0.745
Res. error	0.938	1 (fixed)

E_0^* : Mean of 34 unstructured placebo parameters

Table 4: Parameter estimates with uncertainty based on a nlme model with unstructured placebo. Results for estimated and fixed σ in left and right column respectively.

Model	df	AIC	BIC	logLik	L.Ratio	p-value
model 1 free σ	10	-740.2	-700.1	380.1		
model 1 fix σ	9	-271.1	-235.0	144.5	471.2	<.0001

Table 5: Analysis of variance table for comparison of gnl model 1 between free and fixed σ

Model	df	AIC	BIC	logLik	L.Ratio	p-value
model 2 free σ	43	-1100.5	-928.0	593.3		
model 2 fix σ	42	-1076.8	-908.4	580.4	25.7	<.0001

Table 6: Analysis of variance table for comparison of gnl model 2 between free and fixed σ

Model	df	AIC	BIC	logLik	L.Ratio	p-value
model 3 free σ	46	-1215.8	-1031.3	653.9		
model 3 fix σ	45	-1215.8	-1035.3	652.9	2.0	0.16

Table 7: Analysis of variance table for comparison of nlme model 3 between free and fixed σ

S-Plus were more likely to occur when using REML estimation. Sometimes REML estimation was not possible in either R or S-Plus due to lack of degrees of freedom.

There are some differences between R and S-Plus in the code of the functions. As the code of the C-functions in S-Plus is not public, differences in the code itself could not be checked. The R functions, in particular the interaction between the R-code and the C-functions were fragile in the sense that input was often changed within a function and then used as output. Thus, a small change in code in one part could have unexpected results in other parts of the output.

The code of the **nlme** package is far from modular; e.g., parts of both user-accessible and hidden functions of `lme` were used in `nlme` or `gls` causing lack of transparency. For example, within the function `print.summary.gls`, at some point the function `print.summary.lme` was used, which caused difficulties with passing the right degrees of freedom of the likelihood. Different definitions of the likelihood functions coded in C were used in the R-code of the same parent functions, amongst others in `lme`. Another critical point is that utilities as `print`, `summary` or `print.summary` functions tend to recalculate again some statistics such as the likelihood, AIC or BIC. For these reasons, debugging has been time consuming and given the complexity of the **nlme** package the number of tests provided by CRAN is simply too small. Thus, the warning ‘the use of free software comes with no guarantee’ in the code of each of the functions must be taken in earnest.

A general drawback of the algorithm used in **nlme** is that it is completely geared to the use of the normal distribution. The latter makes it difficult - though not impossible - to adapt the algorithm for other distributions such as binomial or Poisson. The CRAN-library already contains the package **nlme4** allowing non-normal distributions, but at this time lacks the feature of the use of correlated observations. Perhaps a future update will be available that contains an option for fixing the scale parameter σ .

In the `nlme` library of S-Plus the degrees of freedom for residual error and consequently, the t-tests, are not properly adapted for fixed σ . However, in the function `anova`, the degrees of freedom has been changed in S-Plus to allow comparison between models with and without fixed σ . This correction affects both AIC and BIC. For example, AIC will always decrease by 2 while BIC will decrease by $2 \times \log(N)$. Apart from that, the value of the profiled log likelihood for the REML estimation should change, because the degrees of freedom is part of the latter function. For reasons of compatibility with S-Plus, the changes in the degrees of freedom in the t-tests have not been implemented in **nlme** of R.

When comparing the output of `nlme` between R and S-Plus, a flaw became apparent in the output of `anova` if used with a single object. The F-tests for the (grouped) parameters differed between both programs. The F-values from R can be made equivalent to those from S-Plus by multiplying the F-values by $\frac{N}{N-p}$. The same applies to the standard errors of the fixed estimates. There the factor should be $\sqrt{\frac{N}{N-p}}$. We believe that the output from S-Plus is correct considering that a t-test with one degree of freedom is equivalent to an F-test with one degree in the numerator in which case, the latter is equivalent to the square of the former. The finding that, in the function `nlme` in R, that there is no difference between ML and REML estimation is remarkable and may be due to an error in the code or to a compelling reason by the maintainer of the package.

From a statistical point of view, a warning should be issued on the use of the likelihood ratio test when comparing two models; one with a fixed and one with an estimated σ . In fact, the usual F-test is not appropriate. The reason is that the log of both numerator and denominator have an approximate Chi-square distribution up to the log of the true value of σ , which is unknown if σ is estimated. The difference in the log-ratio between the known and estimated value of σ does not go away, which causes a bias and invalidates the F-distribution of the log ratio. A safe way to test whether the a priori value is right, is to use AIC or BIC using the recommendations by Raftery (Raftery, 1980). As Raftery states, difference between AICs of 2 or less is ‘hardly worth mentioning’. As previously mentioned, an increase of one degree of freedom of the model already causes a decrease of 2 in the AIC, even if the likelihood does not change. One could then be tempted to prefer the model with more degrees of freedom although Raftery seems not to recommend this. The influence on the BIC is even stronger, as its decrease will be at least $2 \times \log(\sigma)$. With the developed patches, we have successfully back-ported the ability to fix the residual standard error in the **nlme** package from S-Plus to R. Following review, the patches have been accepted into the official version **nlme** (available through CRAN). This will allow users to correctly estimate models where the residual standard error is known, using standard and publicly available open-source tools.

Acknowledgements

Special thanks go to Martin Maechler for critically looking at the added code, running the CRAN tests independently and fine tuning our coding. We are greatly indebted to Mrs. Flowers Lovern for scrutinizing the English text, both on syntax and idiom.

De-correlation of the normal model

Starting from a normal model with correlated errors, it is shown below that this model can be converted into a model with independent errors and equal variance. Let Λ be a positive-definitive matrix, then a non-singular square root of that matrix exists, $\Lambda = (\Lambda^{1/2})^T \Lambda^{1/2}$. Similarly, for the inverse of the matrix. The original response vector \mathbf{y} and the errors ϵ are now both multiplied by the inverse of the square root matrix, and are $\mathbf{y}^* = (\Lambda^{-1/2})^T \mathbf{y}$ and $\epsilon^* = (\Lambda^{-1/2})^T \epsilon$ respectively. The transformed design matrix is $\mathbf{X}^* = (\Lambda^{-1/2})^T \mathbf{X}$. The covariance matrix of the transformed observations is then $V(\epsilon^*) = \sigma^2 (\Lambda^{-1/2})^T \Lambda \Lambda^{-1/2} = \sigma^2 \mathbf{I}$, and the fixed coefficients β of the model can now be solved with the usual least squares method.

QR decomposition

A $N \times p$ matrix \mathbf{X} of rank p may be decomposed in an orthogonal matrix \mathbf{Q} of dimension $n \times n$ and a $p \times p$ upper triangular matrix \mathbf{R} . More specifically

$$\mathbf{X} = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix} = \mathbf{Q}_t \mathbf{R}$$

The matrix \mathbf{Q}_t is the truncated matrix \mathbf{Q} consisting of the first p columns of the latter matrix. An orthogonal matrix is defined as $(\mathbf{Q}_t)^T \mathbf{Q}_t = \mathbf{I}$. The decomposition is performed in the model (5) for all matrices to simplify the actual numerical computations as well as the representation of the estimation equations. The reason is that multiplication of a vector y augmented with p 0's conserves the Euclidian norm of that vector. So $\|(\mathbf{Q}_t)^T y\|^2 = \|y\|^2$. Similarly, the vector of residuals of a regression model $\|y - \beta \mathbf{X}\|^2$ is conserved and equals $\|c_1 - \mathbf{R} \beta\|^2 + \|c_2\|^2$. The vector $\mathbf{c}^T = (c_1, c_2)$ with length p and $n - p$ and equals $(\mathbf{Q}_t)^T y$. Maximizing the corresponding log likelihood yields the least squares estimator $\hat{\beta} = \mathbf{R}^{-1} c_1$ and the residual sum of squares equals $\|c_2\|^2$.

Bibliography

- H. Akaike. Likelihood and the Bayes procedure. In J. M. Bernardo, M. H. DeGroot, D. V. Lindley, and A. F. M. Smith, editors, *Bayesian Statistics*. University Press, Valencia, Spain, 1980. [p239]
- D. M. Bates and J. M. Chambers. *Nonlinear Regression Analysis and Its Applications*. John Wiley & Sons, New York, U.S.A., 1988. [p243]
- D. M. Bates and D. G. Watts. Nonlinear models. In C. J.M. and H. T.J., editors, *Statistical Models in S*, pages 421–454. Wadsworth and Brooks, Pacific Grove, U.S.A., 1992. ISBN 0 534 16764 0. [p243]
- S. Beal and L. Sheiner. The NONMEM system. *The American Statistician*, 34:118–119, 1988. [p239]
- M. Davidian and D. M. Giltinan. *Nonlinear Models for Repeated Measurement Data*. Chapman and Hall, London, U.K., 1995. ISBN 0 412 98341 9. [p239, 240, 243]
- D. Harville. Maximum likelihood approaches to variance component estimation and to related problems. *Journal of the American Statistical Association*, 72:320–340, 1977. [p242]
- N. M. Laird and J. H. Ware. Random-effects models for longitudinal data. *Biometrics*, 38:963–974, 1982. [p239, 240, 242]
- M. J. Lindstrom and D. M. Bates. Nonlinear mixed-effects models for repeated measures data. *Biometrics*, 46:673–687, 1990. [p243]
- R. C. L. Littell, G. A. Milliken, W. W. Stroup, R. D. Wolfinger, and O. Schabenberger. *SAS for Mixed Models, Second Edition*. SAS Institute, 2006. ISBN 1590475003. [p239]
- J. C. Pinheiro and D. M. Bates. *Mixed Models in S and S-Plus*. Springer-Verlag, New York, U.S.A., 2001. ISBN 0 387 98957 9. [p239, 240, 241, 242, 243, 244, 245]
- A. E. Raftery. Hypothesis testing and model selection. In W. R. Gilks, S. Richardson, Spiegelhalter, and D.J., editors, *Markov Chain Monte Carlo In Practice*, pages 163–186. Chapman and Hall, London, U.K., 1980. ISBN 0 412 05551 1. [p249]

- G. Schwartz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978. [p239]
- S. N. Wood. *Generalized Additive Models, An Introduction with R*. Chapman and Hall, London, U.K., 2006. ISBN 1 58488 474 6. [p240]

S.H. Heisterkamp
BioStat-Plus B.V.
Mariaplaats 32E, 3511LL Utrecht
the Netherlands
info@biostat-plus.nl

E. van Willigen
Van Willigen ICT-consultancy
Kempke 19, 5672PL Nuenen
the Netherlands
e.vanwilligen@onsnet.nu

P.M Diderichsen
Quantitative Solutions B.V.
Parkstraat 1b, 4818SJ Breda
the Netherlands
pmdiderichsen@wequantify.com

J. Maringwa
Quantitative Solutions B.V.
Parkstraat 1b, 4818SJ Breda
the Netherlands
jmaringwa@wequantify.com

EMSaov: An R Package for the Analysis of Variance with the Expected Mean Squares and its Shiny Application

by Hye-Min Choe, Mijeong Kim, and Eun-Kyung Lee

Abstract EMSaov is a new R package that we developed to provide users with an analysis of variance table including the expected mean squares (EMS) for various types of experimental design. It is not easy to find the appropriate test, particularly the denominator for the F statistic that depends on the EMS, when some variables exhibit random effects or when we use a special experimental design such as nested design, repeated measures design, or split-plot design. With EMSaov, a user can easily find the F statistic denominator and can determine how to analyze the data when using a special experimental design. We also develop a web application with a GUI interface using the shiny package in R. We expect that our application can contribute to the efficient and easy analysis of experimental data.

Introduction

The analysis of variance (ANOVA) is a well-known method that can be used to analyze data obtained with different experimental designs. Its use mainly depends on the primary design of the experiment, and the main testing method for the analysis of variance is the F test. If all factors are fixed effects and there is no specific design of the experiment, that is, the experimental design is a factorial design, then the usual way to calculate the F statistic is to use the mean squares of the corresponding source of variation as the numerator and the mean squares of errors as the denominator. However, if some variables exhibit random effects or some variables are nested in the other variables, it is not easy to find the appropriate F statistic. This depends on the expected mean square (EMS), and the denominator of the F statistic is determined by the EMS of the corresponding source of variation. Therefore, we first have to calculate the expected mean squares for the ANOVA and then find the exact F statistic for the test using the EMS, especially when data comes from a special experimental design. Even though the EMS is very important to finding the exact F statistic in the ANOVA, few tools show this EMS and most of the tools that have been developed provide only the result of the ANOVA without any further explanation. Therefore, users cannot figure out how to calculate the test statistics and only know the final result.

Several packages can be used to handle models with various experimental designs. The `lm` function in R can handle factorial design with fixed effects without taking the special experimental design or the random effects into account. The `lme` function in the `nlme` (Pinheiro et al., 2016) package handles the mixed effect model, and in this function, the user can specify the factors with a random effect. However this function mainly focuses on the grouped data and on estimating the variance components instead of testing the corresponding factor. Also, it does not provide the EMS of each source in the ANOVA table.

Another R package that can be used in the analysis of factorial experiments is `afex` (Singmann et al., 2016). This package provides the function `ems` to calculate EMS for the factorial designs. They adapted the Cornfield-Tukey algorithm (Cornfield and Tukey, 1956) to derive the expected values of the mean squares. The `afex` package also provides the `mixed` function to calculate p-values for various ANOVA tables considering the corresponding EMS to find the exact F statistic. However, the `ems` function provides the general information on the factorial design, and the result for `ems` only shows the coefficients of variances. It is provided in a $p \times p$ table form instead of the EMS formula for each source in the ANOVA table, where p is the number of sources, and the elements of this table represent the coefficient of variance in the formula of the EMS. It is not easy to find the corresponding F statistic with this result, and this EMS does not match the analysis of variance for real data due to the use of the number of levels in each factor with characters instead of the real number of levels, so users need to match this character with their own number of levels.

In this paper, we provide a tool to show how to calculate test statistics as well as the final result. We focus on the classical analysis of variance method, based on the F test using EMS, exclusively for balanced designs. We develop a new R package EMSaov to provide users with the analysis of variance table with EMS for various types of experimental design. With the ANOVA table combined with EMS, users can easily understand how to calculate the F statistics, especially the denominator of the F statistic, and then figure out the result of the analysis. We also provide an application for novice users based on Shiny (`shiny`). First, we explain the general concepts of the analysis of variance and the

Source	Df	Fixed model (A,B:fixed)	Random model (A,B:random)	Mixed model (A:fixed, B:random)
A	$a - 1$	$\sigma_\epsilon^2 + nb\phi_A$	$\sigma_\epsilon^2 + n\sigma_{AB}^2 + nb\sigma_A^2$	$\sigma_\epsilon^2 + n\sigma_{AB}^2 + nb\phi_A$
B	$b - 1$	$\sigma_\epsilon^2 + na\phi_B$	$\sigma_\epsilon^2 + n\sigma_{AB}^2 + na\sigma_B^2$	$\sigma_\epsilon^2 + na\sigma_B^2$
AB	$(a - 1)(b - 1)$	$\sigma_\epsilon^2 + n\phi_{AB}$	$\sigma_\epsilon^2 + n\sigma_{AB}^2$	$\sigma_\epsilon^2 + n\sigma_{AB}^2$
Residuals	$ab(n - 1)$	σ_ϵ^2	σ_ϵ^2	σ_ϵ^2

Table 1: Expected mean squares for the three different types of models

special types of experimental designs. Then we introduce **EMSaov**, our newly developed R package, with its implementation, and explain the usage of functions in **EMSaov** in detail. We also introduce the web interface of **EMSaov**, followed by the conclusion.

Analysis of variance

Fixed, random, and mixed models

There are two ways to select the levels of factors for various factorial experimental designs. One is to select the appropriate levels as fixed values, and the other is to choose at random from many possible levels. [Bennett et al. \(1954\)](#) discuss a case in which the chosen levels are obtained from a finite set of possible levels. When all levels are fixed, the statistical model for the experiment is referred to as a fixed model, and when all levels are chosen as random levels, the model is referred to as a random model. When two or more factors are involved and some factors are chosen as fixed levels and the others are chosen as random levels, the model is referred to as a mixed model. There is no difference between the fixed model and the random model during data analysis for a single-factor experiment. However, the EMS for each factor should be different from that of a fixed model if there is more than one factor, some factors exhibit random effects, and the other factors are fixed effects. We thus have to be careful when generating an F statistic to test the significance of each factor.

Consider the two-factor factorial experiment with factors A and B. The corresponding experimental model with a completely randomized design is

$$Y_{ijk} = \mu + A_i + B_j + AB_{ij} + \epsilon_{ijk} \tag{1}$$

where μ is a common effect, A_i represents the effect of the i th level of factor A, and B_j represents the effect of the j th level of factor B, AB_{ij} is the interaction effect of factors A and B, and ϵ_{ijk} represents the random error in the k th observation on the i th level of A and j th level of B, $i = 1, \dots, a, j = 1, \dots, b$, and $k = 1, \dots, n$.

In this model, we assume that μ is a fixed constant, and ϵ_{ijk} is a random variable that follows $N(0, \sigma_\epsilon^2)$. We can assume three cases: the fixed model, the random model, and the mixed model. In a mixed model, we treat factor A as a fixed effect and factor B as a random effect. The main test method for the analysis of variance is an F test. The usual way to calculate the F statistic is to use the mean squares of the corresponding source as the numerator and the mean square error as the denominator. However, if some variables exhibit random effects, it is not easy to find the appropriate denominator for the F statistic. In fact, this depends on the expected mean square (EMS), so we have to calculate the expected mean square for the analysis of variance of the data.

The expected mean squares are different among the three models, and they are represented in Table 1. For all three models, the mean squares error (MS_E) is used as the denominator to test the interaction effect between A and B. For factor A, MS_E is used in the fixed model but MS_{AB} is used in

Source	Mean Sq	Fixed model (A,B:fixed)	Random model (A,B:random)	Mixed model (A:fixed, B:random)
A	MS_A	MS_A / MS_E	MS_A / MS_{AB}	MS_A / MS_{AB}
B	MS_B	MS_B / MS_E	MS_B / MS_{AB}	MS_B / MS_E
AB	MS_{AB}	MS_{AB} / MS_E	MS_{AB} / MS_E	MS_{AB} / MS_E
Residuals	MS_E			

Table 2: F statistics in ANOVA table for the three different models

Source	Df	Sum Sq	Mean Sq	EMS	F statistic
A	$a - 1$	SS_A	MS_A	$\sigma_\epsilon^2 + b\sigma_{B(A)}^2 + ab\sigma_A^2$	$MS_A / MS_{B(A)}$
B(A)	$a(b - 1)$	$SS_{B(A)}$	$MS_{B(A)}$	$\sigma_\epsilon^2 + b\sigma_{B(A)}^2$	$MS_{B(A)} / MS_E$
Residuals	$ab(n - 1)$	SS_E	MS_E	σ_ϵ^2	

Table 3: ANOVA table for the nested design

the other two models. For factor B, MS_{AB} is used in the random model, but MS_E is used in the other two models. The appropriate test statistics for each factor are summarized in Table 2.

EMS rule

As we can see in the review on generating the F statistics for the three different models, the expected mean squares are very important. The previous examples consist of very simple factorial models with only two factors. For complex experimental designs, particularly when using models involving random or mixed effects with nested factors, it is frequently helpful to have a formal procedure to generate the expected mean squares, that is the EMS rules (Montgomery, 2008). The EMS rules are simple and convenient procedures that determine the expected mean squares, and these are also appropriate for manually calculating the expected mean squares for any nested, repeated-measures, or split-plot design. We follow the EMS rules in Montgomery (2008) to generate the expected mean squares in the ANOVA table with various experimental designs.

Nested and nested-factorial design

In the case of experiments with two or more factors without any restriction in the randomization, most experimental designs can be categorized in one of three ways: crossed design, nested design, or nested-factorial design. In this **EMSaov** package, we didn't consider the unbalanced design and the fractional factorial design. The crossed design considers every possible combination of the levels of factors in the model. However, when the levels of one factor are not identical but similar to different levels of another factor, it is referred to as a nested design.

In the nested design, when the levels of factor B are nested under the levels of factor A, the levels of factor B belonging to the first level of factor A are not the same as the levels of factor B in the second level of factor A, as shown in Figure 1. One of the features of this model is the lack of interaction effect between the two factors that are nested, so when the analysis of variance is carried out, the interaction term AB should be pooled to the nested factor $B(A)$. The ANOVA table for this design is shown in Table 3. Thus, the nested design can be extended to a more complex nested design, for example, to a model with another nested factor C under the existing nested factor B. It can also be combined with factorial design - a model with another factor C that is nested in both A and B, while factors A and B are crossed.

Split-plot design

The split-plot design is used when it may not be possible to completely randomize the order of experimentation. In this case, we assume for one factor to be a block. Since a factor treated as a block is restricted during randomization, the effects of the corresponding factor are confounded with the blocks, and it is thus difficult to determine the pure significance of this effect. On the other hand, there is no loss of information for the other factors that are not treated as a block because it is completely randomized under these factors. In this design, two levels of randomization are applied to assign the experimental units to the treatment. The first level of randomization is applied to the whole plot

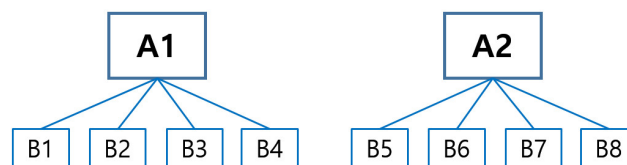


Figure 1: A two-stage nested design

and is used to assign the experimental units to the levels of treatment factor A. The whole plot is split into a split-plot, and the second level of randomization is used to assign the experimental units of the subplot to levels of the treatment factor B. Since the split-plot design has two levels of experimental units, the whole plot and the subplot portions have separate experimental errors. Therefore, the F tests must be run only within the whole plot or within the split plot, and the mean squares in the whole plot should not be compared with the mean squares in the split plot, regardless of the EMS value. We handle this split-plot design as a hierarchical design with respect to the levels of model. The first level of model is the whole plot, and the second level of model is the split plot. These levels of model can then be extended to 3, 4, or more levels.

Approximate F test

In factorial experiments with three or more factors involving a random or mixed model, sometimes there is no exact test statistic for certain effects in the model. We have to calculate a new F statistic if there is no denominator that differs from the expected value of the numerator only by the specific component being tested. Therefore, [Satterthwaite \(1946\)](#) proposed a test procedure that uses linear combinations of the original mean squares to form the F statistic, for example,

$$MS_{num} = MS_{num,1} + \dots + MS_{num,r_n} \tag{2}$$

$$MS_{den} = MS_{den,1} + \dots + MS_{den,r_d}$$

where $MS_{num,1}, \dots, MS_{num,r_n}$ and $MS_{den,1}, \dots, MS_{den,r_d}$ are selected from MS values in ANOVA table such that $E(MS_{num}) - E(MS_{den})$ is equal to the effect considered in the null hypothesis.

Then,

$$ApproxF = \frac{MS_{num}}{MS_{den}} \sim F_{df_{num}, df_{den}} \tag{3}$$

where

$$df_{num} = \frac{(MS_{num,1} + \dots + MS_{num,r_n})^2}{(MS_{num,1})^2/df_{num,1} + \dots + (MS_{num,r_n})^2/df_{num,r_n}} \tag{4}$$

$$df_{den} = \frac{(MS_{den,1} + \dots + MS_{den,r_d})^2}{(MS_{den,1})^2/df_{den,1} + \dots + (MS_{den,r_d})^2/df_{den,r_d}} \tag{5}$$

In df_{num} and df_{den} , $df_{num,i}$ and $df_{den,j}$ are the degrees of freedom associated with the mean square $MS_{num,i}$ and $MS_{den,j}$, respectively, where $i = 1, \dots, r_n$ and $j = 1, \dots, r_d$

Implementation of EMSaov package

The **EMSaov** package includes `EMSanova`, `PooledANOVA`, and `ApproxF` as main functions and `EMSaovApp` as a function for the Shiny application. `EMSanova` generates the analysis of variance (ANOVA) table with the expected mean squares (EMS) and the corrected F tests considered with the EMS. Several arguments are needed for this function (Table 4). We use the `formula` argument to specify the response variable and factors in the ANOVA table with data, nested factors (`nested`), and types of factors (`type`). Sometimes, we cannot find the appropriate denominator for the F statistic, and we have to use the approximate F test. The function `ApproxF` is developed to approximate the results of the F test. The `ApproxF` function takes `SS.table` and `approx.name` as arguments. `SS.table` is the result from `EMSanova`, and `approx.name` designates the source of variation in `SS.table` to calculate the approximate F values for the test. To show how to use these functions in the **EMSaov** package, we use the three sets of example data in [Hicks \(1982\)](#).

Example 1: Mixed effect model with approximate F test

`film` data in **EMSaov** corresponds to the mixed effect model (Example 10.3 in [Hicks \(1982\)](#)). There are three factors: Gate, Operator, and Day. The experiment consists of measuring the dry-film thickness of varnish in millimeters for three different gate setting (1, 2, and 3) twice with operators A, B, and C, for two days.

In `film`, "thickness" is the dependent variable, and "Gate", "Operator", and "Day" are the factors that we want to consider. We use `thickness ~ Gate + Operator + Day` as a formula. In the `EMSanova` function, we use the formula format just for specifying the factors in the model. To specify the types of factors and whether the factors are nested or not, we need to use the other arguments. If the user

argument	
formula	model formula
data	data frame for ANOVA
type	the list of factor types. It designates whether each factor is random or not. use "F" for the fixed effect, "R" for the random effect
nested	the list of the nested effects
level	the list of the model level
n.table	numbers of levels in each factor
approximate	calculate approximate F test when it is TRUE

Table 4: Arguments of EMSanova function

specifies formula = thickness ~ Gate + Operator + Day, type, nested, level, and n.table should follow the order of "Gate", "Operator" and "Day". In this example, "Gate" is treated as a fixed effect and "Operator" and "Day" are treated as random effects. Therefore, type = c("F", "R", "R").

```
> data(film)
> anova.result <- EMSanova(thickness ~ Gate + Operator + Day, data = film,
+                           type = c("F", "R", "R"))
> anova.result
```

	Df	SS	MS	Fvalue	Pvalue	Sig
Gate	2	1.573172222	0.786586111			
Operator	2	0.112072222	0.056036111	18.7656	0.0506	.
Gate:Operator	4	0.042844444	0.010711111	4.3229	0.0926	.
Day	1	0.001002778	0.001002778	0.3358	0.6208	
Gate:Day	2	0.011338889	0.005669444	2.2881	0.2175	
Operator:Day	2	0.005972222	0.002986111	9.188	0.0018	**
Gate:Operator:Day	4	0.009911111	0.002477778	7.6239	9e-04	***
Residuals	18	0.005850000	0.000325000			

```

EMS
Gate      Error+2Gate:Operator:Day+6Gate:Day+4Gate:Operator+12Gate
Operator      Error+6Operator:Day+12Operator
Gate:Operator      Error+2Gate:Operator:Day+4Gate:Operator
Day      Error+6Operator:Day+18Day
Gate:Day      Error+2Gate:Operator:Day+6Gate:Day
Operator:Day      Error+6Operator:Day
Gate:Operator:Day      Error+2Gate:Operator:Day
Residuals      Error

```

For the factor "Gate", the EMS of the denominator should be "Error + 2Gate:Operator:Day + 6Gate:Day + 4Gate:Operator", but it is not so in this table. Therefore, we cannot find the exact denominator for the F test and need to use the approximate F test. The factor "Gate" is in the first row in the result of EMSanova and approx.name should be "Gate" for the ApproxF function.

```
> ApproxF(SS.table = anova.result, approx.name = "Gate")
$Appr.F
[1] 48.17076

$df1
[1] 2.01261

$df2
[1] 5.995597

$Appr.Pvalue
[1] 0.0002010433
```

The approximate F value for the test of the factor "Gate" is 48.17076 with p-value 0.0002. Therefore, we can conclude that there are significant differences among the levels of the factor "Gate" at the significance level 0.05.

If we want to combine "Gate:Day" and "Residuals", and treat them as a combined residual for the further analysis, we can define del.ID as c("Gate:Day", "Residuals") and use the PooledANOVA

function. The first argument for PooledANOVA is the output from EMSanova.

```
> del.ID <- c("Gate:Day", "Residuals")
> PooledANOVA(anova.result, del.ID)
```

	Df	SS	MS	Fvalue	Pvalue	Sig
Gate	2	1.5732	0.7866	73.4365	7e-04	***
Operator	2	0.1121	0.0560	18.7656	0.0506	.
Gate:Operator	4	0.0428	0.0107	4.3229	0.0926	.
Day	1	0.0010	0.0010	0.3358	0.6208	.
Operator:Day	2	0.0060	0.0030	3.4745	0.0507	.
Gate:Operator:Day	4	0.0099	0.0025	2.883	0.0491	*
Residuals	20	0.0172	0.0009			

	EMS
Gate	Error+2Gate:Operator:Day+4Gate:Operator+12Gate
Operator	Error+6Operator:Day+12Operator
Gate:Operator	Error+2Gate:Operator:Day+4Gate:Operator
Day	Error+6Operator:Day+18Day
Operator:Day	Error+6Operator:Day
Gate:Operator:Day	Error+2Gate:Operator:Day
Residuals	Error

Example 2: Nested-factorial model

For the nested model, we need to specify nested, which indicates the parent factor of the nested factor. We use the baseball data (Example 11.4 in Hicks (1982)) to illustrate the use of the EMSanova function with the nested-factorial model. It has three factors where the subjects are nested within groups.

In this example, formula = velocity ~ Group + Subject + test. The factors "Group" and "test" are treated as fixed effects, and the factor "Subject" is treated as a random effect with the argument for fixed and random effects being type = c("F", "R", "F"). We use nested = c(NA, "Group", NA) to indicate the factor "Subject", nested in the factor "Group". The fixed effect "test" is measured twice with "Pre" and "Post" in each subject, which means that there are two levels in this model. The first level consists of "Group" and "Subject" and the second level consists of "test". This model level can thus be represented with level=c(1,1,2).

```
> data(baseball)
> anova.result <- EMSanova(velocity ~ Group + Subject + test,
+                           data = baseball,
+                           type = c("F", "R", "F"),
+                           nested = c(NA, "Group", NA),
+                           level = c(1, 1, 2))
> anova.result
```

	Df	SS	MS	Fvalue	Pvalue	Sig	Model.Level
Group	2	28.139200	14.0696000	1.0426	0.3729		1
Subject(Group)	18	242.905914	13.4947730	115.517	<0.0001	***	1
test	1	21.257486	21.2574857	181.9669	<0.0001	***	2
Group:test	2	12.381943	6.1909714	52.9955	<0.0001	***	2
Residuals	18	2.102771	0.1168206				2

	EMS
Group	Error+2Subject(Group)+14Group
Subject(Group)	Error+2Subject(Group)
test	Error+21test
Group:test	Error+7Group:test
Residuals	Error

Example 3: Split-split plot design

The example rubber is the split-split plot design with 4 replicates (Example 13.3 in Hicks (1982)). Three different laboratories (Lap), three different temperatures (Temp), and three types of rubber mix (Mix) were involved in each replicate (Rep). "Rep" and "Lap" consist of the whole plot, "Temp" is added in the split-plot, and "Mix" is added in the split-split plot.

This design can be treated as a three-level model with "Rep" and "Lap" in the first level, "Temp" in the second level, and "Mix" in the third level. For the EMSanova, we set formula = Y ~ Rep + Mix + Lap + Temp, type = c("R", "F", "F", "F") and level = c(1,3,1,2).

```

> data(rubber)
> anova.result <- EMSanova(cure ~ Rep + Mix + Lap + Temp,
+                           data = rubber,
+                           type = c("R", "F", "F", "F"),
+                           level = c(1, 3, 1, 2))
> anova.result

```

	Df	SS	MS	Fvalue	Pvalue	Sig	Model.Level
Rep	3	5.581019	1.8603395	2.2311	0.1105		1
Lap	2	51.496852	25.7484259	4.9395	0.0539	.	1
Rep:Lap	6	31.276481	5.2127469	6.2517	5e-04	***	1
Temp	2	2978.564630	1489.2823148	2167.7859	<0.0001	***	2
Rep:Temp	6	4.122037	0.6870062	0.8239	0.5626		2
Lap:Temp	4	5.603148	1.4007870	0.5948	0.6732		2
Rep:Lap:Temp	12	28.261296	2.3551080	2.8245	0.0146	*	2
Mix	2	149.217963	74.6089815	53.0128	2e-04	***	3
Rep:Mix	6	8.444259	1.4073765	1.6879	0.1672		3
Lap:Mix	4	4.894815	1.2237037	1.4772	0.2697		3
Rep:Lap:Mix	12	9.940741	0.8283951	0.9935	0.4828		3
Temp:Mix	4	47.853704	11.9634259	15.6467	1e-04	***	3
Rep:Temp:Mix	12	9.175185	0.7645988	0.917	0.5454		3
Lap:Temp:Mix	8	10.688519	1.3360648	1.6024	0.1765		3
Residuals	24	20.011481	0.8338117				3

```

EMS
Rep          Error+27Rep
Lap          Error+9Rep:Lap+36Lap
Rep:Lap      Error+9Rep:Lap
Temp         Error+9Rep:Temp+36Temp
Rep:Temp     Error+9Rep:Temp
Lap:Temp     Error+3Rep:Lap:Temp+12Lap:Temp
Rep:Lap:Temp Error+3Rep:Lap:Temp
Mix          Error+9Rep:Mix+36Mix
Rep:Mix      Error+9Rep:Mix
Lap:Mix      Error+3Rep:Lap:Mix+12Lap:Mix
Rep:Lap:Mix  Error+3Rep:Lap:Mix
Temp:Mix     Error+3Rep:Temp:Mix+12Temp:Mix
Rep:Temp:Mix Error+3Rep:Temp:Mix
Lap:Temp:Mix Error+4Lap:Temp:Mix
Residuals   Error

```

EMSaovApp: Web interface for ANOVA with EMS

Even though we provide three functions to produce the appropriate analysis of variance for many different types of experimental designs, this is not so easy for a novice user of R. For convenience, we provide a Shiny-based application with a graphical user interface (GUI) to obtain the ANOVA of data from various experimental designs. Figure 2 shows the main GUI for EMSaovApp with Example 2. The first part of the GUI can be used to read data in the csv format. The middle part has various input windows to select the dependent variable, the factors in the ANOVA table, the number of categories in each factor, the specification of the nested factor, and the level of the model. In this example, "Y" is selected as the dependent variable (Y variable) and the "Group", "Subject", and "Test" factors are selected. Among the selected factors, "Subject" is treated as the random effect and the others are treated as fixed effects. Therefore "Subject" is checked in the "Random Effect" part. The "Subject" factor is nested in "Group", and the "test" factor is in the second level of the model. This information should thus be specified in this part. The number of categories for each factor is automatically calculated from the original data, but the user can change them in this GUI.

The bottom part has five tabs, including "EDA-main effect", "EDA-interaction", "ANOVA table", "ANOVA table with Approx.F", and "Pooled ANOVA". The "EDA-main effect" tab shows parallel box plots for each factor (Figure 2). "Subject" and "Test" show significant differences among the levels in each factor, and the "EDA-interaction" tab shows the interaction plots to help see whether the interaction effect between the two factors is significant or not. EMSaovApp automatically generates interaction plots for all pairs of selected factors. In Figure 3, the interaction effect between "Group" and "Test" is highly significant. Even though the interaction effect between "Group" and "Subject" and the interaction effect between "Subject" and "Test" are provided, "Subject" is nested in "Group" and

Shiny Application for ANOVA with EMS

File input
 Example-2.csv

Y variable

X variable
 Group
 Subject
 Test
 Y

Random Effect
 Group
 Subject
 Test
 Y

[# of categories] Group

[nested] Group

[model level] Group

Subject

Subject

Subject

Test

Test

Test

Y

Y

Y

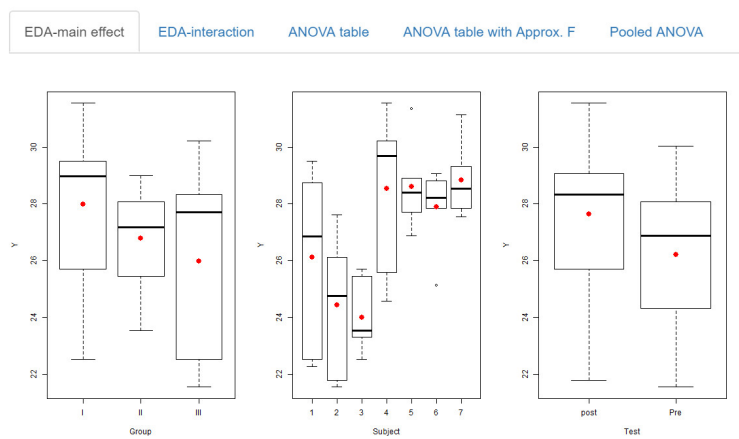


Figure 2: Main feature of EMSaovApp

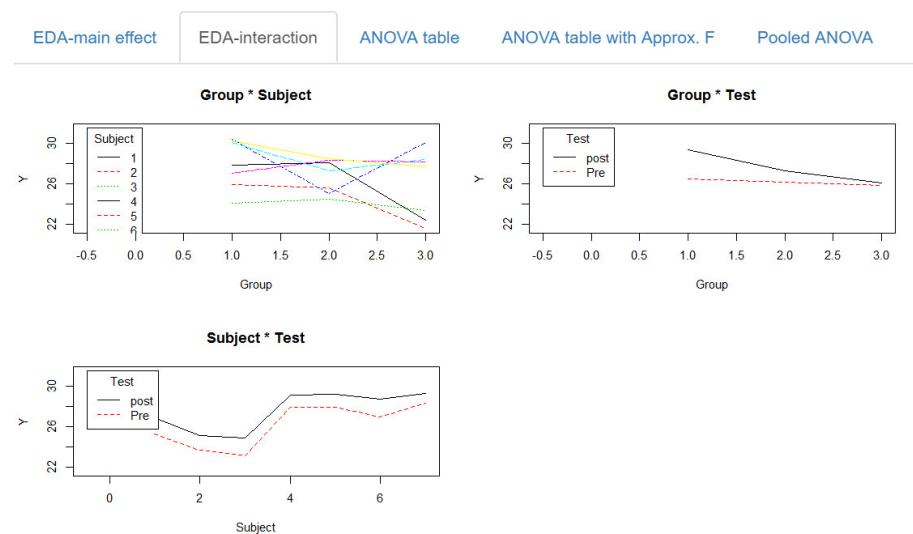


Figure 3: Exploratory data analysis of the interaction effect between two factors

Shiny Application for ANOVA with EMS

File input
 Example-1.csv

Y variable

X variable
 Gate
 Operator
 Day
 Y

Random Effect
 Gate
 Operator
 Day
 Y

[# of categories] Gate

[nested] Gate

[model level] Gate

Operator

Operator

Operator

Day

Day

Day

Y

Y

Y

EDA-main effect EDA-interaction ANOVA table ANOVA table with Approx. F Pooled ANOVA

	Df	SS	MS	Fvalue	Pvalue	Sig	EMS
Gate	2.00	1.57	0.79	48.1708	2e-04	***	Error+2Gate:Operator:Day+6Gate:Day+4Gate:Operator+12Gate
Operator	2.00	0.11	0.06	18.7656	0.0506	.	Error+6Operator:Day+12Operator
Gate:Operator	4.00	0.04	0.01	4.3229	0.0926	.	Error+2Gate:Operator:Day+4Gate:Operator
Day	1.00	0.00	0.00	0.3358	0.6208	.	Error+6Operator:Day+18Day
Gate:Day	2.00	0.01	0.01	2.2881	0.2175	.	Error+2Gate:Operator:Day+6Gate:Day
Operator:Day	2.00	0.01	0.00	9.188	0.0018	**	Error+6Operator:Day
Gate:Operator:Day	4.00	0.01	0.00	7.6239	9e-04	***	Error+2Gate:Operator:Day
Residuals	18.00	0.01	0.00				Error

Signif. codes : <0.0001 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Figure 4: Analysis of variance table with approximate F test result

EDA-main effect EDA-interaction ANOVA table ANOVA table with Approx. F Pooled ANOVA

Combine ANOVA table

Gate
 Operator
 Gate:Operator
 Day
 Gate:Day
 Operator:Day
 Gate:Operator:Day
 Residuals

	Df	SS	MS	Fvalue	Pvalue	Sig	EMS
Gate	2.00	1.57	0.79	48.1101	2e-04	***	Error+6Gate:Day+4Gate:Operator+12Gate
Operator	2.00	0.11	0.06	18.6833	0.0508	.	Error+6Operator:Day+12Operator
Gate:Operator	4.00	0.04	0.01	14.8987	<0.0001	***	Error+4Gate:Operator
Day	1.00	0.00	0.00	0.3333	0.622	.	Error+6Operator:Day+18Day
Gate:Day	2.00	0.01	0.01	7.8671	0.0026	**	Error+6Gate:Day
Operator:Day	2.00	0.01	0.00	4.1772	0.029	*	Error+6Operator:Day
Residuals	22.00	0.02	0.00				Error

Figure 5: Pooled analysis of variance table - pooling three way interaction effect "Gate:Operator:Day" to "Residuals"

these interaction effects are not of interest. Figure 4 show the result of the tab "ANOVA table with Approx. F" with data from Example 1. In this example, we can combine the three-way interaction to the residuals by checking the corresponding items in the "Pooled ANOVA" tab (Figure 5).

Discussion

We have introduced **EMSaov**, an R package for ANOVA with various types of experimental design. We should have information on the EMS to determine the denominator of the F statistics in ANOVA. **EMSaov** is one of few packages for ANOVA with EMS that can handle fixed or random effects, nested factors, or model with multi-level design. Future updates to **EMSaov** include the permutation tests for each source of variation to cover the situation where data does not follow the normal assumption.

Bibliography

- C. A. Bennett, N. L. Franklin, and others. *Statistical Analysis in Chemistry and the Chemical Industry*. John Wiley & Sons, 1954. [p253]
- J. Cornfield and J. W. Tukey. Average values of mean squares in factorials. *The Annals of Mathematical Statistics*, pages 907–949, 1956. URL <https://doi.org/10.1214/aoms/1177728067>. [p252]
- C. R. Hicks. *Fundamental Concepts in the Design of Experiments*. Fort Worth : Saunders College Pub., 1982. [p255, 257]
- D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2008. [p254]
- J. Pinheiro, D. Bates, S. DebRoy, and D. Sarkar. R core team (2016) nlme: Linear and nonlinear mixed effects models. r package version 3.1-117, 2016. URL <https://cran.r-project.org/web/packages/nlme/index.html>. Maintained by R-core. [p252]
- F. E. Satterthwaite. An approximate distribution of estimates of variance components. *Biometrics bulletin*, 2(6):110–114, 1946. URL <https://doi.org/10.2307/3002019>. [p255]
- H. Singmann, B. Bolker, J. Westfall, F. Aust, S. Højsgaard, J. Fox, M. A. Lawrence, U., and Mertens. Afex: Analysis of factorial experiments, 2016. URL <https://cran.r-project.org/web/packages/afex/index.html>. Maintained by Singmann, H. [p252]

Hye-Min Choe
Department of Statistics
Ewha Womans University
Seoul, Korea

Mijeong Kim
Department of Statistics
Ewha Womans University
Seoul, Korea

Eun-Kyung Lee
Department of Statistics
Ewha Womans University
Seoul, Korea
lee.eunk@ewha.ac.kr

GsymPoint: An R Package to Estimate the Generalized Symmetry Point, an Optimal Cut-off Point for Binary Classification in Continuous Diagnostic Tests

by *Mónica López-Ratón, Elisa M. Molanes-López, Emilio Letón, and Carmen Cadarso-Suárez*

Abstract In clinical practice, it is very useful to select an optimal cutpoint in the scale of a continuous biomarker or diagnostic test for classifying individuals as healthy or diseased. Several methods for choosing optimal cutpoints have been presented in the literature, depending on the ultimate goal. One of these methods, the generalized symmetry point, recently introduced, generalizes the symmetry point by incorporating the misclassification costs. Two statistical approaches have been proposed in the literature for estimating this optimal cutpoint and its associated sensitivity and specificity measures, a parametric method based on the generalized pivotal quantity and a nonparametric method based on empirical likelihood. In this paper, we introduce **GsymPoint**, an R package that implements these methods in a user-friendly environment, allowing the end-user to calculate the generalized symmetry point depending on the levels of certain categorical covariates. The practical use of this package is illustrated using three real biomedical datasets.

Introduction

The classification of cases and controls is a common task in several fields. For example, it is conducted in the atmospheric sciences (rainy/non rainy day), finance (good/not good payer), sociology (good/not good citizen), industry (product of good/poor quality), computing science (spam/non-spam) or health sciences (healthy/diseased), among others. However, in this paper we will focus on the latter example, where we will be interested on the classification of individuals as healthy or diseased using a continuous biomarker or diagnostic test that will be based on a cutoff point or discrimination value c . If we suppose, without loss of generality, that high test values are associated with the disease under study, individuals with a diagnostic test value equal to or higher than c are classified as diseased (positive test) and individuals with a value lower than c are classified as healthy (negative test). The test can classify a diseased patient incorrectly, that is, a false negative decision, or detect a patient as diseased when his (or her) true status is healthy (a false positive misclassification). So, for each cutoff point c , it is necessary to quantify these errors to check the validity of the diagnostic test in clinical practice. In a similar way, the test can correctly classify a diseased patient (a true positive decision) or a healthy patient (a true negative decision). For each cutpoint c , the sensitivity and specificity measures of the accuracy of the diagnostic test can be defined from these correct decisions. The sensitivity (Se) is the probability of a true positive decision, that is, the probability of correctly classifying a diseased patient (true positive rate) and the specificity (Sp) is the probability of a true negative decision (true negative rate), that is, the probability of correctly classifying a healthy individual. Similarly, the probabilities of the incorrect classifications are defined as $1 - Se$ (false negative rate) and $1 - Sp$ (false positive rate).

Considering Se versus $1 - Sp$ for all possible values of c , a curve is obtained. This curve is called the Receiver Operating Characteristic (ROC) curve (Metz, 1978; Swets and Swets, 1979; Swets and Pickett, 1982) and it is a graphical global measure of the diagnostic accuracy of a continuous diagnostic test, independent of the cutpoint and the disease prevalence. In addition, it serves as a guide when selecting optimal cutoffs and as a measure of the overlapping of the test values between healthy and diseased populations. Numerical indexes defined from ROC curves are often used to summarize the accuracy of a diagnostic test. For instance, the area under the curve (AUC) (Bamber, 1975; Swets, 1979) is the most commonly used and it takes values between 0.5 (the AUC value of an uninformative test, the same as a random prediction) and 1 (the AUC value of a perfect test that classifies correctly all individuals, either healthy or diseased). The AUC is equal to the Mann-Whitney U statistic for comparing two distributions. Both of them can be interpreted as the probability that the diagnostic value of a randomly chosen diseased individual will be higher than the diagnostic value of a randomly chosen healthy one (Hanley and McNeil, 1982).

A key question in clinical practice is to find a cutpoint that “best” discriminates between patients with and without the disease. However, one cannot talk in absolute terms of a “best choice”. This is the reason why several criteria for selecting optimal cutpoints have been proposed in the literature depending on the ultimate goal of such selection (see Youden, 1950; Feinstein, 1975; Metz, 1978; Schäfer,

1989; Vermont et al., 1991; Greiner, 1995; Pepe, 2003, for example). One of the best-known methods is based on selecting the cutpoint that provides the same value for the sensitivity and specificity. This point is known as the equivalence or symmetry point (Greiner, 1995; Defreitas et al., 2004; Adlhoch et al., 2011). Graphically, it corresponds with the operating point on the ROC curve that intersects the perpendicular to the positive diagonal line, that is, $y = 1 - x$, where x is the false positive rate. The symmetry point can also be seen as the point that maximizes simultaneously both types of correct classifications (Riddle and Stratford, 1999; Gallop et al., 2003), that is, it corresponds to the probability of correctly classifying any subject, whether it is healthy or diseased (Jiménez-Valverde, 2012, 2014). Additionally, the incorporation of costs for the misclassification rates in the estimation of optimal cutpoints is crucial for evaluating not only the test accuracy but also its clinical efficacy, although this aspect is not taken into account most of the times. So, an interesting generalization of the equivalence or symmetry point, c_S , that takes into account the costs associated to the false positive and false negative misclassifications, C_{FP} and C_{FN} , respectively, is the generalized equivalence point or generalized symmetry point, c_{GS} , that satisfies the following equation:

$$\rho(1 - Sp(c_{GS})) = (1 - Se(c_{GS})), \quad (1)$$

where $\rho = \frac{C_{FP}}{C_{FN}}$ is the relative loss (cost) of a false-positive decision as compared with a false-negative decision (see López-Ratón et al., 2016, for more details). Similarly to the symmetry point, this cost-based generalization is obtained by intersecting the ROC curve and the line $y = 1 - \rho x$, where x is the false positive rate. Obviously, when $\rho = 1$ in Equation 1, the generalized symmetry point yields the traditional symmetry point. The reader can see some medical examples, that have taken into account the misclassification costs in their ROC analysis, in the review conducted by Cantor et al. (1999) where the Cost/Benefit (C/B) ratio is discussed ($C/B = \frac{1}{\rho}$). Additionally, Subtil and Rabilloud (2015) include some common values for the C/B ratio ($C/B = 2, 5, 10, 100$). High values of C/B ratio mean that it is considered more harmful not to treat a diseased individual than to treat a healthy one.

Two statistical approaches have been recently introduced in the literature (López-Ratón et al., 2016) to obtain point estimates and confidence intervals for the generalized symmetry point and its associated sensitivity and specificity measures, a parametric method based on the Generalized Pivotal Quantity (GPQ) under the assumption of normality (Weerahandi, 1993, 1995; Lai et al., 2012), and a nonparametric method based on the Empirical Likelihood (EL) methodology without any parametric assumptions (Thomas and Grunkemeier, 1975; Molanes-López and Letón, 2011).

The availability of software for estimating optimal cutpoints in a user-friendly environment is very important and necessary for facilitating, mainly to the biomedical staff, the selection of optimal cutpoints in clinical practice. There are several packages in R to carry out this task, such as **PresenceAbsence** (Freeman and Moisen, 2008), **DiagnosisMed** (Brasil, 2010), **pROC** (Robin et al., 2011) and **OptimalCutpoints** (López-Ratón and Rodríguez-Álvarez, 2014; López-Ratón et al., 2014). However, these packages only consider the classical non-parametric empirical method for estimating optimal cutpoints and accuracy measures, that is, none of them take into account recent methodology introduced in ROC analysis such as the GPQ and EL approaches above-mentioned (Molanes-López and Letón, 2011; Lai et al., 2012).

In this paper we present **GsymPoint**, a package written in R for estimating the generalized symmetry point (López-Ratón et al., 2017), which is freely available from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=GsymPoint>. Specifically, this package enables end-users to obtain point estimates and $100(1 - \alpha)\%$ confidence intervals (with α the signification level) for the cost-based generalization of the symmetry point, c_{GS} , and its associated sensitivity and specificity measures, $Se(c_{GS})$ and $Sp(c_{GS})$, by means of the two statistical approaches pointed out above, the GPQ and EL approaches, that turn out to be more efficient than the empirical one. Therefore, we take into account in the estimation process when normality can be assumed, and we consider one of the most widely used criteria for selecting optimal cutpoints in clinical practice, the criterion that generalizes the method where the sensitivity and specificity indexes are the same, taking into account the misclassification costs, a very important issue when selecting the optimal cutpoint in a specific clinical setting. In addition, since the test accuracy can be influenced by specific characteristics such as the patient's age or gender, or the disease severity (Pepe, 2003), the **GsymPoint** package allows the computation of the generalized symmetry point for each level of a given categorical covariate. On one hand, the numerical output of **GsymPoint** includes the generalized symmetry point and its associated sensitivity and specificity indexes with their corresponding $100(1 - \alpha)\%$ confidence intervals. On the other hand, based on the graphical interpretation of the generalized symmetry point, the graphical output shows the empirical ROC curve and the line $y = 1 - \rho x$.

The rest of paper is organized as follows. In Section D.2, we briefly review the two methods included in our **GsymPoint** package for obtaining point estimates and confidence intervals for the generalized symmetry point and its sensitivity and specificity measures. In Section D.3, we describe the general use of this package, describing the most important functions. In Section D.4, we illustrate

the practical application of the package using three real biomedical datasets on melanoma, prostate cancer, and coronary artery disease. Finally, in Section D.5 we conclude with a discussion and some interesting future extensions.

Generalized symmetry point estimating methods

In this section we briefly explain the two methods included in the **GsymPoint** package for estimating and constructing confidence intervals for the generalized symmetry point c_{GS} and its associated accuracy measures $Sp(c_{GS})$ and $Se(c_{GS})$. To make inference on these parameters of interest, we consider two independent samples of i.i.d. (independently and identically distributed) observations, $\{Y_{0k_0}\}_{k_0=1}^{n_0}$ and $\{Y_{1k_1}\}_{k_1=1}^{n_1}$, taken from the healthy and diseased populations, Y_0 and Y_1 , respectively, with sample sizes n_0 and n_1 .

Generalized pivotal quantity method

Generalized confidence intervals refer to a parametric methodology based on the normality assumption, first introduced by Weerahandi (1993, 1995) and recently applied in the context of diagnostic studies to the Youden index by Lai et al. (2012) and Zhou and Qin (2013), and to the generalized symmetry point by López-Ratón et al. (2016).

Assuming that the diagnostic test in healthy and diseased populations Y_i follows a normal distribution with mean μ_i and standard deviation σ_i , for $i = 0, 1$, it follows that the ROC curve has an explicit expression:

$$ROC(x) = \Phi(a + b\Phi^{-1}(x)), \quad (2)$$

where $a = \frac{\mu_1 - \mu_0}{\sigma_1}$, $b = \frac{\sigma_0}{\sigma_1}$, $x = 1 - Sp(c_{GS})$ and Φ denotes the standard normal cumulative distribution function (cdf).

Therefore, under the normality assumption, using if necessary a monotone transformation of Box-Cox type to achieve normality, it follows from Equations 1–2 that the generalized symmetry point c_{GS} can be estimated from the following equation:

$$\Phi(a + b\Phi^{-1}(x)) = 1 - \rho x \Leftrightarrow \Phi\left(\frac{\Phi^{-1}(1 - \rho x) - a}{b}\right) - x = 0, \quad (3)$$

replacing a and b by their sample versions $\tilde{a} = \frac{m_1 - m_0}{s_1}$ and $\tilde{b} = \frac{s_0}{s_1}$, respectively, where m_i and s_i denote the sample mean and sample standard deviation of each population, for $i = 0, 1$. Once the root of Equation 3 is obtained, \tilde{x} , then the parametric point estimates of c_{GS} , $Sp(c_{GS})$ and $Se(c_{GS})$ are given by $\tilde{c}_{GS} = m_0 + s_0\Phi^{-1}(1 - \tilde{x})$, $\tilde{Sp}(c_{GS}) = 1 - \tilde{x}$ and $\tilde{Se}(c_{GS}) = 1 - \rho\tilde{x}$, respectively.

For computing the GPQ-based confidence intervals of c_{GS} , $Sp(c_{GS})$ and $Se(c_{GS})$, López-Ratón et al. (2016) follow the same reasoning as in Lai et al. (2012), based on considering their corresponding generalized pivotal quantities, that is, substituting a and b into Equation 3 with their generalized pivotal values. For instance, if $R_{c_{GS}}$ denotes the generalized pivotal quantity for estimating c_{GS} , and $R_{c_{GS},\alpha}$ denotes the α th quantile of the distribution of $R_{c_{GS}}$, then the corresponding $100(1 - \alpha)\%$ confidence interval for c_{GS} based on its generalized pivotal quantity is given by the percentile method, that is, by $(R_{c_{GS},\alpha/2}, R_{c_{GS},1-\alpha/2})$.

Empirical likelihood method

The empirical likelihood method was firstly introduced by Thomas and Grunkemeier (1975) that proposed the construction of EL-based confidence intervals for the Kaplan-Meier estimator. Nowadays, this methodology is an active area of research in several fields due to the good properties presented by EL-based confidence intervals and regions (see, for example, Molanes-López et al., 2009, among others). Moreover, this methodology has the advantages of easy implementation and not requiring any particular parametric assumption. In the recent literature, Molanes-López and Letón (2011) proposed a bootstrap-based EL approach to make inference on the Youden index and its associated optimal cutpoint, and López-Ratón et al. (2016) applied these same bootstrap-based EL ideas for estimating and constructing confidence intervals for the generalized symmetry point and its corresponding specificity and sensitivity measures. The key point in both works is that the optimal cutpoint of interest can be seen as specific quantiles of the two populations involved.

As López-Ratón et al. (2016) mention, c_{GS} can be seen as the $Sp(c_{GS})$ -th quantile of the healthy population and the $\rho(1 - Sp(c_{GS}))$ -th quantile of the diseased population. Considering that the value

of $Sp(c_{GS})$ is known in advance, they derive the following log-likelihood function to make inference on the generalized symmetry point:

$$\begin{aligned} \ell(c) &= -2 \log(L(c)) \\ &= 2n_0 \hat{F}_{0,g_0}(c) \log \left(\frac{\hat{F}_{0,g_0}(c)}{Sp(c_{GS})} \right) + 2n_0 (1 - \hat{F}_{0,g_0}(c)) \log \left(\frac{1 - \hat{F}_{0,g_0}(c)}{1 - Sp(c_{GS})} \right) \\ &\quad + 2n_1 \hat{F}_{1,g_1}(c) \log \left(\frac{\hat{F}_{1,g_1}(c)}{\rho(1 - Sp(c_{GS}))} \right) + 2n_1 (1 - \hat{F}_{1,g_1}(c)) \log \left(\frac{1 - \hat{F}_{1,g_1}(c)}{1 - \rho(1 - Sp(c_{GS}))} \right), \end{aligned}$$

where, for $i = 0, 1$, \hat{F}_{i,g_i} is a kernel-type estimate of the cdf F_i given by

$$\hat{F}_{i,g_i}(y) = \frac{1}{n_i} \sum_{k_i=1}^{n_i} \mathbb{K} \left(\frac{y - Y_{ik_i}}{g_i} \right),$$

with $\mathbb{K}(x) = \int_{-\infty}^y K(z) dz$, K a kernel function and g_i the corresponding smoothing parameter, for $i = 0, 1$. Therefore, assuming that $Sp(c_{GS})$ is known in advance, a nonparametric point estimate of c_{GS} could be found by minimizing $\ell(c)$ over c , and a confidence interval for it could be obtained based on the usual χ^2 limiting distribution of $\ell(c)$. However, taking into account the fact that $Sp(c_{GS})$ is unknown, López-Ratón et al. (2016) propose to estimate this parameter first by means of a kernel-based method and then obtain a non-parametric point estimate of c_{GS} by minimizing the previous log-likelihood function, $\ell(c)$, but where now the unknown parameter $Sp(c_{GS})$ is replaced by its estimate. Besides, they introduce a bootstrap-based EL approach that reproduces this procedure for each pair of resamples taken independently from the two original samples and construct percentile based confidence intervals from all the EL-based estimates previously calculated for those bootstrap resamples.

The GsymPoint package

In this section we present **GsymPoint**, a package written in R for estimating the generalized symmetry point (López-Ratón et al., 2017), which is freely available from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=GsymPoint>. This package enables end-users to obtain point estimates and $100(1 - \alpha)\%$ confidence intervals using two recent methodologies introduced in ROC analysis (Molanes-López and Letón, 2011; Lai et al., 2012) for the generalized symmetry point and its corresponding sensitivity and specificity indexes. Specifically, the two estimating methods presented in the previous section, the generalized pivotal quantity method (Weerahandi, 1993, 1995; Lai et al., 2012; Zhou and Qin, 2013) and the empirical likelihood method (Thomas and Grunkemeier, 1975; Molanes-López and Letón, 2011) have been incorporated in a clear and user-friendly way for the end-user. The **GsymPoint** package only requires a data-entry file where each column indicates a variable and each row indicates an individual or patient. This dataset must, at least, contain the variable with the diagnostic test values, the variable that indicates the true disease status (diseased or healthy) and the variable with the levels of a categorical covariate of interest in case the optimal cutpoint has to be computed for each level of that covariate. The numerical output of **GsymPoint** package includes the generalized symmetry point and its corresponding sensitivity and specificity indexes with their associated $100(1 - \alpha)\%$ confidence intervals. In basis on the graphical interpretation of the generalized symmetry point, the graphical output shows the intersection point between the empirical ROC curve and the line $y = 1 - \rho x$, that is, an empirical estimate of the operating point corresponding to the generalized symmetry point. Table 1 provides a summary of the most important functions included in the package.

The `gsym.point()` function

The main function of the package is the `gsym.point()` function, which uses the selected method(s) (GPQ, EL or auto, where this last option automatically selects the most appropriate method based on the normality assumption) to obtain (parametric and/or nonparametric) confidence intervals and point estimates for c_{GS} , $Sp(c_{GS})$ and $Se(c_{GS})$, and creates an object of class "gsym.point". The code to use the `gsym.point()` function is as follows:

```
gsym.point(methods, data, marker, status, tag.healthy,
           categorical.cov = NULL, control = control.gsym.point(),
           CFN = 1, CFP = 1, confidence.level = 0.95,
           trace = FALSE, seed = FALSE, value.seed = 3, verbose = TRUE)
```

Function	Description
<code>gsym.point()</code>	Main function that computes the generalized symmetry point and its sensitivity and specificity measures with their corresponding confidence intervals.
<code>control.gsym.point()</code>	Used to set several parameters that control the estimation process of the optimal cutpoint.
<code>print()</code>	Print method for "gsym.point" class objects.
<code>summary()</code>	Summary method for "gsym.point" class objects.
<code>plot()</code>	Plot method for "gsym.point" class objects that shows in the same graph the empirical ROC curve and the line $y = 1 - \rho x$. The intersection point between them is an empirical estimate of the operating point associated to the optimal cutpoint given by the generalized symmetry point.

Table 1: Most important functions included in the **GsymPoint** package.

The `methods` argument is a character vector to select the estimation method(s) to be used. The possible options are: "GPQ", "EL", `c("GPQ", "EL")`, `c("EL", "GPQ")` or "auto".

The `data` argument is the data frame containing all the needed variables: the diagnostic marker, the true disease status and, when necessary, the categorical covariate; the `marker` and `status` arguments are character strings with the names of the diagnostic test and the variable that distinguishes healthy from diseased individuals, respectively. The value codifying healthy individuals in this last variable status is indicated in the `tag.healthy` argument.

The `categorical.cov` argument is a character string with the name of the categorical covariate according to which c_{GS} is automatically computed for each of its levels. By default it is NULL, that is, no categorical covariate is considered in the analysis.

The `control` argument indicates the output of the `control.gsym.point()` function, which controls the whole calculation process. This function will be explained in detail in the following subsection.

The `CFN` and `CFP` arguments are the misclassification costs of false negative and false positive classifications, respectively. The default value is 1 for both, that is, no misclassification costs are taking into account by default.

The `confidence.level` argument is the numerical value of the confidence level $1 - \alpha$ considered for the construction of confidence intervals. By default it is equal to 0.95.

The `trace` argument is a logical value that shows information on the calculation progress if TRUE. By default it is FALSE.

The `seed` argument is a logical value, such that if TRUE, a seed is fixed for generating the trials when computing the confidence intervals, allowing the reproducibility of the results at any other time. The default value is FALSE.

The `value.seed` argument is the numerical value for the fixed seed if `seed` is TRUE. By default it is equal to 3.

The `verbose` argument is a logical value that allows to show extra information on progress of running. By default it is TRUE.

Some of these arguments, `methods`, `data`, `marker`, `status` and `tag.healthy`, are essential and must be specified in the call to the `gsym.point()` function because, otherwise, an error is produced. The other arguments, `categorical.cov`, `control`, `confidence.level`, `trace`, `seed` and `value.seed`, are optional and, if they are not specified explicitly in the call, the values by default are taken.

The `control.gsym.point()` function

It should be noted that there are some extra arguments, specific to each estimation method. We considered to include all of them in the `control` argument, which is a list of control values specified by calling to the `control.gsym.point()` function, designed to replace the default values yielded by the `control.gsym.point()` function. The arguments of the `control.gsym.point()` function are presented

in Table 2.

Argument	Description
B	The number of simulations in the empirical likelihood ("EL") method. The default value is 499 based on Carpenter and Bithell (2000) .
c_sampling	The constant needed for resampling in the empirical likelihood ("EL") method. The default value is 0.25.
c_F	The constant needed for estimating the distribution in the empirical likelihood ("EL") method. The default value is 0.25.
c_ELq	The constant needed for estimating the empirical likelihood function in the empirical likelihood ("EL") method. The default value is 0.25.
c_R	The constant needed for estimating the ROC curve in the empirical likelihood ("EL") method. The default value is 0.25.
I	The number of replicates in the generalized pivotal quantity ("GPQ") method. The default value is 2500.

Table 2: Arguments of the `control.gsym.point()` function.

The `summary.gsym.point()` function

Numerical results are printed on the screen, and the output yielded by the `summary.gsym.point()` function or the `summary()` method always includes:

- The matched call to the main function `gsym.point()`.
- The estimated value of the area under the ROC curve (AUC) based on the Mann-Whitney U statistic.
- Information related to the generalized symmetry point:
 - The method(s) (EL and/or GPQ) used for estimating c_{GS} , $Sp(c_{GS})$ and $Se(c_{GS})$.
 - The point estimates and confidence intervals for c_{GS} , $Sp(c_{GS})$ and $Se(c_{GS})$.

Apart from the matched call, that it is presented only once at the beginning, all the other information will be shown for each level of the categorical covariate if this is specified in the call, that is, if the `categorical.cov` argument in the `gsym.point()` function is not `NULL`.

The call to this function is as follows:

```
summary(object, ...)
```

where the `object` argument is a "gsym.point" class object as produced by the `gsym.point()` function and the ellipsis `...` indicate further arguments passed to or from other methods. None are used in this method.

The `plot.gsym.point()` function

The graphical output of the **GsymPoint** package is yielded by the `plot.gsym.point()` function or by the `plot()` method. This function plots the empirical ROC curve and the line $y = 1 - \rho x$. The call to this function is as follows:

```
plot(x, legend = TRUE, ...)
```

where the `x` argument is a "gsym.point" class object as produced by the `gsym.point()` function, the argument `legend` is a logical value for including the AUC value in the legend when it is `TRUE` (value by default) and the ellipsis `...` indicate further arguments passed to the `plot.default()` method.

Practical application of GsymPoint package

This section illustrates the use of the R-based **GsymPoint** package by means of three real biomedical datasets on melanoma, prostate cancer, and coronary artery disease.

Melanoma dataset

Dermatologists use a clinical scoring scheme without dermoscope (CSS) or a dermoscopic scoring scheme (DSS) to clinically evaluate lesions on the skin on the basis of several visible features such as asymmetry, border irregularity, colouration and size. We have considered a dataset on 72 patients with suspicious lesions of being a melanoma (Venkatraman and Begg, 1996). Taking into account that 21 melanomas were detected through biopsies, our objective here is to evaluate the capacity of the CSS as a potential non-invasive diagnostic marker for discriminating between melanomas and non-melanomas using the generalized symmetry point as the binary classification threshold. In the following, we illustrate the use of the **GsymPoint** package for estimating that optimal threshold value. The first step consists on attaching the **GsymPoint** package and the melanoma dataset as follows:

```
> library("GsymPoint")
> data("melanoma")
```

With the following instruction, we can get some basic summary statistics of the variables included in the melanoma dataset:

```
> summary(melanoma)
      X                group
Min.  :-5.88100  Min.   :0.0000
1st Qu.:-3.22100  1st Qu.:0.0000
Median :-1.69550  Median :0.0000
Mean   :-1.55642  Mean   :0.2917
3rd Qu.: 0.00675  3rd Qu.:1.0000
Max.   : 3.03200  Max.   :1.0000
```

To estimate the generalized symmetry point of the CSS marker, we need to call the `gsym.point()` function. For instance, as specified below.

```
> melanoma.cutpoint1 <- gsym.point(methods = "GPQ", data = melanoma,
+   marker = "X", status = "group", tag.healthy = 0,
+   categorical.cov = NULL, CFN = 2, CFP = 1,
+   control = control.gsym.point(), confidence.level = 0.95,
+   trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)
```

In this call, we have considered that a false negative decision is 2 times more serious than a false positive decision, and so we have set $CFN = 2$ and $CFP = 1$ for the misclassification costs. Besides, we have considered the GPQ method of estimation (`methods = "GPQ"`) because, according to the Shapiro-Wilk normality test, the CSS values can be assumed normally distributed in both melanoma and non-melanoma groups, and under this assumption the GPQ method is more adequate than the EL method in this case.

The `melanoma.cutpoint1` object produced by this call is a list that consists of the following components: "GPQ", "methods", "call", and "data", as can be checked below with the `names` command:

```
> names(melanoma.cutpoint1)
[1] "GPQ"      "methods"  "call"     "data"
```

The last three components, "methods", "call" and "data" are, respectively, a character vector with the value of the argument `methods` used in the call, the matched call, and the data frame used in the analysis. The first component, "GPQ", contains the results provided by the GPQ method regarding the estimation of the generalized symmetry point. Below, we detail these results:

```
> names(melanoma.cutpoint1$GPQ)
[1] "Global"

> names(melanoma.cutpoint1$GPQ$Global)
[1] "optimal.result" "AUC"          "rho"
[4] "pvalue.healthy" "pvalue.diseased"
```

```

> melanoma.cutpoint1$GPQ

$Global
$Global$optimal.result
$Global$optimal.result$cutoff
  Value      ll      ul
1 -1.213237 -1.792908 -0.6283236

$Global$optimal.result$Specificity
  Value      ll      ul
1 0.75465 0.6249716 0.8485824

$Global$optimal.result$Sensitivity
  Value      ll      ul
1 0.877325 0.8124858 0.9242912

$Global$AUC
[1] 0.9056956

$Global$rho
[1] 0.5

$Global$pvalue.healthy
[1] 0.4719117

$Global$pvalue.diseased
[1] 0.9084176

```

The "optimal.result" component is a list with the point estimates and $100(1 - \alpha)\%$ confidence intervals of the generalized symmetry point and its associated sensitivity and specificity accuracy measures, "AUC" is the numerical value of the area under the ROC curve, "rho" is the numerical value of the costs ratio $\rho = \frac{C_{FP}}{C_{FN}}$, "pvalue.healthy" is the p -value of the Shapiro-Wilk normality test used to check the normality assumption of the marker in the healthy population, and "pvalue.diseased" is the p -value of the Shapiro-Wilk normality test used to check the normality assumption in the diseased population.

The end-user can directly access each of these components. For example, the following instruction only yields the value of the AUC:

```

> melanoma.cutpoint1$GPQ$Global$AUC
[1] 0.9056956

```

A numerical summary of the results can be obtained by means of the `print.gsym.point()` or `summary.gsym.point()` functions, which can be abbreviated by the `print()` and `summary()` methods, respectively, as can be seen below:

```

> summary(melanoma.cutpoint1)

*****
OPTIMAL CUTOFF: GENERALIZED SYMMETRY POINT
*****

Call:
gsym.point(methods = "GPQ", data = melanoma, marker = "X", status = "group",
  tag.healthy = 0, categorical.cov = NULL, CFN = 2, CFP = 1,
  control = control.gsym.point(), confidence.level = 0.95,
  trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)

```

According to the Shapiro-Wilk normality test, the marker can be considered normally distributed in both groups.

Shapiro-Wilk test p-values

```

                Group 0 Group 1
Original marker 0.4719 0.9084

```

Area under the ROC curve (AUC): 0.906

METHOD: GPQ

	Estimate	95% CI lower limit	95% CI upper limit
cutoff	-1.213237	-1.7929079	-0.6283236
Specificity	0.754650	0.6249716	0.8485824
Sensitivity	0.877325	0.8124858	0.9242912

As `seed = TRUE` in the previous call, the user can replicate the output by simply running again the same call.

In this case, as can be seen above, the output provided by the `summary.gsym.point()` function shows:

1. An informative message indicating that the marker can be considered normally distributed in both groups, according to the Shapiro-Wilk normality test.
2. The Shapiro-Wilk test p -values indicating normality in both groups.
3. The AUC value and information related to the generalized symmetry point, that is, the point estimates and the GPQ based $100(1 - \alpha)\%$ confidence intervals for the generalized symmetry point and its corresponding sensitivity and specificity indexes. By default, confidence intervals are computed with a default confidence level, $1 - \alpha$, equal to 95%, although a different value can be set in the `confidence.level` argument of the main `gsym.point()` function.

So far, we have considered the GPQ method that, under normality assumptions, turns out to be more appropriate than the EL method. As you can see below, if we now replace the GPQ method by the EL method in the call, the program shows in the first place an informative message on the normality assumption and the better appropriateness of the GPQ method.

```
> melanoma.cutpoint2 <- gsym.point(methods = "EL", data = melanoma,
+   marker = "X", status = "group", tag.healthy = 0,
+   categorical.cov = NULL, CFN = 2, CFP = 1,
+   control = control.gsym.point(), confidence.level = 0.95,
+   trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)
```

According to the Shapiro-Wilk normality test, the marker can be considered normally distributed in both groups. Therefore the GPQ method would be more suitable for this dataset.

Shapiro-Wilk test p -values

	Group 0	Group 1
Original marker	0.4719	0.9084

By means of the `summary()` function, we show below the results obtained with the EL method. In this case, after reproducing the call used to create the `melanoma.cutpoint2` object, the same informative message is shown regarding the better appropriateness of the GPQ method to this dataset.

```
> summary(melanoma.cutpoint2)
```

```
*****
OPTIMAL CUTOFF: GENERALIZED SYMMETRY POINT
*****
```

Call:

```
gsym.point(methods = "EL", data = melanoma, marker = "X", status = "group",
  tag.healthy = 0, categorical.cov = NULL, CFN = 2, CFP = 1,
  control = control.gsym.point(), confidence.level = 0.95,
  trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)
```

According to the Shapiro-Wilk normality test, the marker can be considered normally distributed in both groups. Therefore the GPQ method would be more suitable for this dataset.

Shapiro-Wilk test p -values

```

                Group 0 Group 1
Original marker 0.4719 0.9084

```

Area under the ROC curve (AUC): 0.906

METHOD: EL

```

                Estimate 95% CI lower limit 95% CI upper limit
cutoff         -1.2382325      -1.8403497      -0.4565671
Specificity     0.7901833         0.6326184         0.8973174
Sensitivity     0.8950916         0.8163092         0.9486587

```

Since both estimating methods are adequate for this dataset, we could specify them simultaneously in the call to the `gsym.point()` function as follows.

```

> melanoma.cutpoint3 <- gsym.point(methods = c("EL", "GPQ"),
+   data = melanoma, marker = "X", status = "group",
+   tag.healthy = 0, categorical.cov = NULL, CFN = 2, CFP = 1,
+   control = control.gsym.point(), confidence.level = 0.95,
+   trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)

```

According to the Shapiro-Wilk normality test, the marker can be considered normally distributed in both groups.

Therefore, although the results of both methods will be shown, the GPQ method would be more suitable for this dataset.

Shapiro-Wilk test p-values

```

                Group 0 Group 1
Original marker 0.4719 0.9084

```

With the option `methods = "auto"` the program selects in this case the GPQ method based on the normality assumption satisfied for this dataset.

The graphical output of any of the three objects previously created, can be obtained by means of the `plot.gsym.point()` function, which can be abbreviated by the `plot()` method.

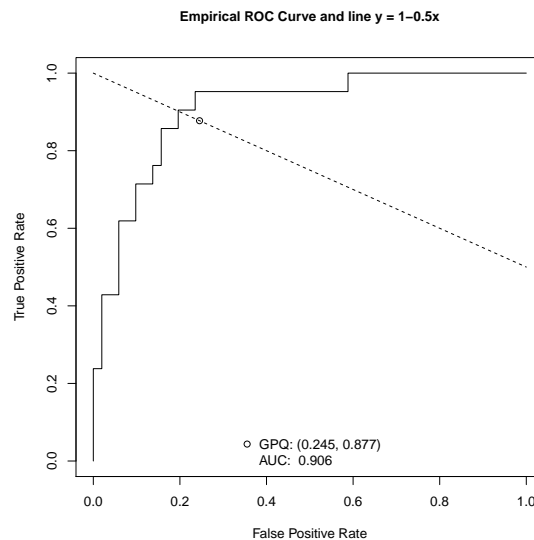


Figure 1: Graphical output for the melanoma dataset.

For instance, the call below

```

> plot(melanoma.cutpoint1)

```

shows the plot of the empirical Receiver Operating Characteristic (ROC) curve, the line $y = 1 - \rho x$, and the intersection point between them, that is, an empirical estimate of the operating point associated to the generalized symmetry point of the CSS marker for discriminating between patients with and without melanoma. Although this is the default output (see Figure 1), the end-user can add specific graphic parameters, such as color, legend, etc.

Prostate cancer dataset

We have considered here the dataset on prostate cancer analyzed by Le (2006). In order to design an appropriate treatment strategy for a patient with prostate cancer, it is important to know first whether cancer has spread or not to the neighboring lymph nodes. Although a laparoscopic surgery could confirm the true status of the patient, it is interesting to find a non-invasive diagnostic method to predict whether nodal involvement is present or not. In this dataset, 20 patients (of the total of 55 patients) had nodal involvement, and the level of acid phosphatase in blood serum (APBS) ($\times 100$) was considered as the diagnostic marker for predicting nodal involvement. We illustrate below how to apply the **GsymPoint** package for estimating the generalized symmetry point of the APBS marker that will be used for discriminating between individuals with and without nodal involvement.

As shown below, after loading the **Gsympoint** package and the prostate cancer dataset, we use the `gsym.point()` function to estimate the generalized symmetry point of the APBS marker and its associated specificity and sensitivity indexes.

```
> library("GsymPoint")
> data("prostate")
> prostate.cutpoint1 <- gsym.point(methods = "GPQ", data = prostate,
+   marker = "marker", status = "status", tag.healthy = 0,
+   categorical.cov = NULL, CFN = 10, CFP = 1,
+   control = control.gsym.point(I = 1500), confidence.level = 0.95,
+   trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)
```

Since cancer is a very serious disease which can cause death, a *FN* result in this biomedical example is much more harmful than a *FP* result. Specifically, we have considered that a false negative classification is exactly 10 times more serious than a false positive classification ($CFN = 10$, $CFP = 1$). Since the Shapiro-Wilk normality test indicated that both groups could be assumed normally distributed (after a Box-Cox transformation of the data), the GPQ method is more adequate than the EL method in this case. For the sake of illustration, we have changed the default value of the number of replicates associated to the GPQ method by setting `control = control.gsym.point(I = 1500)` in the call to the `gsym.point()` function. The default value for this parameter is $I = 2500$.

Below we show the numerical results obtained by means of the `summary.gsym.point()` function, which can be abbreviated by the `summary()` method:

```
> summary(prostate.cutpoint1)

*****
OPTIMAL CUTOFF: GENERALIZED SYMMETRY POINT
*****

Call:
gsym.point(methods = "GPQ", data = prostate, marker = "marker",
  status = "status", tag.healthy = 0, categorical.cov = NULL,
  CFN = 10, CFP = 1, control = control.gsym.point(I = 1500),
  confidence.level = 0.95, trace = FALSE, seed = TRUE, value.seed = 3,
  verbose = TRUE)
```

According to the Shapiro-Wilk normality test, the marker can not be considered normally distributed in both groups. However, after transforming the marker using the Box-Cox transformation estimate, the Shapiro-Wilk normality test indicates that the transformed marker can be considered normally distributed in both groups.

Box-Cox lambda estimate = -1.2494

Shapiro-Wilk test p-values

	Group 0	Group 1
Original marker	0.0000	0.0232
Box-Cox transformed marker	0.3641	0.2118

Area under the ROC curve (AUC): 0.725

METHOD: GPQ

	Estimate	95% CI lower limit	95% CI upper limit
cutoff	51.9522523	46.8013315	57.3009307
Specificity	0.3233012	0.1420636	0.5191686
Sensitivity	0.9323301	0.9142064	0.9519169

In this case, the numerical output obtained with the `summary.gsym.point()` function shows:

1. An informative message indicating that the original data can not be assumed normally distributed in both groups, but the Box-Cox transformed data can be considered normally distributed in both groups, according to the Shapiro-Wilk normality test.
2. The estimated value of the Box-Cox power lambda.
3. The estimated value of the area under the ROC curve (AUC).
4. Information corresponding to the generalized symmetry point: the point estimates and the GPQ based $100(1 - \alpha)\%$ confidence intervals (where α is the signification level) of the generalized symmetry point in the scale of the APBS marker and its corresponding sensitivity and specificity measures. By default, confidence intervals are computed for a confidence level $1 - \alpha$ of 0.95. However, this value can be changed in the `confidence.level` argument of the main `gsym.point()` function.

As can be checked with the command below, the `prostate.cutpoint1` object yields a list with the following components: "GPQ", "methods", "call", and "data".

```
> names(prostate.cutpoint1)
[1] "GPQ"      "methods"  "call"     "data"
```

The first component, "GPQ", contains the results associated with the GPQ method.

```
> names(prostate.cutpoint1$GPQ)
[1] "Global"

> names(prostate.cutpoint1$GPQ$Global)
[1] "optimal.result"      "AUC"
[3] "rho"                 "lambda"
[5] "normality.transformed" "pvalue.healthy"
[7] "pvalue.diseased"     "pvalue.healthy.transformed"
[9] "pvalue.diseased.transformed"
```

The "methods" component is a character vector with the value of the argument methods used in the call, "call" is the matched call, and "data" is the data frame used in the analysis:

```
> prostate.cutpoint1$methods
[1] "GPQ"

> prostate.cutpoint1$call
gsym.point(methods = "GPQ", data = prostate, marker = "marker",
  status = "status", tag.healthy = 0, categorical.cov = NULL,
  CFN = 10, CFP = 1, control = control.gsym.point(I = 1500),
  confidence.level = 0.95, trace = FALSE,
  seed = TRUE, value.seed = 3, verbose = TRUE)

> prostate.cutpoint1$data
  marker status
1     40      0
2     40      0
3     46      0

[...]
```

```
51 99 1
52 126 1
53 136 1
```

We list below the elements of the first component, "GPQ", of the `prostate.cutpoint1` object and show the corresponding R-based commands that allow end-users to directly access these elements:

1. "optimal.result" is a list with the generalized symmetry point and its sensitivity and specificity accuracy measures with the corresponding $100(1 - \alpha)\%$ confidence intervals.

```
> prostate.cutpoint1$GPQ$Global$optimal.result
$cutoff
  Value      ll      ul
1 51.95225 46.80133 57.30093
```

```
$Specificity
  Value      ll      ul
1 0.3233012 0.1420636 0.5191686
```

```
$Sensitivity
  Value      ll      ul
1 0.9323301 0.9142064 0.9519169
```

2. "AUC" is the numerical value of the area under the ROC curve.

```
> prostate.cutpoint1$GPQ$Global$AUC
[1] 0.725
```

3. "rho" is the numerical value of $\rho = C_{FP}/C_{FN}$ (in this case $\rho = 0.1$).

```
> prostate.cutpoint1$GPQ$Global$rho
[1] 0.1
```

4. "lambda" is the numerical value of the power used in the Box-cox transformation of the GPQ method.

```
> prostate.cutpoint1$GPQ$Global$lambda
[1] -1.249428
```

5. "normality.transformed" is a character string indicating if the transformed marker values by the Box-Cox transformation are normally distributed ("yes") or not ("no").

```
> prostate.cutpoint1$GPQ$Global$normality.transformed
[1] "yes"
```

6. "pvalue.healthy" is the numerical value of the p -value obtained by the Shapiro-Wilk normality test for checking the normality assumption of the marker in the healthy population.

```
> prostate.cutpoint1$GPQ$Global$pvalue.healthy
[1] 3.276498e-07
```

7. "pvalue.diseased" is the numerical value of the p -value obtained by the Shapiro-Wilk normality test for checking the normality assumption of the marker in the diseased population.

```
> prostate.cutpoint1$GPQ$Global$pvalue.diseased
[1] 0.02323895
```

8. "pvalue.healthy.transformed" is the numerical value of the p -value obtained by the Shapiro-Wilk normality test for checking the normality assumption of the Box-Cox transformed marker in the healthy population.

```
> prostate.cutpoint1$GPQ$Global$pvalue.healthy.transformed
[1] 0.3640662
```

9. "pvalue.diseased.transformed" is the numerical value of the p -value obtained by the Shapiro-Wilk normality test for checking the normality assumption of the Box-Cox transformed marker in the diseased population.

```
> prostate.cutpoint1$GPQ$Global$pvalue.diseased.transformed
[1] 0.2118137
```

Similarly to the previous example, if we now consider the EL method instead of the GPQ method for estimating the generalized symmetry point and its accuracy measures, an informative message is shown by the package, advising the user that the GPQ method is more suitable for this dataset due to the fact that the Box-Cox transformed marker can be assumed normally distributed in both groups.

```
> prostate.cutpoint2 <- gsym.point(methods = "EL", data = prostate,
+   marker = "marker", status = "status", tag.healthy = 0,
+   categorical.cov = NULL, CFN = 10, CFP = 1,
+   control = control.gsym.point(B = 999), confidence.level = 0.95,
+   trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)
```

According to the Shapiro-Wilk normality test, the marker can not be considered normally distributed in both groups. However, after transforming the marker using the Box-Cox transformation estimate, the Shapiro-Wilk normality test indicates that the transformed marker can be considered normally distributed in both groups. Therefore the GPQ method would be more suitable for this dataset.

Box-Cox lambda estimate = -1.2494

Shapiro-Wilk test p-values

	Group 0	Group 1
Original marker	0.0000	0.0232
Box-Cox transformed marker	0.3641	0.2118

For the sake of illustration, we have set `control = control.gsym.point(B = 999)` in the previous call to change the default value of the number of bootstrap replicates `B` required in the empirical likelihood method. This parameter is `B = 499` by default.

The results obtained with the EL method are the following.

```
> summary(prostate.cutpoint2)
```

```
*****
OPTIMAL CUTOFF: GENERALIZED SYMMETRY POINT
*****
```

Call:

```
gsym.point(methods = "EL", data = prostate, marker = "marker",
  status = "status", tag.healthy = 0, categorical.cov = NULL,
  CFN = 10, CFP = 1, control = control.gsym.point(B = 999),
  confidence.level = 0.95, trace = FALSE, seed = TRUE, value.seed = 3,
  verbose = TRUE)
```

According to the Shapiro-Wilk normality test, the marker can not be considered normally distributed in both groups. However, after transforming the marker using the Box-Cox transformation estimate, the Shapiro-Wilk normality test indicates that the transformed marker can be considered normally distributed in both groups. Therefore the GPQ method would be more suitable for this dataset.

Box-Cox lambda estimate = -1.2494

Shapiro-Wilk test p-values

	Group 0	Group 1
Original marker	0.0000	0.0232
Box-Cox transformed marker	0.3641	0.2118

Area under the ROC curve (AUC): 0.725

METHOD: EL

	Estimate	95% CI lower limit	95% CI upper limit
cutoff	49.2249839	45.39058266	58.7032623

Specificity	0.2451690	0.09891113	0.5269153
Sensitivity	0.9245169	0.90989111	0.9526915

Figure 2 shows the graphical output corresponding to any of the two "gsym.point" class objects previously created, prostate.cutpoint1 and prostate.cutpoint2, as generated by means of the plot() method. For instance, the code below produces Figure 2.

```
> plot(prostate.cutpoint1)
```

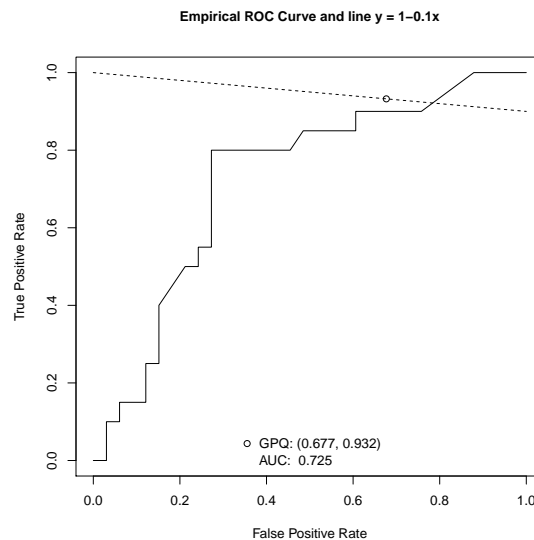


Figure 2: Graphical output for the prostate cancer dataset.

Coronary artery disease dataset

We now consider a study conducted on 141 patients (96 with coronary lesions and 45 with non-stenotic coronaries) admitted to the cardiology department of a teaching hospital in Galicia (Northwest Spain) for evaluating chest pain or cardiovascular disease, where the leukocyte elastase determination was investigated as a potential clinical marker for the diagnosis of coronary artery disease (Amaro et al., 1995). Since in this biomedical example there is available information regarding the gender of the patient (female or male), we will illustrate the practical application of the **GsymPoint** package to these data taking into account the covariate gender, that is, the generalized symmetry point will be computed for each gender in the scale of the elastase concentration to diagnose coronary artery disease (CAD). From here on, we will refer to this dataset as elastase.

First of all, we need to load the **GsymPoint** package and the corresponding elastase dataset:

```
> library("GsymPoint")
> data("elastase")
```

Now, for computing the generalized symmetry point in the elastase scale taking into account the categorical covariate gender, we simply have to include categorical.cov = "gender" in the call below.

```
> elastase.gender.cutpoint1 <- gsym.point(methods = "GPQ",
+   data = elastase, marker = "elas", status = "status",
+   tag.healthy = 0, categorical.cov = "gender", CFN = 10, CFP = 1,
+   control = control.gsym.point(), confidence.level = 0.95,
+   trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)
```

In this case we are interested in having a high sensitivity, that is, a low number of false negatives. Therefore, the same values as in the previous prostate cancer dataset were considered for the misclassification costs, CFN = 10 and CFP = 1, that is, a false negative result is regarded as 10 times more serious than a false positive one. Since the elastase concentration in females and males follow the

Box-Cox family in both CAD and non-CAD groups according to the Shapiro-Wilk normality test, the GPQ method of estimation is more adequate than the EL method in this case.

The `elastase.gender.cutpoint1` object produced by the previous call is a list with the following components:

```
> names(elastase.gender.cutpoint1)
[1] "GPQ"      "methods"  "levels.cat" "call"
[5] "data"
```

Similarly to the previous datasets, the "methods" component is a character vector with the value of the argument methods used in the call, "call" is the matched call and "data" is the data frame used in the analysis. However, now there is an extra component, "levels.cat", a character vector indicating the levels of the categorical covariate ("Female" and "Male" in this case). Besides, the first component, "GPQ", that includes the results associated with the GPQ method is itself a two-component list with "Female" and "Male" as can be seen below.

```
> names(elastase.gender.cutpoint1$GPQ)
[1] "Female" "Male"
```

For each level of the categorical covariate, you get a list with nine components. For instance, for the subgroup of males, `elastase.gender.cutpoint1GPQMale` contains the following components:

```
> names(elastase.gender.cutpoint1$GPQ$Male)
[1] "optimal.result"      "AUC"
[3] "rho"                 "lambda"
[5] "normality.transformed" "pvalue.healthy"
[7] "pvalue.diseased"     "pvalue.healthy.transformed"
[9] "pvalue.diseased.transformed"
```

The "optimal.result" component of `elastase.gender.cutpoint1GPQMale` is a list with the point estimates and $100(1 - \alpha)\%$ confidence intervals of the generalized symmetry point in the scale of the elastase concentration and its associated sensitivity and specificity accuracy measures in the ROC space corresponding to the subgroup of individuals that are males.

```
> elastase.gender.cutpoint1$GPQ$Male$optimal.result
$cutoff
  Value      ll      ul
1 20.72776 18.08961 23.49228

$Specificity
  Value      ll      ul
1 0.2739826 0.1345484 0.4326794

$Sensitivity
  Value      ll      ul
1 0.9273983 0.9134548 0.9432679
```

The "AUC" component of `elastase.gender.cutpoint1GPQMale` is the numerical value of the area under the ROC curve for the subgroup of individuals that are males.

```
> elastase.gender.cutpoint1$GPQ$Male$AUC
[1] 0.7216855
```

The "rho" component of `elastase.gender.cutpoint1GPQMale` is the numerical value of the ratio $\rho = \frac{C_{FP}}{C_{FN}}$, which is the same for females and males.

```
> elastase.gender.cutpoint1$GPQ$Male$rho
[1] 0.1
```

The "lambda" component of `elastase.gender.cutpoint1GPQMale` is the estimated numerical value of the power in the Box-Cox transformation obtained when considering only the subgroup of individuals that are males.

```
> elastase.gender.cutpoint1$GPQ$Male$lambda
[1] -0.04277911
```

The "normality.transformed" component of `elastase.gender.cutpoint1GPQMale` is a character string indicating if the Box-Cox transformed marker values in the subgroup of males are normally distributed ("yes") or not ("no").

```
> elastase.gender.cutpoint1$GPQ$Male$normality.transformed
[1] "yes"
```

The "pvalue.healthy" component of `elastase.gender.cutpoint1GPQMale` is the p -value of the Shapiro-Wilk normality test used to check the normality assumption of the marker in the healthy population of males.

```
> elastase.gender.cutpoint1$GPQ$Male$pvalue.healthy
[1] 0.5866506
```

The "pvalue.diseased" component of `elastase.gender.cutpoint1GPQMale` is the p -value of the Shapiro-Wilk normality test used to check the normality assumption of the marker in the diseased population of males.

```
> elastase.gender.cutpoint1$GPQ$Male$pvalue.diseased
[1] 5.44323e-09
```

Similarly, the "pvalue.healthy.transformed" and "pvalue.diseased.transformed" components are the p -values of the Shapiro-Wilk normality test used to check the normality assumption of the Box-Cox transformed marker in, respectively, the healthy and diseased populations of males.

```
> elastase.gender.cutpoint1$GPQ$Male$pvalue.healthy.transformed
[1] 0.06656483
> elastase.gender.cutpoint1$GPQ$Male$pvalue.diseased.transformed
[1] 0.2147409
```

A summary of the numerical results is shown below. In this case, the results obtained are shown for each level of the categorical covariate gender, that is, for females and males. However, similarly to previous examples, the output shows first the AUC value and then information related to the generalized symmetry point (point estimates and $100(1 - \alpha)\%$ confidence intervals obtained for each method of estimation specified in the main `gsym.point()` function). By default, confidence intervals are computed for a confidence level $1 - \alpha$ of 0.95, but this value can be changed directly in the `confidence.level` argument of the `gsym.point()` function.

```
> summary(elastase.gender.cutpoint1)
```

```
*****
OPTIMAL CUTOFF: GENERALIZED SYMMETRY POINT
*****
```

Call:

```
gsym.point(methods = "GPQ", data = elastase, marker = "elas",
  status = "status", tag.healthy = 0, categorical.cov = "gender",
  CFN = 10, CFP = 1, control = control.gsym.point(), confidence.level = 0.95,
  trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)
```

```
*****
Female
*****
```

According to the Shapiro-Wilk normality test, the marker can be considered normally distributed in both groups.

Shapiro-Wilk test p-values

```
                Group 0 Group 1
Original marker 0.0837 0.9077
```

Area under the ROC curve (AUC): 0.818

METHOD: GPQ

	Estimate	95% CI lower limit	95% CI upper limit
cutoff	25.0929510	12.3370641	34.0526540
Specificity	0.4246091	0.1460251	0.6618634
Sensitivity	0.9424609	0.9146025	0.9661863

```
*****
Male
*****
According to the Shapiro-Wilk normality test, the marker can not
be considered normally distributed in both groups.
However, after transforming the marker using the Box-Cox
transformation estimate, the Shapiro-Wilk normality test
indicates that the transformed marker can be considered
normally distributed in both groups.
```

Box-Cox lambda estimate = -0.0428

Shapiro-Wilk test p-values

	Group 0	Group 1
Original marker	0.5867	0.0000
Box-Cox transformed marker	0.0666	0.2147

Area under the ROC curve (AUC): 0.722

METHOD: GPQ

	Estimate	95% CI lower limit	95% CI upper limit
cuttoff	20.7277556	18.0896126	23.4922813
Specificity	0.2739826	0.1345484	0.4326794
Sensitivity	0.9273983	0.9134548	0.9432679

If we consider now the EL method in the call to the main `gsym.point()` function, the **GsymPoint** package will show an informative message indicating that the GPQ method would be more adequate in this case because for the two levels of the categorical covariate gender, either the original marker or the Box-Cox transformed marker in both diseased and healthy populations can be assumed normally distributed according to the Shapiro-Wilk normality test:

```
> elastase.gender.cutpoint2 <- gsym.point (methods = "EL", data = elastase,
+   marker = "elas", status = "status", tag.healthy = 0,
+   categorical.cov = "gender", CFN = 10, CFP = 1,
+   control = control.gsym.point(), confidence.level = 0.95,
+   trace = FALSE, seed = TRUE, value.seed = 3, verbose = TRUE)
```

Female :

According to the Shapiro-Wilk normality test, the marker can be considered normally distributed in both groups. Therefore the GPQ method would be more suitable for this dataset.

Shapiro-Wilk test p-values

	Group 0	Group 1
Original marker	0.0837	0.9077

Male :

According to the Shapiro-Wilk normality test, the marker can not be considered normally distributed in both groups. However, after transforming the marker using the Box-Cox transformation estimate, the Shapiro-Wilk normality test indicates that the transformed marker can be considered normally distributed in both groups. Therefore the GPQ method would be more suitable for this dataset.

Box-Cox lambda estimate = -0.0428

Shapiro-Wilk test p-values

	Group 0	Group 1
Original marker	0.5867	0.0000
Box-Cox transformed marker	0.0666	0.2147

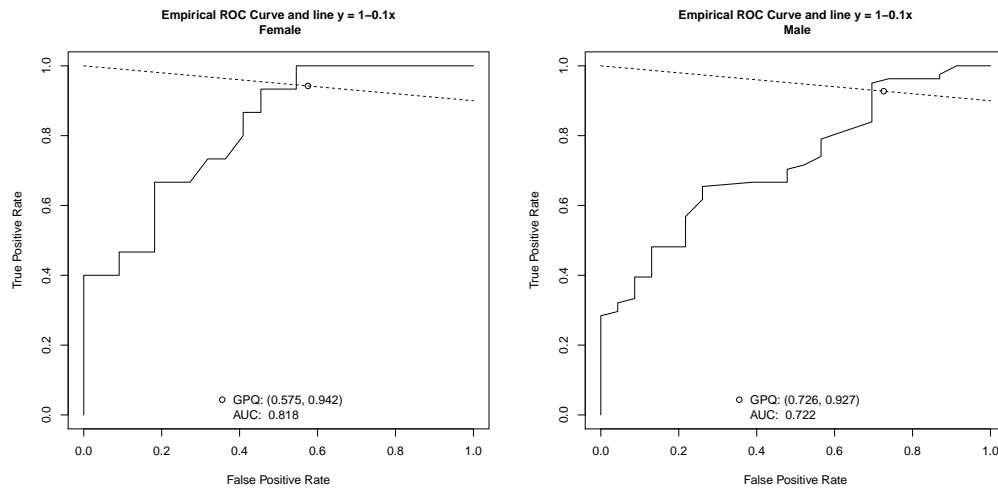


Figure 3: Graphical output for the coronary artery disease dataset, distinguishing by gender.

The graphical results can be obtained by means of the `plot.gsym.point()` function or merely the `plot()` method. For instance, the following command

```
> plot(elastase.gender.cutpoint1)
```

yields the graphical output shown in Figure 3, where the empirical ROC curve of the elastase concentration is represented separately for females and males, together with the line $y = 1 - \rho x$ and the intersection point associated to the generalized symmetry point for detecting CAD. This is the output that appears by default for females and males, respectively. However, as usual, the end-user can change several graphical parameters, such as legends, colors, etc.

Discussion

The selection of optimal cutpoints in the scale of a quantitative diagnostic test or biomarker that can help in the diagnosis of a disease is an important issue in the clinical sciences. Depending on the main objective of such selection, several criteria of optimality have been proposed in the literature to carry out this task. One of the most popular in clinical practice is based on selecting the cutoff that provides the same sensitivity and specificity, known in the literature as the equivalence or symmetry point (Greiner, 1995; Defreitas et al., 2004; Adlhoch et al., 2011). However, this cutpoint is not valid in scenarios where the severity of misclassifying a diseased patient is not the same as the severity of misclassifying a healthy patient, which is normally the case in practice. For instance, in cancer disease, a false negative decision is in general more serious than a false positive decision. Hence, when selecting an optimal cutpoint it is very important to take into account the costs of the different incorrect diagnostic decisions. For this reason, the generalized symmetry point, a generalization of the symmetry point that incorporates the costs of the misclassifications is very adequate and useful in practice (López-Ratón et al., 2016). Although there are several R packages that implement different criteria for selecting the optimal cutoff point such as **PresenceAbsence** (Freeman and Moisen, 2008), **DiagnosisMed** (Brasil, 2010), **pROC** (Robin et al., 2011) and **OptimalCutpoints** (López-Ratón and Rodríguez-Álvarez, 2014; López-Ratón et al., 2014), up to our knowledge, none of them includes the criterion based on the generalized symmetry point nor recent estimation approaches such as the GPQ and EL methods that provide more efficient estimates than the empirical ones (Molanes-López and Letón, 2011; Lai et al., 2012). In order to avoid that the use of the generalized symmetry point is limited in the clinical practice by the lack of software that implements all necessary computations to estimate it, we have developed the **Gsympoint** package, a user-friendly R package that fills this gap. As it has been illustrated in this paper, the **GsymPoint** package allows the possibility that the generalized symmetry point is straightforwardly estimated for each level of a certain categorical covariate that represents an individual characteristic such as gender, age or disease severity, and that may influence the discrimination capacity of the diagnostic test (Pepe, 2003). Possible interesting extensions of the **GsymPoint** package could be taken into account. For instance, the implementation of this same methodology to estimate other accuracy measures such as predictive values or diagnostic likelihood ratios, the incorporation of more efficient methods for estimating the generalized symmetry point and its accuracy measures, and the extension of this methodology to other more complex scenarios

where, for instance, the diagnostic test is subject to the measurement of error, there is presence of partial disease verification (see Alonzo, 2014, and references therein), the disease status evolves over time and the disease onset time is subject to censoring (Rota et al., 2015) or there are continuous covariates available that may affect the diagnostic capacity of the biomarker and that should be taken into consideration.

Acknowledgments

This research has been supported by several Grants from the Spanish Ministry of Science and Innovation. M. López-Ratón and C. Cadarso-Suárez acknowledge support to MTM2011-15849-E, MTM2011-28285-C02-00, MTM2014-52975-C2-1-R and MTM2015-69068-REDT. E.M. Molanes-López acknowledges support to MTM2011-28285-C02-02, ECO2011-25706, MTM2011-15849-E and MTM2015-69068-REDT. E. Letón acknowledges support to MTM2011-15849-E, MTM2011-28285-C02-02, PI13/02446 and MTM2015-69068-REDT.

Bibliography

- C. Adlhoch, M. Kaiser, M. Hoehne, A. M. Marques, I. Stefas, F. Veas, and H. Ellerbrok. Highly sensitive detection of the Group A rotavirus using apolipoprotein H-coated ELISA plates compared to quantitative real-time PCR. *Virology Journal*, 8:63, 2011. [p263, 280]
- TA. Alonzo. Verification bias-impact and methods for correction when assessing accuracy of diagnostic tests. *REVSTAT-Statistical Journal*, 12(1):67–83, 2014. [p281]
- A. Amaro, F. Gude, R. González-Juanatey, F. Iglesias, F. Fernández-Vázquez, J. García-Acuña, and M. Gil. Plasma leukocyte elastase concentration in angiographically diagnosed coronary artery disease. *European Heart Journal*, 16(5):615–622, 1995. [p276]
- D. Bamber. The area above the ordinal dominance graph and the area below the receiver operating graph. *Journal of Mathematical Psychology*, 12(4):387–415, 1975. [p262]
- P. Brasil. *DiagnosisMed: Diagnostic Test Accuracy Evaluation for Medical Professionals*, 2010. URL <https://CRAN.R-project.org/src/contrib/Archive/DiagnosisMed/>. R package version 0.2.3. [p263, 280]
- S. Cantor, C. Sun, G. Tortolero-Luna, R. Richards-Kortum, and M. Follen. A comparison of C/B ratios from studies using Receiver Operating Characteristic curve analysis. *Journal of Clinical Epidemiology*, 52(9):885–892, 1999. [p263]
- J. Carpenter and J. Bithell. A practical guide for medical statisticians. *Statistics in Medicine*, 19:1141–1164, 2000. [p267]
- GA. Defreitas, PE. Zimmern, GE. Lemack, and SF. Shariat. Refining diagnosis of anatomic female bladder outlet obstruction: Comparison of pressureflow study parameters in clinically obstructed women with those of normal controls. *Urology*, 64(4):675–679, 2004. [p263, 280]
- SH. Feinstein. The accuracy of diver sound localization by pointing. *Understanding Biomedical Research*, 2(3):173–184, 1975. [p262]
- EA. Freeman and G. Moisen. PresenceAbsence: An R package for Presence Absence analysis. *Journal of Statistical Software*, 23(11):1–31, 2008. URL <http://www.jstatsoft.org/v23/i11/>. [p263, 280]
- RJ. Gallop, P. Crits-Christoph, LR. Muenz, and XM. Tu. Determination and interpretation of the optimal operating point for ROC curves derived through generalized linear models. *Understanding Statistics*, 2(4):219–242, 2003. [p263]
- M. Greiner. Two-graph receiver operating characteristic (TG-ROC): A Microsoft-EXCEL template for the selection of cut-off values in diagnostic tests. *Journal of Immunological Methods*, 185(1):145–146, 1995. [p263, 280]
- J. Hanley and J. McNeil. The meaning and use of the area under a Receiver Operating Characteristic (ROC) curve. *Radiology*, 143:29–36, 1982. [p262]
- A. Jiménez-Valverde. Insights into the Area under the Receiver Operating Characteristic curve (AUC) as a discrimination measure in species distribution modelling. *Global Ecology and Biogeography*, 21: 498–507, 2012. [p263]

- A. Jiménez-Valverde. Threshold-dependence as a desirable attribute for discrimination assessment: Implications for the evaluation of species distribution models. *Biodiversity Conservation*, 23(2): 369–385, 2014. [p263]
- CY. Lai, L. Tian, and EF. Schisterman. Exact confidence interval estimation for the Youden index and its corresponding optimal cut-point. *Computational Statistics & Data Analysis*, 56(5):1103–1114, 2012. [p263, 264, 265, 280]
- CT. Le. A solution for the most basic optimization problem associated with an ROC curve. *Statistical Methods in Medical Research*, 15(6):571–584, 2006. [p272]
- M. López-Ratón and MX. Rodríguez-Álvarez. *OptimalCutpoints: Computing Optimal Cutpoints in Diagnostic Tests*, 2014. URL <https://CRAN.R-project.org/package=OptimalCutpoints>. R package version 1.1-1. [p263, 280]
- M. López-Ratón, MX. Rodríguez-Álvarez, C. Cadarso-Suárez, and F. Gude-Sampedro. OptimalCutpoints: An R package for selecting optimal cutpoints in diagnostic tests. *Journal of Statistical Software*, 61(8):1–36, 2014. URL <http://www.jstatsoft.org/v61/i08/>. [p263, 280]
- M. López-Ratón, C. Cadarso-Suárez, EM. Molanes-López, and E. Letón. Confidence intervals for the symmetry point: An optimal cutpoint in continuous diagnostic tests. *Pharmaceutical Statistics*, 15(2): 178–192, 2016. [p263, 264, 265, 280]
- M. López-Ratón, C. Cadarso-Suárez, EM. Molanes-López, and E. Letón. *GsymPoint: Estimation of the Generalized Symmetry Point, an Optimal Cutpoint in Continuous Diagnostic Tests*, 2017. URL <https://CRAN.R-project.org/package=GsymPoint>. R package version 1.1. [p263, 265]
- CE. Metz. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8(4):183–298, 1978. [p262]
- EM. Molanes-López and E. Letón. Inference of the Youden index and associated threshold using empirical likelihood for quantiles. *Statistics in Medicine*, 30(19):2467–2480, 2011. [p263, 264, 265, 280]
- EM. Molanes-López, I. Van Keilegom, and N. Veraverbeke. Empirical likelihood for non-smooth criterion functions. *Scandinavian Journal of Statistics*, 36(3):413–432, 2009. [p264]
- M. Pepe. *The Statistical Evaluation of Medical Tests for Classification and Prediction*. Oxford University Press, New York, 2003. [p263, 280]
- DL. Riddle and PW. Stratford. Interpreting validity indexes for diagnostic tests: An illustration using the Berg balance test. *Physical Therapy*, 79(10):939–948, 1999. [p263]
- X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, JC. Sanchez, and M. Müller. pROC: An open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics*, 12:77, 2011. [p263, 280]
- M. Rota, L. Antolini, and MG. Valsecchi. Optimal cut-point definition in biomarkers: The case of censored failure time outcome. *BMC Medical Research Methodology*, 15(24):1–11, 2015. [p281]
- H. Schäfer. Constructing a cut-off point for a quantitative diagnostic test. *Statistics in Medicine*, 8(11): 1381–1391, 1989. [p262]
- F. Subtil and M. Rabilloud. An enhancement of ROC curves made them clinically relevant for diagnostic-test comparison and optimal-threshold determination. *Journal of Clinical Epidemiology*, 68: 752–759, 2015. [p263]
- JA. Swets. ROC analysis applied to the evaluation of medical imaging techniques. *Investigative Radiology*, 14:109–121, 1979. [p262]
- JA. Swets and RM. Pickett. *Evaluation of Diagnostic Systems: Methods from Signal Detection Theory*. Academic Press, New York, 1982. [p262]
- JA. Swets and JB. Swets. ROC approach to cost/benefit analysis. In *Proceedings of the Sixth IEEE Conference on Computer Applications in Radiology*, pages 203–206. Reprinted in Ripley KL, Murray A, eds, 1979. [p262]
- DR. Thomas and GL. Grunkemeier. Confidence interval estimation of survival probabilities for censored data. *Journal of the American Statistical Association*, 70(352):865–871, 1975. [p263, 264, 265]
- ES. Venkatraman and CB. Begg. A distribution-free procedure for comparing Receiver Operating Characteristic curves from a paired experiment. *Biometrika*, 83(4):835–848, 1996. [p268]

- J. Vermont, J.L. Bosson, P. François, C. Robert, A. Rueff, and J. Demongeot. Strategies for graphical threshold determination. *Computer Methods and Programs in Biomedicine*, 35(2):141–150, 1991. [p263]
- S. Weerahandi. Generalized confidence intervals. *Journal of the American Statistical Association*, 88: 899–905, 1993. [p263, 264, 265]
- S. Weerahandi. *Exact Statistical Methods for Data Analysis*. Springer-Verlag, 1995. [p263, 264, 265]
- WJ. Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950. [p262]
- H. Zhou and G. Qin. Confidence intervals for the difference in paired Youden indices. *Pharmaceutical Statistics*, 12(1):17–27, 2013. [p264, 265]

Mónica López-Ratón
Biostatistics Unit, Department of Statistics and Operations Research
Universidad de Santiago de Compostela
San Francisco s/n
15782 Santiago de Compostela, Spain
monica.lopez.raton@edu.xunta.es

Elisa M. Molanes-López
Department of Statistics and Operations Research
Universidad Complutense de Madrid
Plaza Ramón y Cajal s/n
28040 Ciudad Universitaria-Madrid, Spain
emolanes@ucm.es

Emilio Letón
Department of Artificial Intelligence
Universidad Nacional de Educación a Distancia
C/ Juan del Rosal 16
28040 Madrid, Spain
emilio.leton@dia.uned.es

Carmen Cadarso-Suárez
Biostatistics Unit, Department of Statistics and Operations Research
Universidad de Santiago de Compostela
San Francisco s/n
15782 Santiago de Compostela, Spain
carmen.cadarso@usc.es

autoimage: Multiple Heat Maps for Projected Coordinates

by Joshua P. French

Abstract Heat maps are commonly used to display the spatial distribution of a response observed on a two-dimensional grid. The **autoimage** package provides convenient functions for constructing multiple heat maps in unified, seamless way, particularly when working with projected coordinates. The **autoimage** package natively supports: 1. automatic inclusion of a color scale with the plotted image, 2. construction of heat maps for responses observed on regular or irregular grids, as well as non-gridded data, 3. construction of a matrix of heat maps with a common color scale, 4. construction of a matrix of heat maps with individual color scales, 5. projecting coordinates before plotting, 6. easily adding geographic borders, points, and other features to the heat maps. After comparing the **autoimage** package's capabilities for constructing heat maps to those of existing tools, a carefully selected set of examples is used to highlight the capabilities of the **autoimage** package.

Introduction

A *heat map* is a graphic commonly used to visualize the spatial distribution of a response observed on a two-dimensional grid. For example, a climate scientist may want to visualize the spatial distribution of an aerosol across a grid of locations covering the study area. In general, a heat map can be used to display response variation as a function of two covariates varying in a grid-like pattern, such as the tensile strength of a substance as a function of heat and pressure. Each grid point in a heat map is associated with a polygon, and each polygon is colored using a color scheme related to the level of the response at each grid point. The plot of colored polygons is an *image* or *heat map*. A color scale relating the colors to their associated levels often accompanies the image. For clarity, we use the term *image* to refer to the plot of colored polygons alone and the term *heat map* to refer to the combination of image(s) and color scale.

The **autoimage** package (French, 2017) makes it easy to plot a sequence of heat maps with straightforward, native options for projection of geographical coordinates. The package makes it simple to add lines, points, and other features to the images, even when the coordinates are projected. The package allows for seamless creation of heat maps for data on regular or irregular grids, as well as data that is not on a grid.

As a quick introduction to some of the **autoimage** package's capabilities, we utilize the narccap data included in the **autoimage** package. The narccap data set comes from the North American Regional Climate Change Assessment Program (Mearns et al., 2009, 2012; Mearns and others, 2007, updated 2014), which is a program designed to model climate scenarios in North America (the United States, Canada, and northern Mexico) by coupling regional and global climate models. Specifically, the narccap data are the maximum daily surface air temperature (abbreviated tasmax) in degrees Kelvin (K) for the five consecutive days between May 15, 2041 and May 19, 2041. The data were simulated using the Canadian Regional Climate Model (Caya and Laprise, 1999) forced by the Community Climate System Model atmosphere-ocean general circular model (Collins et al., 2006). The data set contains lon, a 140×115 matrix of longitude coordinates, lat, a 140×115 matrix of latitude coordinates, and tasmax, a $140 \times 115 \times 5$ array, where each element of the third dimension of the array corresponds to the tasmax measurements of the respective day. We create the heat map shown in Figure 1 for the first four days of the narccap data by executing the command

```
autoimage(lon, lat, tasmax[, , 1:4])
```

The outline of the United States and Mexico are somewhat noticeable in the images of Figure 1, but this would be easier to see if the relevant national borders of the countries were included. This is accomplished in Figure 2, which is created by specifying a map in the previous command:

```
autoimage(lon, lat, tasmax[, , 1:4], map = "world")
```

Images and heat maps can be created in R using functions from several different R packages. We discuss six functions in more detail. They are

1. the `image` function in the **graphics** package,
2. the `filled.contour` function in the **graphics** package,
3. the `image.plot` function in the **fields** package (Douglas Nychka et al., 2015),
4. the `levelplot` function in the **lattice** package (Sarkar, 2008),

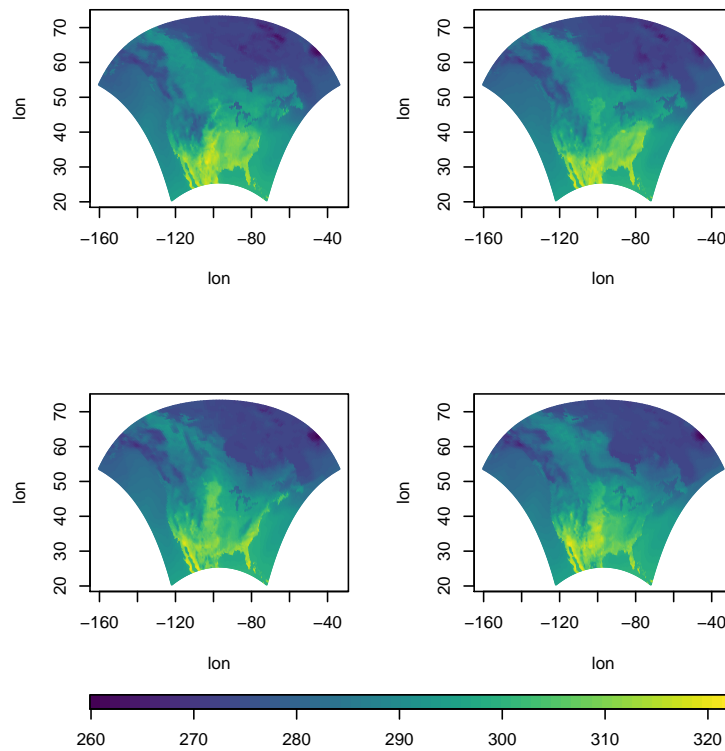


Figure 1: A heat map for the first four days of the narccap data.

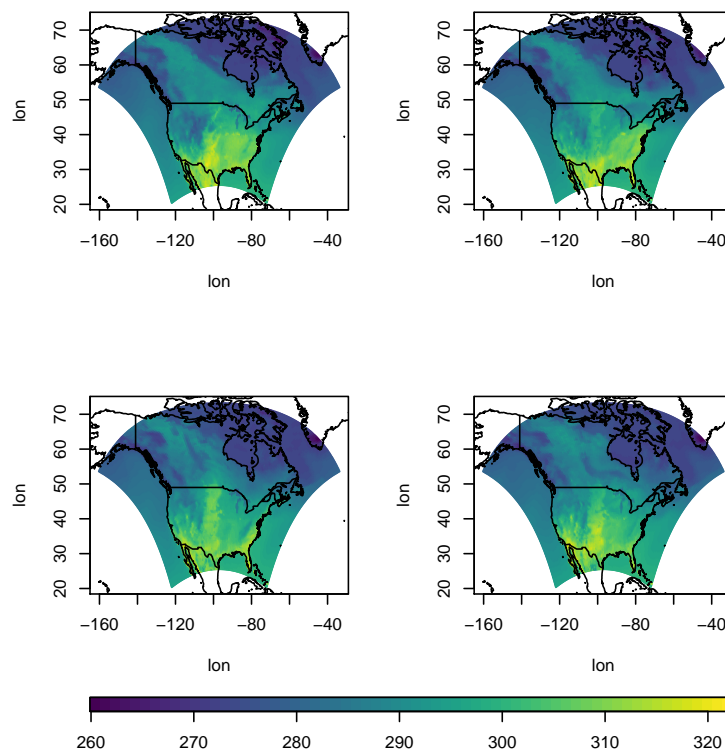


Figure 2: A heat map for the first four days of the narccap data, along with the relevant national borders.

5. the `splot` function in the `sp` package (Pebesma and Bivand, 2005; Bivand et al., 2015),
6. and the `geom_tile` function in the `ggplot2` package (Wickham, 2009).

In the next section, we compare the native capabilities of these tools with those of the `autoimage` package. We define the *native capabilities* of a function or package to be the capabilities of the function or package without loading additional packages or requiring additional user code. This will be followed by a section demonstrating the main capabilities of the `autoimage` package. We will conclude with a brief summary of conclusions in the final section.

Tools for creating heat maps in R

We now compare the capabilities of existing tools for creating heat maps in R.

We begin by discussing the `image` function included in the `graphics` package. This function is included in a standard R installation. The `image` function produces an image without a color scale. While one can discern spatial patterns from the plotted image, one cannot determine the magnitude of the response values without knowing more about the coloring scheme. Naturally, members of the R community have provided solutions to add this functionality. For example, the `spatstat` package (Baddeley and Turner, 2005) extends the `image` function using S3 methods to automatically include a color scale with the image when an `im` class object is plotted. The `image` function requires the responses to be on a *regular grid*. Specifically, if x and y are each an increasing sequence of values, then the coordinates of a regular grid are obtained by considering all combinations of the elements in x and y . Regular grids frequently become irregular when rotated or transformed to a different coordinate system. This is common for longitude/latitude coordinates that are transformed using a projection. When a matrix of images is desired, the user can modify the `mfrac` argument of the `par` function in the `graphics` package and then plot a sequence of images. However, the matrix of images will not include individual color scales or a common color scale. The `image` function has no capabilities related to coordinate projection. Existing images can be added to using standard `graphics` functions such as `lines` and `points`. It is worth noting that the standard R installation also includes a `heatmap` function in the `stats` package, but this displays a different result than the `image` function. The `heatmap` function documentation indicates that the `heatmap` function produces, "... a false color image ... with a dendrogram added to the left side and to the top. Typically, reordering of the rows and columns according to some set of values (row or column means) within the restrictions imposed by the dendrogram is carried out." Consequently, a heat map is sometimes called an *image plot* by researchers whose data analysis is primarily done in R. We will continue to use the more general terminology "heat map" to describe this type of graphic in what follows.

The `filled.contour` function in the `graphics` package produces an image with a vertical color scale on the right side of the image. The algorithm used to color the polygons in the `filled.contour` function differs slightly from the `image` function, but produces similar results. The `filled.contour` function requires the responses to be on a regular grid. The `filled.contour` function cannot natively produce a sequence of heat maps with a common color scale or individual color scales because the function internally calls the `layout` function. Coordinate projection is not natively supported by the `filled.contour` function. An existing image can be added to using `graphics` functions such as `lines`, `points`, or `text`.

Another popular tool for creating heat maps in R is the `image.plot` function in the `fields` package. The `image.plot` function retains the strengths of the `image` function while providing additional functionality. The `image.plot` function automatically includes a color scale with the plotted image. More uniquely, the `image.plot` function can natively produce heat maps for data observed on an "irregular grid". Let x be an $r \times c$ matrix of x coordinates and y be an $r \times c$ matrix of y coordinates. Then x and y define an irregular grid if $x[i, j] \leq x[i+1, j]$, $y[i, j] \leq y[i+1, j]$, $x[i, j] \leq x[i, j+1]$, and $y[i, j] \leq y[i, j+1]$ for all valid choices of i and j . The `image.plot` function can be used to construct a matrix of heat maps with individual color scales by specifying the `mfrac` argument of the `par` function and calling `image.plot` the desired number of times. The `image.plot` function cannot natively create a matrix of images with a common color scale. Though the `image.plot` function includes functionality to create images for geographically-referenced irregular grids, it does not include any native functionality for projecting coordinates. Features can be added to existing images using the `lines`, `points`, and related functions.

The `lattice` package provided the first well-known alternative to the base graphics system in R. The `lattice` package produces heat maps using the `levelplot` function. Images produced by the `levelplot` function automatically include a color scale. The `levelplot` function can only produce heat maps for responses observed on a regular grid. The `levelplot` function can natively create a matrix of heat maps sharing a common color scale. To obtain a matrix of heat maps with individual color scales, one must combine a sequence of `levelplot` function calls with the `grid.arrange` function in

the **gridExtra** package (Auguie, 2016). The `levelplot` function includes no functionality related to coordinate projection. Adding lines, points, or other features is usually accomplished by defining an appropriate panel function.

The **sp** package provides powerful classes for representing geographically-referenced data. The **sp** package can produce heat maps automatically for `SpatialGriddedDataFrame` class objects using the `splot` function. The `splot` function produces heat maps by extending the `levelplot` function in the **lattice** package, and the two functions have similar feature sets. One major difference between the two functions is that the **sp** package includes coordinate projection information within the `SpatialGriddedDataFrame`, so the `spTransform` function in the **sp** package can be used to easily project coordinates to a different system. However, if the projected set of coordinates is not regular, the `splot` function will no longer produce a heat map for the transformed data, but will instead produce a scatterplot of colored points with a related color scale.

The grammar of graphics plotting system implemented in the **ggplot2** package continues to grow in popularity. A heat map can be constructed using **ggplot2** by combining the `geom_tile` geometry (or the `geom_rect` or `geom_raster` geometries) with a `ggplot` object. The heat map automatically includes a color scale with the image. Similar to the `image`, `levelplot`, and `splot` functions, the **ggplot2** package requires locations to be on a regular grid. A matrix of images sharing a common color scale can be created directly in **ggplot2** using `facet_wrap` or `facet_grid`. A matrix of heat maps with individual color scales can be created by supplementing **ggplot2** with additional functionality provided by the **gridExtra** or **cowplot** (Wilke, 2016) packages. The `coord_map` function can be used to natively project geographic coordinates on a regular grid to an irregular grid. Additional features can be added to the images by specifying the appropriate geometry calls when creating the heat maps. These additional features are automatically projected if the `coord_map` function is utilized.

The **autoimage** package was created to easily produce a sequence of heat maps while natively working with projected coordinates, even when the data are not on a regular grid. The **autoimage** package utilizes the base R graphics system (like the `image`, `filled.contour`, and `image.plot` functions) to quickly and straightforwardly create heat maps. The `autoimage` function is the most important function in the **autoimage** package. The `autoimage` function automatically supplies a color scale with the constructed image(s). The `autoimage` function can automatically create heat maps for responses on an irregular grid, similar to the `image.plot` function. In fact, the **autoimage** function relies on the `poly.image` function from the **fields** package to create heat maps for this type of data, which is the function used by the `image.plot` function for the same purpose. Additionally, the **autoimage** package can natively create heat maps for non-gridded data by automatically interpolating the surface onto a regular grid before plotting using the **akima** package¹ (Akima and Gebhardt, 2016). The `autoimage` function can create a matrix of heat maps with either individual or shared color scales automatically, without the use of additional user code or R packages. Additionally, functionality for coordinate projection is available automatically in the `autoimage` function using the `mapproject` function in the **mapproj** package (McIlroy, 2015). Additional features can be added to the plotted images (even for projected coordinates) using the `plines`, `ppoints`, `ptext`, and related functions available in the **autoimage** package.

We now summarize the similarities and differences between the *native functionality* of the **autoimage** package and the previous tools discussed. Most of the tools automatically include a color scale with the plotted image, similar to the `autoimage` function. The **lattice**, **sp**, and **ggplot2** packages can create a sequence of heat maps with a common color scale. The `image.plot` function can be used in combination with the `par` function to create a matrix of heat maps with individual color scales. The **lattice**, **sp**, and **ggplot2** packages can be used in combination with the **gridExtra** package to produce a matrix of heat maps with individual color scales. The `autoimage` function has native functionality for creating a matrix of heat maps with either a common or individual color scales. In contrast, only the `autoimage` and `image.plot` functions can *natively* create heat maps for data on an irregular grid. Additionally, only the **autoimage** package can *natively* create heat maps for non-gridded data (by automatically interpolating the responses onto a regular grid before plotting). The **sp**, **ggplot2**, and **autoimage** packages support coordinate projection of longitude/latitude coordinates, though only the **ggplot2** and **autoimage** packages can produce heat maps if the projected coordinates result in an irregular grid. Table 1 summarizes the features of the heat map-generating tools mentioned above. If a feature is not natively available, but becomes available by using a relevant package or function, then the appropriate tool is listed.

The feature set for creating heat maps using the **ggplot2** package largely overlaps with the **autoimage** package, especially if additional packages are used to extend the functionality of **ggplot2**. A user may wonder whether the **autoimage** package has any other advantages that would support its use over the more well-known and broadly capable **ggplot2** package. As previously mentioned, **ggplot2**

¹The **akima** package has a restrictive license and must be installed manually. We plan to replace **akima** with a package offering similar functionality, but with a more permissive license, in a future **autoimage** release.

Package or function name	Color scale	Irregular grid	Non-gridded data	Shared color scale	Individual color scales	Coordinate projection
<code>image</code>						
<code>filled.contour</code>	x					
<code>fields</code>	x	x			par	
<code>lattice</code>	x			x	gridExtra	
<code>sp</code>	x			x	gridExtra	x
<code>ggplot2</code>	x			x	gridExtra	x
<code>autoimage</code>	x	x	x	x	x	x

Table 1: Feature comparison for several functions and packages used for creating heat maps. “x” indicates the tool natively includes that feature. A function or package name indicates that the functionality is easily obtainable using that function or package.

requires that the original (unprojected) data be observed on a regular grid, while the **autoimage** package supports heat map creation for data on regular and irregular grids, as well as non-gridded data. Another advantage of the **autoimage** package is that it renders a sequence of heat maps for projected coordinates faster than the **ggplot2** package. In order to compare the speed of heat map generation using the **autoimage** and **ggplot2** packages when natively projecting the coordinates before plotting, we constructed a matrix of 36 heat maps (with a common color scale) using coordinates observed on a regular 116×50 grid and projected the coordinates using the Lambert projection. Creating the heat maps on a mid-2012 MacBook Pro with a 2.6 GHz Intel Core i7 processor and 16 GB of RAM took roughly 4 *seconds* using the **autoimage** package, but over 12 *minutes* using the **ggplot2** package. This is by no means an exhaustive timing comparison, but the result suggests that when coordinate projections are utilized in creating a matrix of heat maps, the **autoimage** package can render the plot more quickly than the **ggplot2** package.

autoimage examples

We now examine the capabilities of the **autoimage** package in more detail. The most important functions in **autoimage** are the `pimage` and `autoimage` functions. We illustrate the basic usage of these functions using two data sets: the first is the irregularly-gridded `narccap` data previously discussed, while the second is a set of non-gridded geochemical measurements for 960 locations in the state of Colorado. The Colorado geochemical measurements were obtained by the United States Geological Survey (USGS) as a baseline for the natural variation in soil geochemistry in Colorado (Smith and Ellefsen, 2010). The data are stored as a data frame with 960 rows and 31 columns. `easting`, `northing`, `latitude`, and `longitude` variables are provided in the data frame, as well as Aluminum (Al), Calcium (Ca), Iron (Fe), and many more chemical measurements. The Colorado data are available in the `co` data set in the **gear** package (French, 2015).

The `autoimage` function is a generalization of the `pimage` function, so we discuss the `pimage` function first.

Basic usage of the `pimage` function

The most important arguments of the `pimage` function are `x`, `y`, and `z`. `x` and `y` are the coordinate locations and `z` is the responses associated with the coordinates. `x`, `y`, and `z` can have differing formats depending on the type of data to be plotted. If the data are observed on a regular grid, then `z` will be a matrix with dimensions matching the dimensions of the grid and `x` and `y` will be vectors of increasing values that define the grid lines. If the data are observed on an irregular grid, then `z` will be a matrix with dimensions matching the dimensions of the grid, and `x` and `y` will be matrices whose coordinates specify the `x` and `y` coordinates of each value in `z`. If the data are not on a grid, then `x` and `y` will be vectors specifying the coordinate locations, and `z` will be the vector of responses at each coordinate. If the data are not on a grid, then the data are automatically interpolated onto a grid before plotting. The command

```
pimage(x = lon, y = lat, z = tasmax[, , 1])
```

is used to create a heat map of the `tasmax` measurements for the first day of the `narccap` data. This heat map is shown in Figure 3. Recall that the `narccap` data are observed on an irregular grid. A

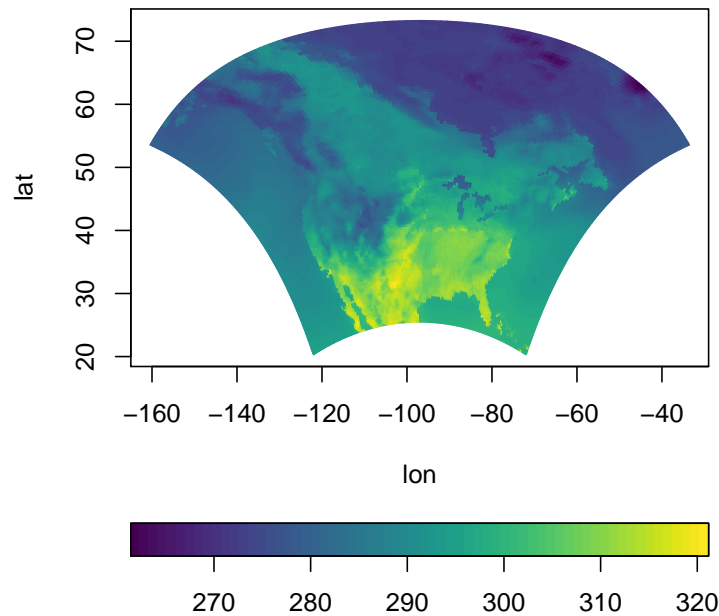


Figure 3: A heat map of the tasmax measurements for the first day of the narccap data.

heat map of the Aluminum measurements for the non-gridded co data set is created by executing the commands

```
data(co, package = 'gear')
pimage(co$longitude, co$latitude, co$Al)
```

with the resulting heat map shown in Figure 4.

We now discuss other basic arguments used by the `pimage` function.

The first two arguments we discuss are the `col` and `legend` arguments. The color scheme used for coloring a heat map is of great importance. The default color scheme used in the **autoimage** package is the **viridis** color scheme from the **viridisLite** package. This color scheme is, "... designed in such a way that [it] will analytically be perfectly perceptually-uniform, both in regular form and also when converted to black-and-white. [It is] also designed to be perceived by readers with the most common form of color blindness." (Garnier, 2016) The color scale can be modified by passing a vector of colors to the `col` argument through the `ellipses` argument (`...`), as in the `image` function in the **graphics** package. The orientation of the color scale can be changed using the `legend` argument. The default value is `legend = 'horizontal'`, which produces a color scale horizontally underneath the image. The color scale can be removed by specifying `legend = 'none'` or can be placed vertically along the right side of the image by specifying `legend = 'vertical'`. A heat map of the first day of narccap tasmax measurements with a vertical color scale using 6 colors from the magma color palette in the **viridisLite** package is created by executing the command

```
pimage(lon, lat, tasmax[,1], col = viridisLite::magma(6), legend = 'vertical')
```

The resulting heat map is shown in Figure 5.

We now discuss arguments related to coordinate projection and adding geographic maps to an image. Longitude and latitude coordinates can be projected before plotting by specifying the `proj`, `parameters`, and `orientation` arguments. When specified, the coordinates are projected using the `mapproject` function in the **mapproj** package. `proj` specifies the name of the projection to utilize (the default is `'none'`, i.e. no projection). The `parameters` argument specifies the parameter values of the chosen projection, and `orientation` can be used to change the orientation of the projection. See the `mapproject` function in the **mapproj** function for more details regarding these arguments. Several geographic maps can be automatically added to the image by specifying the `map` argument. The available geographic maps come from the **maps** package (Brownrigg et al., 2016), and include the `world`, `usa`, `state`, `county`, `france`, `nz` (New Zealand), `italy`, `lakes`, and `world2` maps. Interested users

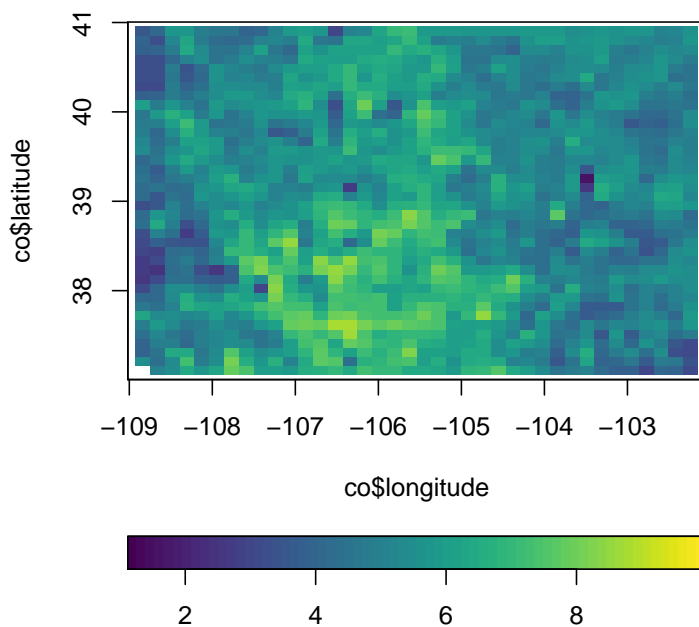


Figure 4: A heat map of the Aluminum measurements for the co data set.

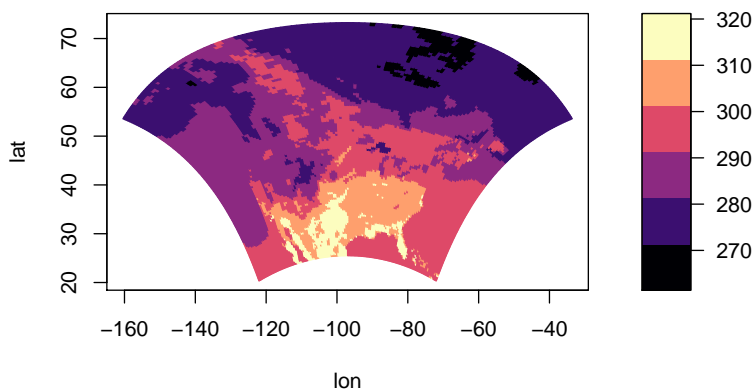


Figure 5: A heat map of the tasmax measurements for the first day of the narccap data using a custom color scheme and vertical color scale.

can find more details about these maps in the **maps** package. We now create a heat map with projected coordinates for the first day of `narccap` `tasmax` measurements. We utilize the Bonne projection using 45 degrees as the standard parallel. We also add a geographic map of the continental United States (U.S.). Note that a grid is automatically added to the image because latitude and longitude parallels are not straight for most projections. The command

```
pimage(lon, lat, tasmax[, , 1], proj = 'bonne', parameters = 45, map = 'usa')
```

produces the heat map in Figure 6.

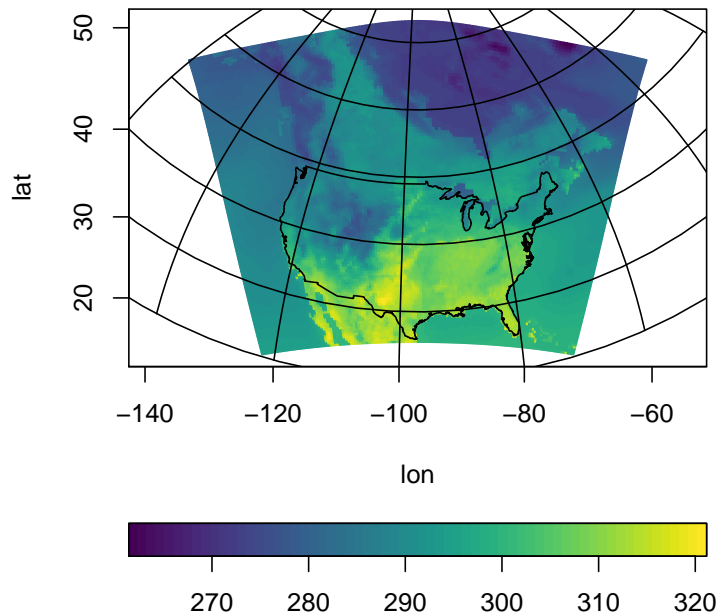


Figure 6: A heat map of the `tasmax` measurements for the first day of the `narccap` data using projected coordinates with an added geographic map of the continental U.S.

The last major argument to the `pimage` function is the `lratio` argument. This argument controls the relative height or width of the color scale in comparison with the main plotting area. Increasing `lratio` increases the thickness of the color scale, while decreasing `lratio` decreases the thickness of the color scale.

Additional customizations to heat maps produced by the `pimage` function can be made via the `ellipses(...)` argument. This includes adding custom geographic maps and points to the image, customizing the grid lines created when the coordinates are projected, further customizing the appearance of the color scale, and customizing the axis labels. These customizations are discussed in detail in the vignette included in the **autoimage** package, which can be accessed by executing the command `vignette('autoimage')`.

Basic usage of the `autoimage` function

We next discuss the `autoimage` function, which generalizes the `pimage` function to create a heat map with a sequence of images. The arguments for the `autoimage` and `pimage` functions are mostly the same, and we replicate their discussion only when necessary.

The structure of the `z` argument for the `autoimage` function may vary slightly from the `pimage` function. Specifically, if multiple gridded images are to be constructed, then `z` will be a three-dimensional array instead of a matrix. Each element of the third dimension of `z` corresponds to the matrix of gridded values for each image. If images for multiple non-gridded variables are to be constructed, then `z` will be a matrix where each column corresponds to a different variable. The `autoimage` function automatically constructs a sequence of images with a common color scale. If individual color scales are desired for each image, then the `common.legend` argument can be set to `FALSE`. The `size` argument can be used to specify the layout of the sequence of images, similar to the `mfrac` argument of the

par function in the **graphics** package. If this is not specified, the autosize function in the **autoimage** package is used to automatically choose the layout, with a tendency to produce a layout closer to square dimensions. We produce a 1×3 matrix of images with a common color scale for the narccap data using the command

```
autoimage(lon, lat, tasmax[,1:3], size = c(1, 3))
```

as shown in Figure 7. Similarly, we create a 2×2 layout of images for the Colorado geochemical

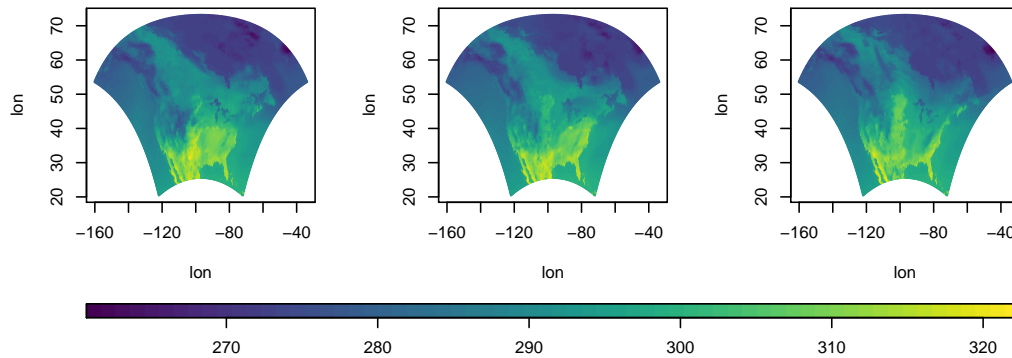


Figure 7: A heat map of the narccap data set tasmax measurements in a 1×3 layout.

measurements by passing 4 columns of the co data frame to the autoimage function. The values of each variable have substantially different ranges, so we use individual color scales for each image. Titles are added to each image using the main argument by providing a character vector whose length matches the number of plotted images. Executing the command

```
autoimage(co$lon, co$lati, co[,c('Al', 'Ca', 'Fe', 'K')], common.legend = FALSE,
          main = c('(a) Aluminum %', '(b) Calcium %', '(c) Iron %', '(d) Potassium %'))
```

produces Figure 8.

It is natural to add a common title to a heat map with multiple images. This can be accomplished by passing the desired title to the outer.title argument. The title will be added in the outer margin of the resulting heat map. The size of the outer margins should be specified using the oma argument of the par function of the **graphics** package before adding the common title. However, a sensible choice for the size of the outer margins is automatically made by the autoimage package if this is not specified by the user. We create the heat map of narccap tasmax measurements shown in Figure 9 by executing the command

```
autoimage(lon, lat, tasmax, outer.title = 'tasmax for 5 days')
```

Additional customizations to heat maps produced by the autoimage function can be made via the ellipses (...) argument. In addition to the customizations mentioned in the context of the pimage function, this includes the ability to change the appearance of the common title. These customizations are discussed in detail in the vignette included in the **autoimage** package, which can be accessed by executing the command vignette('autoimage').

Richer plots using the autolayout and autolegend functions

Suppose we want to add custom features to a sequences of images, with each image receiving different features. One can create a richer sequence of images using the autolayout and autolegend functions.

The autolayout function partitions the graphic device into the sections needed to create a sequence of images. The most important function arguments include size, legend, common.legend, and lratio, which correspond to the same arguments in the pimage and autoimage functions. The outer argument specifies whether a common title will be utilized in the plot. The default is outer = FALSE, indicating that no common title will be added to the plot. When autolayout is called, numbers identify the plotting order of the sections, though these can be hidden by setting show = FALSE. As an initial example, we create a 2×3 matrix of images with a common vertical color scale. The command

```
autolayout(c(2, 3), legend = 'v')
```

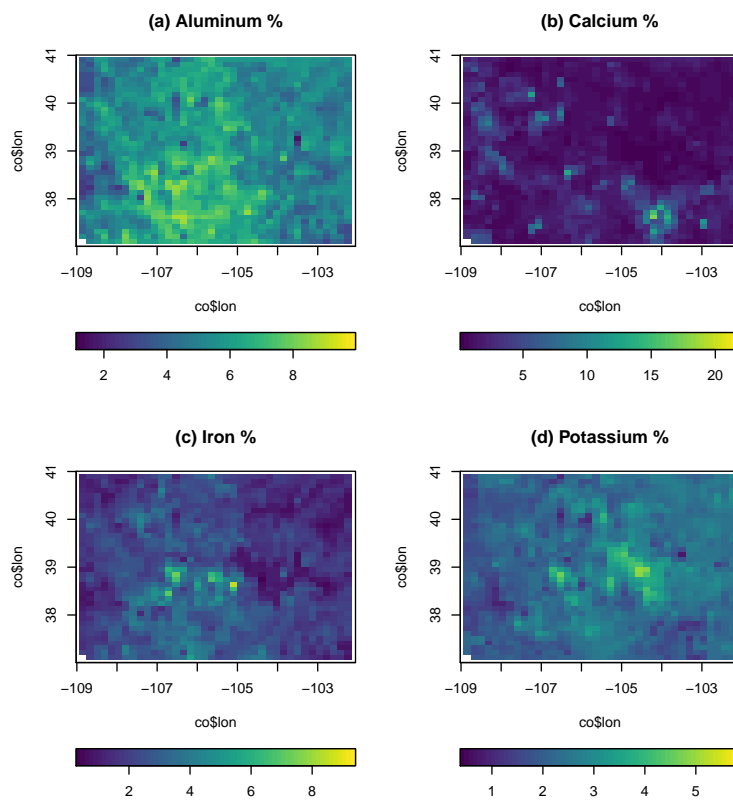


Figure 8: A heat map of various geochemical measurements taken in Colorado.

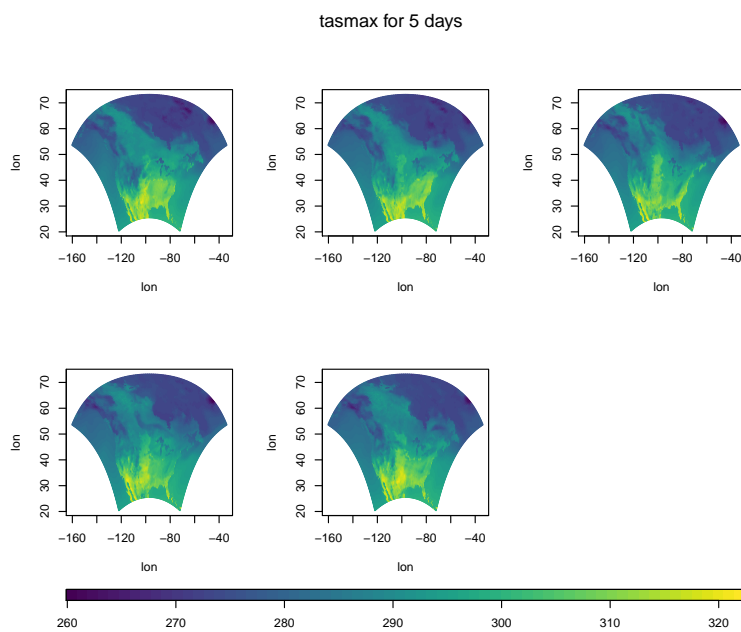


Figure 9: A heat map of the NARCCAP tasmax measurements with a common title.

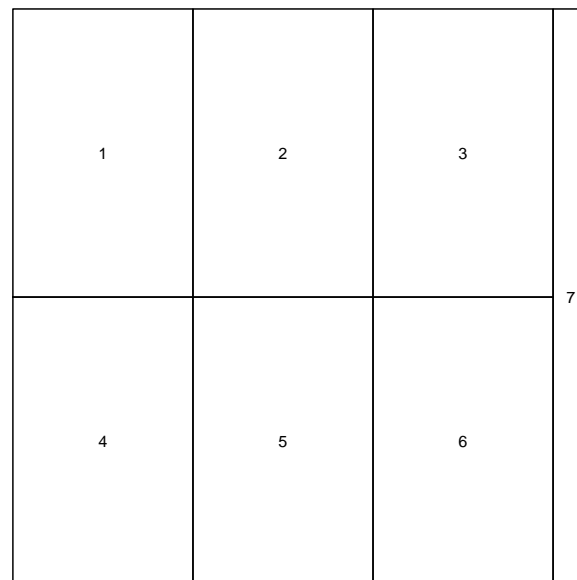


Figure 10: A 2×3 matrix of plotting regions with a common vertical legend created by the `autolayout` function.

produces the plot in Figure 10.

After creating the layout of a complicated heat map using the `autolayout` function, the images should be created using the `pimage` function while specifying `legend = 'none'`. After the desired image or set of images is created, one can automatically add the appropriate legend by calling `autolegend`. The `autolegend` function recovers relevant color scale parameters from the most recent `pimage` call. Consequently, if a common legend is desired, it is important to specify a common `zlim` argument among all relevant `pimage` calls.

Various features can be added to the images using the `ppoints`, `plines`, `ptext`, `psegments`, `parrows`, and `ppolygon` functions. These are analogues of the `points`, `lines`, `text`, `segments`, `arrows`, and `polygon` functions in the **graphics** package, to be used with images containing projected coordinates.

We provide an (unrealistic, but informative) example of the kinds of plots these functions can be used to create. Suppose we wish to create a heat map for both the `narccap` and Colorado geochemical data sets. We desire to add a geographic map for each state in the U.S. for the `narccap` data while using the Mercator projection to transform the coordinates before plotting. We desire to plot the Aluminum measurements for the Colorado data. We want a custom color scale similar to the jet color scheme in Matlab. We desire to project the coordinates before plotting using the Bonne projection with a reference latitude of 39 degrees. We also desire to add geographic maps of the relevant U.S. counties to the map, along with labels for select Colorado cities. Lastly, we will add a large, purple title to the two heat maps.

We begin by obtaining some of the relevant information for these plots. The borders for the continental U.S. states is available in the state map in the **maps** package. However, we must extract the borders of Alaska and Hawaii from the world map in the **maps** package. Executing the commands

```
data(worldMapEnv, package = 'maps')
hiak <- maps::map('world', c('USA:Hawaii', 'USA:Alaska'), plot = FALSE)
```

loads and extracts the borders for Hawaii and Alaska. The results are stored in a list named `hiak` with named vectors `x` and `y` containing the coordinates of the borders. Each state border is separated by an NA value. The commands

```
data(us.cities, package = 'maps')
codf <- us.cities[us.cities$country.etc == 'CO', ]
codf <- codf[c(3, 5, 7:10, 18), ]
```

loads the `us.cities` data set from the **maps** package, extracts the Colorado cities from this data set, then stores a small sample of these Colorado cities in the `codf` data frame. Information related to the name, longitude, and latitude of each city is included in the data frame.

Having obtained the relevant information, we setup a 1×2 matrix of images with individual horizontal color scales and an area for a common title using the command:

```
autolayout(c(1, 2), legend = "h", common.legend = FALSE, outer = TRUE)
```

We now create a heat map for the first day of NARCCAP tasmax measurements using the command

```
pimage(lon, lat, tasmax[,1], legend = 'none', proj = 'mercator',
       map = 'state', lines.args = list(col = 'grey'))
```

Note that `legend = 'none'` since this will be added afterward using the `autolegend` function. The Mercator projection was chosen via the `proj` argument, and the state geographic map was added. The color of the geographic map was changed by passing the `lines.args` argument to `pimage`. The `lines.args` argument is a named list with components matching the arguments of the `lines` function in the **graphics** package, and changes the appearance of any geographic maps plotted using the `pimage` function. We next add the state borders for Hawaii and Alaska using the `plines` function, title the heat map, and add the color scale to the image by executing the following commands:

```
plines(hiak, proj = 'mercator', col = 'grey')
title('tasmax for North America')
autolegend()
```

Note the specification of the projection and line color in the `plines` function call.

Next, we construct the image for the Colorado Aluminum measurements using the commands

```
pimage(co$lon, co$lat, co$Al, map = 'county', legend = 'none',
       proj = 'bonne', parameters = 39, paxes.args = list(grid = FALSE),
       col = fields::tim.colors(64), lines.args = list(col = 'grey'))
```

Note the custom color scheme specified by the `col` argument. Also, the `paxes.args` argument is passed to the `pimage` function to suppress the grid lines that would normally be plotted when a projection is used. The locations and names of the Colorado cities are added to the image using the commands

```
ppoints(codf$lon, codf$lat, pch = 16, proj = 'bonne')
ptext(codf$lon, codf$lat, labels = codf$name, proj = 'bonne', pos = 4)
```

A title and color scale are added to the image using the commands

```
title('Colorado Aluminum levels (%)')
autolegend()
```

Lastly, we specify a large common title for the heat maps using the `mtext` function in the **graphics** package. Specifically, we execute the command:

```
mtext('Two complicated maps', col = 'purple', outer = TRUE, cex = 2)
```

The plot resulting from the commands listed in this section is shown in Figure 11.

Two complicated maps

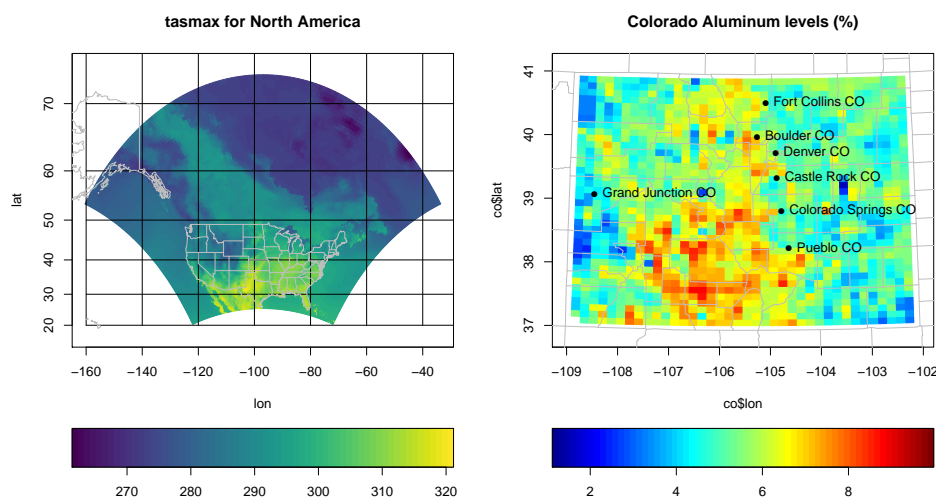


Figure 11: A complicated set of heat maps created using the `autolayout` and `autolegend` functions.

Summary

We have introduced the **autoimage** package for constructing heat maps. The **autoimage** package natively supports: 1. automatic inclusion of a color scale with the plotted image, 2. construction of heat maps for responses observed on regular or irregular grids, as well as non-gridded data, 3. construction of a matrix of heat maps with a common color scale, 4. construction of a matrix of heat maps with individual color scales, 5. projecting coordinates before plotting, 6. easily adding geographic borders, points, and other features to the heat maps. We believe that the breadth of native features, simplicity, and speed of the **autoimage** package make it a unique contribution to the R user community in comparison with other R tools that can be used for creating heat maps. We note however that the **autoimage** package is a specialized tool and is limited to the construction of heat maps. This is in contrast to other plotting packages mentioned earlier, such as the **lattice**, **sp**, and **ggplot2** packages, which have many plotting capabilities unrelated to heat maps.

A number of additional features and heat map customizations are available in the **autoimage** package, but were not discussed in order to shorten the length of this article. Additional discussion and examples may be found in the documentation of the **autoimage** package. A more detailed introduction to the capabilities of the **autoimage** package can be accessed by executing the command `vignette('autoimage')`. Additionally, a vignette comparing some of the heat map capabilities of the **autoimage** and **ggplot2** packages can be accessed by executing the command `vignette('ggplot2-comparison')`.

The source code for the **autoimage** package is publicly hosted at <https://github.com/jpfrench81/autoimage>. Bugs and other issues may be reported there. Additionally, interested parties are encouraged to submit improvements for the **autoimage** package at the GitHub repository listed above.

Acknowledgments

This **autoimage** package was created for research supported by NSF Grant DMS-1463642 and NIH Grant R01 CA157528. This package would not have been created without inspiration from the internals of the `image.plot` function in the **fields** package written by Doug Nychka and from the `image.scale.2` function written by Marc Taylor and discussed at <http://menugget.blogspot.com/2013/12/new-version-of-imagescale-function.html>.

Bibliography

- H. Akima and A. Gebhardt. *akima: Interpolation of Irregularly and Regularly Spaced Data*, 2016. URL <https://CRAN.R-project.org/package=akima>. R package version 0.6-2. [p287]
- B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2016. URL <https://CRAN.R-project.org/package=gridExtra>. R package version 2.2.1. [p287]
- A. Baddeley and R. Turner. *spatstat: An R package for analyzing spatial point patterns*. *Journal of Statistical Software*, 12(6):1–42, 2005. URL <https://doi.org/10.18637/jss.v012.i06>. [p286]
- R. S. Bivand, E. Pebesma, and V. Gómez-Rubio. *Applied spatial data analysis with R, Second edition*. Springer, NY, 2015. URL <https://doi.org/10.1007/978-1-4614-7618-4>. asdar-book.org. [p286]
- R. Brownrigg, T. P. Minka, and A. Deckmyn. *maps: Draw Geographical Maps*, 2016. URL <https://CRAN.R-project.org/package=maps>. R package version 3.1.1. [p289]
- D. Caya and R. Laprise. A semi-implicit semi-Lagrangian regional climate model: The Canadian RCM. *Monthly Weather Review*, 127(3):341–362, 1999. URL [https://doi.org/10.1175/1520-0493\(1999\)127<0341:ASISLR>2.0.CO;2](https://doi.org/10.1175/1520-0493(1999)127<0341:ASISLR>2.0.CO;2). [p284]
- M. Collins, B. B. Booth, G. R. Harris, J. M. Murphy, D. M. Sexton, and M. J. Webb. Towards quantifying uncertainty in transient climate change. *Climate Dynamics*, 27(2-3):127–147, 2006. URL <https://doi.org/10.1007/s00382-006-0121-0>. [p284]
- Douglas Nychka, Reinhard Furrer, John Paige, and Stephan Sain. *fields: Tools for spatial data*, 2015. URL <https://doi.org/10.5065/D6W957CT>. R package version 8.10. [p284]
- J. French. *gear: Geostatistical Analysis in R*, 2015. URL <https://CRAN.R-project.org/package=gear>. R package version 0.1.1. [p288]

- J. French. *autoimage: Multiple Heat Maps for Projected Coordinates*, 2017. URL <https://CRAN.R-project.org/package=autoimage>. R package version 1.3. [p284]
- S. Garnier. *viridisLite: Default Color Maps from 'matplotlib' (Lite Version)*, 2016. URL <https://CRAN.R-project.org/package=viridisLite>. R package version 0.1.3. [p289]
- D. McIlroy. *mapproj: Map Projections*, 2015. URL <https://CRAN.R-project.org/package=mapproj>. R package version 1.2-4. Packaged for R by Ray Brownrigg and Thomas P Minka and transition to Plan 9 codebase by Roger Bivand. [p287]
- L. O. Mearns and others. The North American Regional Climate Change Assessment Program dataset, National Center for Atmospheric Research Earth System Grid data portal, 2007, updated 2014. URL <https://doi.org/10.5065/D6RN35ST>. Boulder, CO. Data downloaded 2016-08-12. [p284]
- L. O. Mearns, W. Gutowski, R. Jones, R. Leung, S. McGinnis, A. Nunes, and Y. Qian. A regional climate change assessment program for North America. *EOS, Transactions American Geophysical Union*, 90(36):311–311, 2009. URL <https://doi.org/10.1029/2009E0360002>. [p284]
- L. O. Mearns, R. Arritt, S. Biner, M. S. Bukovsky, S. McGinnis, S. Sain, D. Caya, J. Correia Jr, D. Flory, W. Gutowski, and others. The North American regional climate change assessment program: Overview of phase i results. *Bulletin of the American Meteorological Society*, 93(9):1337–1362, 2012. URL <https://doi.org/10.1175/BAMS-D-11-00223.1>. [p284]
- E. J. Pebesma and R. S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2), 2005. URL <http://cran.r-project.org/doc/Rnews/>. [p286]
- D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York, 2008. URL <https://doi.org/10.1007/978-0-387-75969-2>. ISBN 978-0-387-75968-5. [p284]
- D. B. Smith and K. J. Ellefsen. Soil geochemical data for the Wyoming landscape conservation initiative study area: Data series 510. Technical report, U.S. Geological Survey, 2010. <http://pubs.usgs.gov/ds/510/>. [p288]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. URL <https://doi.org/10.1007/978-0-387-98141-3>. <http://ggplot2.org>. [p286]
- C. O. Wilke. *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'*, 2016. URL <https://CRAN.R-project.org/package=cowplot>. R package version 0.7.0. [p287]

Joshua P. French
University of Colorado Denver
Campus Box 170, PO Box 173364, Denver, CO 80217
USA
joshua.french@ucdenver.edu

Market Area Analysis for Retail and Service Locations with MCI

by Thomas Wieland

Abstract In retail location analysis, marketing research and spatial planning, the market areas of stores and/or locations are a frequent subject. Market area analyses consist of empirical observations and modeling via theoretical and/or econometric models such as the Huff Model or the Multiplicative Competitive Interaction Model. The authors' package MCI implements the steps of market area analysis into R with a focus on fitting the models and data preparation and processing.

Introduction

A *market area* (also called *trading area*, *service area* or *catchment area*) is a part of the earth's surface where the actual or potential customers of a supply location come from. This *supply location* can be any kind of location which provides goods and/or services and generates a geographically segmented market. In retailing, a supply location can be a single store, a retail agglomeration (planned or unplanned shopping centre) or even an entire city (Berman and Evans, 2013; Löffler, 1998).

The market area of a store/location can be regarded as the spatial equivalent to its sum of all customers and/or sales. The total customers/sales of a supply location can be determined by summing the customer flows/expenditures from each geographic region in its market area (Huff and McCallum, 2008; Rodrigue et al., 2006). For practical reasons, a market area can also be divided by zones of market penetration and/or distance/travel time (Berman and Evans, 2013).

Market areas of retail locations result from the consumer spatial shopping behaviour, more precisely, their *store choice*. Thus, a market area is influenced by many factors such as the *transport costs* (e.g. distance, travel time) between customers and locations and, of course, the characteristics of the competitors (e.g. perceived "attraction", pricing, image or the opportunity for multi-purpose and comparison shopping). Mostly, the market areas of competing supply locations overlap, which means that they are in *spatial competition* (Rodrigue et al., 2006; Wieland, 2015a).

Traditionally, *market area analysis* includes the delineation and segmentation of market areas and can be divided into *inductive-empirical* and *deductive-theoretical* approaches. The first type consists of constructing market areas based on empirical observations such as point-of-sale (POS) surveys (*customer spotting*), while in the latter approach this work is done by using mathematical *market area models* (Löffler, 1998). Modern market area analyses are mostly a combination of both, especially when using econometric market area models which are fitted by empirical data (Wieland, 2015a).

Market area models can be used in retail location analysis to find new locations, to evaluate the existing outlets or to assess the impact of changes in the competitive landscape (Berman and Evans, 2013; Huff and McCallum, 2008). Market area analyses are also subject of governmental spatial planning in Germany (Wolf, 2012). The econometric models can also be utilized to identify variables influencing consumer decisions and market areas and to check hypotheses about these relations, which means to find out what affects store choice (Wieland, 2015a). The usage of market area models can also be transferred to other service locations, such as health services (Jia et al., 2015).

This paper presents two market area models, the *Huff Model* and the *Multiplicative Competitive Interaction (MCI) Model*, and their implementation into R by the authors' package **MCI** (Wieland, 2016). Yet, only the basic Huff Model formula is integrated in R by the packages **SpatialPosition** (Giraud and Commenges, 2016), which can be used especially for graphical visualization, and **huff-tools** (Pavlis et al., 2014), which combines the basic formulation with GIS functions. In contrast, the emphases of the **MCI** package lie on 1) fitting the MCI Model and the Huff Model via OLS (ordinary least squares) regression and nonlinear techniques, respectively, and 2) the steps of processing and transforming empirical data to be usable in the models (especially working with *interaction matrices*).

Market area models: theory and application

Almost all market area models have in common that they are based on an attraction/utility function including transport costs and further characteristics of the supply locations which are subjects of a trade-off by the consumers. The dependent variable of these models is the store choice and/or the choice probability and/or the market shares of the stores/locations (Wieland, 2015a).

The first retail market area models (Reilly, 1929; Converse, 1949) were *deterministic*, which means that each customer origin is assigned completely to one supply location while overlapping market

areas are not envisaged. Furthermore only two supply locations can be processed and the explaining variables of the models are not founded theoretically. These gaps were filled by Huff (1962, 1963, 1964) by introducing his *probabilistic* market area model. An important content enhancement and, at the same time, an econometric transformation of the Huff Model was introduced in the form of the Multiplicative Competitive Interaction Model by Nakanishi and Cooper (1974, 1982).

The Huff Model

Theoretical background and formulation

The basis of the Huff Model is the following multiplicative utility function with two explanatory variables representing two determinants of store choice (Huff and Batsell, 1975):

$$U_{ij} = A_j^\gamma d_{ij}^{-\lambda}, \quad (1)$$

where U_{ij} is the utility of the supply location j for the customers at origin i , A_j reflects the attraction of supply location j and d_{ij} contains the transport costs customers from i have to take to reach j . The exponents γ and λ are weighting parameters.

The attraction is translated as the size of the location due to the increasing probability for a "successful" shopping trip on condition of consumer uncertainty (the greater the locations' offer, the more likely is to get the desired goods). The size is operationalized by the sales area of the locations. But, as the consumers' decision costs normally rise with an increasing number of offered goods, the marginal utility of the locations' offer decreases which is reflected by a degressive weighting of size ($0 < \gamma < 1$). The indicator for the transport costs is the travel time from i to j , reflecting the time consumed by a shopping trip. To integrate the opportunity costs and the perceived disutility of traveling, the travel time is weighted negatively and progressively ($|\lambda| > 1$) (Huff, 1962).

The parameter λ also reflects the range of the offered goods dependent on the shopping frequency: the more high-order the good/less frequently purchased, the less is the disutility of transport costs (Güssefeldt, 2002). But this *distance decay function* of the power type can also be replaced by an exponential or a logistic function (Kanhäußer, 2007). As suggested by Huff (1962), also the attraction function can be logistic to better reflect the effect of decreasing marginal utility of size.

Derived from the behavioral scientific *Luce choice axiom* (Luce, 1959), the consumer decision in the Huff Model is regarded as probabilistic. The probability to choose the alternative j from a set of alternatives ($j = 1, \dots, n$) is the quotient of its utility U_{ij} and the sum of the utilities of all alternatives (Huff, 1962):

$$p_{ij} = \frac{U_{ij}}{\sum_{j=1}^n U_{ij}} = \frac{A_j^\gamma d_{ij}^{-\lambda}}{\sum_{j=1}^n A_j^\gamma d_{ij}^{-\lambda}}, \quad (2)$$

where p_{ij} is the probability that the customers from origin i travel to location j , what can be called *interaction probability*, where: $\sum_{j=1}^n p_{ij} = 1$ and $0 < p_{ij} < 1$. These probabilities can be interpreted as market shares of location j in origin i , what can be called *local market shares*. These shares implicitly represent a final state of consumer preference patterns in a spatial equilibrium (Huff and Batsell, 1975). Thus, in the Huff Model, the revenues of a retail store/location j depend on its own attraction, the attraction of all competitors and the transport costs between all locations and the customer origins.

The expected customer/expenditure flows from i to j are estimated by multiplying the local market shares with the local market potential (Huff, 1962):

$$E_{ij} = p_{ij} C_i, \quad (3)$$

where E_{ij} is the number of expected customer/purchasing power flows from origin i to location j and C_i is the total market potential (number of potential customers or purchasing power) in i .

The complete market area of location j is the sum of all regional customer or purchasing power flows, while the former represents the total number of customers and the latter equals the total sales of the location, e.g. within a year (Huff, 1964):

$$T_j = \sum_{i=1}^m E_{ij}, \quad (4)$$

where T_j is the market area of j containing m submarkets, normally measured in persons or money.

i	j	d_{ij}	A_j	U_{ij}	$\sum U_{ij}$	p_{ij}	C_i	E_{ij}
i_1	j_1	$d_{i_1j_1}$	A_{j_1}	$U_{i_1j_1}$	$\sum U_{i_1}$	$p_{i_1j_1}$	C_{i_1}	$E_{i_1j_1}$
i_1	j_2	$d_{i_1j_2}$	A_{j_2}	$U_{i_1j_2}$	$\sum U_{i_1}$	$p_{i_1j_2}$	C_{i_1}	$E_{i_1j_2}$
i_1	j_3	$d_{i_1j_3}$	A_{j_3}	$U_{i_1j_3}$	$\sum U_{i_1}$	$p_{i_1j_3}$	C_{i_1}	$E_{i_1j_3}$
i_2	j_1	$d_{i_2j_1}$	A_{j_1}	$U_{i_2j_1}$	$\sum U_{i_2}$	$p_{i_2j_1}$	C_{i_2}	$E_{i_2j_1}$
i_2	j_2	$d_{i_2j_2}$	A_{j_2}	$U_{i_2j_2}$	$\sum U_{i_2}$	$p_{i_2j_2}$	C_{i_2}	$E_{i_2j_2}$
...
i_m	j_n	$d_{i_mj_n}$	A_{j_n}	$U_{i_mj_n}$	$\sum U_{i_m}$	$p_{i_mj_n}$	C_{i_m}	$E_{i_mj_n}$

Table 1: Huff interaction matrix (schematic).

Empirical usage

While working with the Huff Model, the first step is to delineate the study area itself and to record the focused origins and locations. The explanatory variables must be observed empirically by mapping the regarded stores/locations (A_j) and calculating travel times (d_{ij}), respectively (Huff, 1962).

All data is stored into a special kind of linear table that is called *interaction matrix* and is the basis for all further calculation steps (see Table 1). The interaction matrix contains one row for each (potential) spatial interaction between the origins, i (where $i = 1, \dots, m$), and the locations, j (where $j = 1, \dots, n$), so the number of rows sums up to $m * n$. For every row, the utilities and probabilities are calculated using the formulas mentioned above (Wieland, 2015a).

Many approaches were developed to estimate the weighting coefficients by nonlinear iterative techniques. Huff (1962) showed a way to estimate λ with a fitting algorithm comparing expected and observed market shares by correlation coefficients while keeping γ constantly equal to one. But, if p_{ij} , the dependent variable in the Huff Model, was observed, probably the best way to estimate the parameters is the econometric transformation of the model which was achieved by the Multiplicative Competitive Interaction Model by Nakanishi and Cooper (1974) (see the next section).

In many cases, no empirical market shares are available but only observed total sales of stores or locations, T_j . Thus, many studies focus the model fitting by nonlinear optimization algorithms based on empirical total values (Baecker-Neuchl and Wagenseil, 2015; De Beule et al., 2014; Güssefeldt, 2002; Marinov and Czamanski, 2012; Orpana and Lampinen, 2003; Klein, 1988; Yingru and Lin, 2012). In the following, a new optimization algorithm of this type is introduced which is based on the idea of the *local optimization of attraction* approach by Güssefeldt (2002) (which is the only one with an explicit theoretical fundament) and features of other mentioned procedures.

A new optimization algorithm for the Huff Model

Consider a location system with j supply locations (where $j = 1, \dots, n$), where $T_{j_{obs}}$ defines the real (observed) annual sales of j . Güssefeldt (2002) argues that every store or location has its own unknown revenue function and the sales (=revenues) depend on its own attraction (=input) and each competitors' behaviour. The attraction variable in the Huff Model, A_j , is a proxy variable for size (e.g. sales area) reflecting the extent of offers which is the input in the unknown revenue function. But this non-adjusted attraction measure does not reflect the "real" attraction because every competitor modifies the input in his revenue function (e.g. via marketing efforts) without any change in the size variable.

Thus, the expected sales using non-adjusted attractions in the Huff Model, $T_{j_{exp}}$, do not correspond to the observed sales what can be measured on the *local* level (which means a single store/location) by the *absolute percentage error*:

$$APE_j = \frac{|T_{j_{exp}} - T_{j_{obs}}|}{T_{j_{obs}}}, \tag{5}$$

where $T_{j_{exp}}$ is the expected total sales of supply location j (Huff Model result), $T_{j_{obs}}$ is the observed total sales and APE_j is the absolute percentage error for supply location j .

Since the algorithm by Güssefeldt (2002) addresses the local level, there is no fit measure for the *global* level (which means the model fit for the *complete* location system). The global model fit can be evaluated in different ways, where three fit measures were already used in the context of optimizing the Huff Model: the *mean absolute percentage error (MAPE)* and a *PseudoR²* measure, both used by

De Beule et al. (2014), and the *global error* (GE) used by Klein (1988):

$$MAPE = \frac{1}{N} \sum_{n=1}^N \frac{|T_{j_{exp}} - T_{j_{obs}}|}{T_{j_{obs}}}, \quad (6)$$

where $T_{j_{exp}}$ is the expected total sales of location j (Huff Model result), $T_{j_{obs}}$ is the observed total sales and N is the number of objects.

$$PseudoR^2 = \frac{var(T_{j_{obs}}) - var(\epsilon_j)}{var(T_{j_{obs}})}, \quad (7)$$

where $T_{j_{obs}}$ is the observed total sales and ϵ_j is the residuum, $T_{j_{obs}} - T_{j_{exp}}$.

$$GE = \frac{\sum |\epsilon_j|}{\sum T_{j_{obs}}}, \quad (8)$$

where $T_{j_{obs}}$ is the observed total sales and ϵ_j is the residuum, $T_{j_{obs}} - T_{j_{exp}}$.

Since the real sales are known and on condition that the location attraction and the sales are related to each other, the attraction A_j can be described as a function of the sales (Güssefeldt, 2002):

$$A_j = a + bT_j, \quad (9)$$

where A_j is the predicted attraction of location j , T_j is the total sales of j , a is the intercept and b is the slope of the attraction function. As every supply location has its own revenue function (each competitor has an individual factor use), the attraction function is also different for each store/location.

On condition that an interval is known, this function can be parametrized by calculating the slope using the difference quotient and the intercept, thereafter (Güssefeldt, 2002). Unlike in the mentioned approach, this function *must* pass the origin because in the Huff Model with its multiplicative utility function (Formula 1), an attraction equal to zero ($A_j = 0$) must result in an utility equal to zero ($U_{ij} = 0$) which results in local market shares equal to zero ($p_{ij} = 0$, Formula 2). Thus, the total sales (Formulae 3 and 4) must be equal to zero ($T_j = 0$), too. This is a logical consequence from the theoretical basement of the Huff Model but is also part of the principle of *logical consistency of market shares* in marketing research (Cooper and Nakanishi, 2010).

Thus, formula 9 has no intercept ($a = 0$) which means a directly proportional relation between sales and attraction. But, as the revenue function differs by each supply location, the attraction function must also be parametrized for each store/location. The slope in the attraction function of location j can be calculated via:

$$b_j = \frac{A_j - A_{j_0}}{T_{j_{exp}} - T_{j_0}} = \frac{A_j}{T_{j_{exp}}}, \quad (10)$$

where b_j is the slope of the attraction function for j , A_j is the non-adjusted attraction of location j (such as sales area), $T_{j_{exp}}$ is the total expected sales of j predicted by the Huff Model, A_{j_0} is the attraction when the sales are equal to zero and T_{j_0} represents the sales when the attraction is equal to zero, both equal to zero ($A_j = 0 \Leftrightarrow T_j = 0$). The adjusted attraction of supply location j can be calculated via:

$$A_{j_{adj}} = b_j T_{j_{obs}}, \quad (11)$$

where $A_{j_{adj}}$ is the adjusted attraction of j , b_j is the slope of the attraction function for j and $T_{j_{obs}}$ represents the real (observed) sales of j .

The relations mentioned above can be brought together in an optimization algorithm for the Huff Model with respect to a location system with j locations ($j = 1, \dots, n$) containing the following 8 steps:

1. Set a tolerance value, tol_{APE} , to define which difference between the real and the expected sales of location j is accepted, e.g. $tol_{APE} = 5$, which means an accepted deviation of +/- 5 percent. Define a transport costs weighting function (power, exponential or logistic) and the weighting parameter(s) λ for formula 2.
2. Calculate the market areas for the location system and the total sales of the n locations using formulae 2 to 4 with $\gamma = 1$ and the transport costs weighting as defined in step 1.
3. Calculate the absolute percentage error between the expected and the observed total sales of location j (APE_j) by formula 5. If the error APE_j is smaller than the tolerance tol_{APE} , no further local optimization for location j is needed, so you can repeat step 3 with location $j + 1$. If $APE_j > tol_{APE}$, go to step 4.
4. Calculate the slope of the attraction function by formula 10. Calculate the adjusted attraction via formula 11.

5. Save the adjusted attraction of location j , $A_{j,adj}$, in the actual Huff interaction matrix and repeat the procedure beginning at step 2 with the next location, $j + 1$.
6. Repeat steps 2 to 5 for all locations ($j = 1, \dots, n$).
7. After the last location $j = n$ was processed, calculate the global fit measures for the complete location system by formulae 6 to 8.
8. Repeat steps 2 to 7 for the complete location system until the local optima and/or the global optimum is sufficiently approximated. The former can be evaluated by the tolerance value (step 3) for every j location, while the latter can be controlled by the global fit measures (step 7).

The Multiplicative Competitive Interaction (MCI) Model

Theoretical background and formulation

The Multiplicative Competitive Interaction Model (in short: *MCI Model*) is based on the Huff Model but also belongs to the model family of *market share models* which were developed in marketing science. Thus, it can be regarded as a crossover of these two model families (Cliquet, 2013). The fundamental theorem behind market share models is the following simple relationship between the competitors' characteristics and their market shares (Cooper and Nakanishi, 2010):

$$MS_j = \frac{A_j}{\sum_{j=1}^n A_j}, \quad (12)$$

where MS_j is the market share of competitor j and A_j is the attraction of j . This leads to two characteristics of market shares which are summarized as *logical-consistency requirements* for market shares: $0 < MS_j < 1$, and $\sum_{j=1}^n MS_j = 1$, respectively (Cooper and Nakanishi, 2010). This market share logic is obviously related to the probabilistic concept of the Huff Model when the term "market shares" is replaced by "choice probabilities", "interaction probabilities" or "local market shares" and the construct "attraction" is replaced by the construct "utility" (Wieland, 2015a).

Derived from the Huff Model, the MCI Model is explicitly formulated to regard a market which is segmented into i submarkets ($i = 1, \dots, m$) and which is served by j suppliers ($j = 1, \dots, n$). The attraction function is multiplicative and consists of h ($h = 1, \dots, H$) explanatory variables which are weighted exponentially to reflect their sensitivity (Nakanishi and Cooper, 1974):

$$p_{ij} = \frac{\prod_{h=1}^H A_{hj}^{\gamma_h}}{\sum_{j=1}^n \prod_{h=1}^H A_{hj}^{\gamma_h}}, \quad (13)$$

where p_{ij} is the probability that the customers from submarket i choose supplier j , A_{hj} is the value of the h -th variable describing the object j , γ_h is the weighting parameter for the sensitivity of p_{ij} with respect to the variable h . The next steps (customer or expenditure flows, total market area) can be taken analogously to the Huff Model (Formulae 3 and 4).

The market can be subdivided in any kind of submarkets (e.g. customer groups, time periods, geographic areas). When the market is segmented geographically, it is a matter of a *Spatial MCI Model*, especially when integrating transport costs as an explanatory variable (Cliquet, 2013; Huff and McCallum, 2008). The MCI Model can also be regarded as a generalization of the Huff Model, while the Huff Model can be considered as a special case of multiplicative competitive interaction model.

The log-centering transformation

The models mentioned above are deterministic (no random variation) and nonlinear models which cannot be estimated directly by common econometric techniques but by iterative algorithms (see the former section) which do not allow statements about the statistical significance of the explanatory variables and other inference statistics. The main breakthrough of the MCI Model is the transformation of the nonlinear structure into a linear stochastic model which can be estimated via OLS (Ordinary Least Squares) regression (Huff and McCallum, 2008).

This requires a re-arrangement of the model to be linear in parameters which is achieved by a multi-step transformation of the variables using geometric means and logarithms for standardization and linearization called the *log-centering transformation* (Nakanishi and Cooper, 1974):

$$\log\left(\frac{p_{ij}}{\bar{p}_i}\right) = \sum_{h=1}^H \gamma_h \log\left(\frac{A_{hj}}{\bar{A}_{hj}}\right) + \log\left(\frac{\varepsilon_{ij}}{\bar{\varepsilon}_i}\right), \quad (14)$$

where \tilde{p}_i and \tilde{A}_{h_i} are the geometric means of p_{ij} and A_{h_i} , respectively, and $\tilde{\varepsilon}_i$ is the geometric mean of the disturbance term (residuum) ε_{ij} which is added to the original model to be stochastic. The geometric means of p_{ij} and any submarket-related explanatory variable (such as travel time, d_{ij}) are calculated on the submarket level. As in the original model, the transformation does not include an intercept (regression through the origin) to match the logical-consistency requirements for market shares (Nakanishi and Cooper, 1982).

Once the variables are transformed, the model function is linear in its parameters and can be processed as a multiple linear regression model to be estimated by OLS regression (Rawlings et al., 1998). Thus, the model allows to test hypotheses about the influence of store/location characteristics (such as sales area, pricing) and transport costs (such as travel time, distance) on the local market shares in the submarkets (customer origins) by interpreting the regression coefficients and their inference statistics (Nakanishi and Cooper, 1974, 1982; Huff and McCallum, 2008).

After an estimation of the parameters, they can be included as exponents in the original nonlinear model (Formula 13) to be utilized for market share/market area predictions (Nakanishi and Cooper, 1982; Huff and McCallum, 2008). It is also possible to integrate dummy variables reflecting qualitative information (such as brands or store chains) or an intercept (if necessary). Since the former causes problems in the multiplicative attraction/utility function (multiplication by zero) and the latter is contrary to the logical consistency requirement, a different retransformation of the model called the *inverse log-centering transformation* is required (Nakanishi and Cooper, 1982):

$$\hat{y}_{ij} = \sum_{h=1}^H \hat{\gamma}_h \log \left(\frac{A_{h_i}}{A_{h_j}} \right), \tag{15}$$

$$\hat{p}_{ij} = \frac{e^{\hat{y}_{ij}}}{\sum_{j=1}^n e^{\hat{y}_{ij}}}, \tag{16}$$

where \hat{y}_{ij} is the transformed attraction/utility function and \hat{p}_{ij} is the expected response variable, the interaction probabilities/market shares of the supplier j in the submarket i . Thus, in that cases, the variables are processed as they were transformed in formula 14.

Empirical usage

In the first step, a MCI Model analysis requires the formulation of hypotheses and/or research questions addressing the influence of the H explanatory variables on the market shares based on theoretical considerations. The Huff Model can be regarded as a theoretical base since Huff (1962) assumes size and transport costs as explanatory variables which can be tested by the MCI Model (Kubis and Hartmann, 2007; Suárez-Vega et al., 2015). But the number of additional influences tested is nearly unlimited and ranges from further store/location attributes like age or price level (Huff and McCallum, 2008; Tihl and Oruc, 2012) to the surrounding coupling and competition potential (Wieland, 2015a) to consumer-related subjective variables (Cliquet, 2013; González-Benito et al., 2000).

Since the delineation of the study area and the identification of the relevant competing locations have an enormous impact on the results, these definitions should be made corresponding to the *LIFO* (little in from outside) and *LOFI* (little out from inside) principles (Huff and McCallum, 2008). This means that the majority of shopping interactions should take place within the study area.

The supplier characteristics can be obtained by mapping the relevant stores/locations and additional research. The market shares, p_{ij} , cannot be observed directly but have to be calculated based on empirically observed shopping interactions (shopping trips and/or expenditures) which are collected on the individual or household level. In a representative household survey (or, if not possible, a point-of-sale survey), every respondent is asked for the destination(s) of the last shopping trip(s) at the j location(s) and/or the associated expenditures (Huff and McCallum, 2008; Wieland, 2015a).

To calculate the local market shares of shopping trips and/or expenditures, the individual data must be aggregated on the submarket level:

$$p_{ij} = \frac{O_{ij}}{\sum_{j=1}^n O_{ij}}, \tag{17}$$

where p_{ij} is the empirical market share of supplier j in submarket i and O_{ij} equals the observed frequencies/expenditures of the customers in i with respect to supplier j . $\sum_{j=1}^n O_{ij}$ is the empirical equivalent to the total customer/purchasing power potential in i , C_i , in the Huff Model (formula 3). As in the Huff Model, the empirical market shares, p_{ij} , and the observed explanatory variables (A_1, \dots, A_H, d_{ij}) are stored in an interaction matrix (see Table 2).

Mostly, the observed variables cannot be processed directly: the log-centering transformation

i	j	p_{ij}	A_{1j}	A_{2j}	A_{3j}	...	A_{Hj}	d_{ij}
<i>i</i> ₁	<i>j</i> ₁	<i>p</i> _{<i>i</i>₁<i>j</i>₁}	<i>A</i> ₁₁	<i>A</i> ₂₁	<i>A</i> ₃₁	...	<i>A</i> _{<i>H</i>1}	<i>d</i> _{<i>i</i>₁<i>j</i>₁}
<i>i</i> ₁	<i>j</i> ₂	<i>p</i> _{<i>i</i>₁<i>j</i>₂}	<i>A</i> ₁₂	<i>A</i> ₂₂	<i>A</i> ₃₂	...	<i>A</i> _{<i>H</i>2}	<i>d</i> _{<i>i</i>₁<i>j</i>₂}
<i>i</i> ₁	<i>j</i> ₃	<i>p</i> _{<i>i</i>₁<i>j</i>₃}	<i>A</i> ₁₃	<i>A</i> ₂₃	<i>A</i> ₃₃	...	<i>A</i> _{<i>H</i>3}	<i>d</i> _{<i>i</i>₁<i>j</i>₃}
<i>i</i> ₂	<i>j</i> ₁	<i>p</i> _{<i>i</i>₂<i>j</i>₁}	<i>A</i> ₁₁	<i>A</i> ₂₁	<i>A</i> ₃₁	...	<i>A</i> _{<i>H</i>1}	<i>d</i> _{<i>i</i>₂<i>j</i>₁}
<i>i</i> ₂	<i>j</i> ₂	<i>p</i> _{<i>i</i>₂<i>j</i>₂}	<i>A</i> ₁₂	<i>A</i> ₂₂	<i>A</i> ₃₂	...	<i>A</i> _{<i>H</i>2}	<i>d</i> _{<i>i</i>₂<i>j</i>₂}
...
<i>i</i> _{<i>m</i>}	<i>j</i> _{<i>n</i>}	<i>p</i> _{<i>i</i>_{<i>m</i>}<i>j</i>_{<i>n</i>}}	<i>A</i> _{1<i>n</i>}	<i>A</i> _{2<i>n</i>}	<i>A</i> _{3<i>n</i>}	...	<i>A</i> _{<i>H</i><i>n</i>}	<i>d</i> _{<i>i</i>_{<i>m</i>}<i>j</i>_{<i>n</i>}}

Table 2: MCI interaction matrix (schematic).

requires every variable to be ratio-scaled, non-negative and greater than zero. Thus, also market shares equal to zero, $p_{ij} = 0$, what may occur, are invalid. A simple way to correct the raw data accepting a small bias is to increase the variable by a small constant (Kubis and Hartmann, 2007; Wieland, 2015a) and/or to aggregate the submarkets (Perales, 2002; Tihl and Oruc, 2012). Anyway, the raw data should be adjusted by removing singular instances and outliers to fulfil the LIFO/LOFI requirements mentioned above (Huff and McCallum, 2008; Wieland, 2015a).

Interval scaled variables (e.g. consumer judgements in rating scales) which may contain negative values can be transformed by the *zeta-squared transformation* (Cooper and Nakanishi, 1983):

$$z_{h_{ij}} = \frac{X_{h_{ij}} - \overline{X_{h_i}}}{\rho_{h_i}}, \tag{18}$$

$$\zeta_{h_{ij}}^2 = \begin{cases} (1 + z_{h_{ij}}^2) & \text{if } z_{h_{ij}} \geq 0 \\ \left(\frac{1}{1+z_{h_{ij}}^2}\right) & \text{if } z_{h_{ij}} \leq 0 \end{cases} \tag{19}$$

where $z_{h_{ij}}$ is the z-standardized score of $X_{h_{ij}}$ and $\zeta_{h_{ij}}^2$ is the zeta-squared value resulting from $z_{h_{ij}}$.

Nominal scaled variables (e.g. store chains, brands) cannot be processed directly in the MCI Model but can be transformed into dummy variables which are ignored in the log-centering transformation.

All in all, an econometric market share/market area analysis using the MCI Model on condition of existing research questions and hypotheses contains the following 8 steps:

1. Define the study area and divide it into i submarkets (here: customer origins).
2. Obtain the relevant variables: shopping trips and/or expenditures on the individual/household level (O_{ij}), H characteristics of the j regarded suppliers and the transport costs, d_{ij} .
3. *If necessary*, correct or transform the variables to match the requirements of the log-centering transformation: if some O_{ij} or other variables are equal to zero, add a constant, aggregate the submarkets in the study area and remove outliers, respectively. If there are interval scaled variables, transform them by formulae 18 and 19. If there are nominal variables, transform them to dummy variables.
4. Calculate the market shares, p_{ij} , by formula 17.
5. Store the submarkets, suppliers, local market shares and the H explanatory variables in an interaction matrix (see Table 2).
6. Apply the log-centering transformation (formula 14) to the previously created interaction matrix.
7. Apply the OLS regression to the log-centering transformed interaction matrix with $\log\left(\frac{p_{ij}}{p_i}\right)$ as dependent variable treating the model like any other multiple linear regression model: determine the estimators, compute fit measures and inference statistics for the hypothesis tests.
8. Interpret the model results. For market share predictions, insert the estimated parameters in the nonlinear model using formula 13 or formulae 15 and 16.

R implementation

As outlined in the former section, market area analysis for retail and service locations requires a mixture of descriptive and inference statistics, iterative optimization and, of course, a lot of processing

and preparation of empirical data which may be even more complex (or, at least, more time-consuming) than the models themselves. Except the Huff Model basis formulation which is implemented in R by the packages **SpatialPosition** and **huff-tools** none of the mentioned models were integrated in R yet. The former mentioned package is focused on graphical visualization and other spatial interaction models while the latter combines the basic Huff formula with GIS-related functions. The Multiplicative Competitive Interaction Model and the related procedures of data handling are just as little integrated in R as the OLS and nonlinear fitting procedures for the Huff Model and the MCI Model, respectively. The motivation of the presented package **MCI** is to fill this gap.

The MCI package

The Huff Model and the Multiplicative Competitive Interaction (MCI) Model are implemented in the **MCI** package which is focused on:

1. Fitting the mentioned models by empirically observed data using the MCI linearization (log-centering transformation) combined with OLS regression (functions `mci.fit()`, `mci.transmat()` and `mci.transvar()`) and the Huff Model optimization algorithm described above (functions `huff.attrac()` and `huff.fit()`).
2. The steps of data preparation and processing to make empirical data usable in these models, especially the creation and processing of interaction matrices used in MCI analyses which is subject of the function `ijmatrix.create()`.

The **MCI** package also provides tools that can be used for descriptive analyses of empirical or estimated market areas, such as zoning (see the function `shares.segm()`) and, of course, the basic Huff and MCI formulations (see the functions `mci.shares()` and `huff.shares()`, respectively). The correction of variables to match the MCI requirements can be done by the functions `var.correct()` and `var.asdummy()`.

The input of the most functions is required to be a "data.frame" where the first function argument is the dataset name, followed by a set of variable names (columns) each one in double quotation marks and further function arguments. Also, the output of nearly every function is a "data.frame", except the `mci.fit()` function which returns an object of type "lm" and `model.fit()` which returns a "list". Three functions (`huff.decay()`, `model.fit()` and `shares.total()`) also provide an optional graphical output.

The package does not import any other packages (except some already implemented functions from **stats** and **graphics**, of course) and does not contain any non-R scripts.

The data used in the following examples is distributed over several datasets to demonstrate the several data sources and the components of a market area analysis, respectively. In fact, there is no need to split the working data like this, but, of course, it is recommended since the voluminous data may lead to confusion.

Examples

Analyzing market areas of single locations

The first example deals with a more descriptive analysis of empirical market areas obtained by a POS survey (customer spotting technique). The package-included example dataset `shopping1` is a survey conducted at two supply locations in the east of Karlsruhe, Germany, in May 2016.

The dataset contains 434 surveyed individuals at both locations, including 410 cases from the main survey and 24 cases from the pretest. Amongst other things, the respondents were asked about their place of residence, their shopping preferences in general (last shopping trip with respect to different goods) and their on-site shopping behaviour (duration of stay, expenditures). The customers' origin is stored in the column `resid_code`. The variable `POS` indicates the location: "POS1" is a town centre and thus, an evolved retail agglomeration, while "POS2" is an out-of-town planned shopping centre.

It should be noted that a POS survey mostly does not fulfill the requirements of statistical representativity: besides that shopping behaviour differs by weekdays, time periods and, of course, weather, independent from the size and heterogeneity of the sample, the statistical population is unknown.

```
library(MCI)
data(shopping1)
# The survey dataset
data(shopping2)
# Dataset with distances and travel times
```

The first step is to filter the dataset since only the interviews are needed which were conducted at both supply locations simultaneously. We subset the relevant data and store it into a new working data frame (`shopping1_adj`) which we use to create an interaction matrix of type "data.frame" called `ijmatrix` using the function `ijmatrix.create()`:

```
shopping1_adj <- shopping1[(shopping1$weekday != 3) & (shopping1$holiday != 1)
& (shopping1$survey != "pretest"),]
# Removing every case from tuesday, holidays and the ones belonging to the pretest

ijmatrix <- ijmatrix.create(shopping1_adj, "resid_code", "POS", "POS_expen")
# Creates an interaction matrix based on the observed frequencies (automatically)
# and the POS expenditures (Variable "POS_expen" separately stated)

ijmatrix
  interaction resid_code POS freq_ij_abs freq_i_total p_ij_obs
1 resid1-POS1 resid1 POS1 91 113 0.8053097
2 resid1-POS2 resid1 POS2 22 113 0.1946903
3 resid10-POS1 resid10 POS1 3 7 0.4285714
...
freq_ij_abs_POS_expen freq_i_total_POS_expen p_ij_obs_POS_expen
1 2318.25 3245.25 0.71435174
2 927.00 3245.25 0.28564826
3 30.00 328.00 0.09146341
...
```

The resulting data frame (rows 1-3 are displayed) contains the interaction (from each origin to each destination) in the first column (`interaction`) and each origin and destination in the columns `resid_code` and `POS`, respectively, both adopted from the column names in the input dataset. The absolute number of respondents and the total value of expenditures from the origins, i , at the destinations, j , is stored in `freq_ij_abs` and `freq_ij_abs_POS_expen`, respectively. The `freq_i_total*` columns contain the total number/sum of observed customers/expenditures from each origin, while the `p_ij_obs*` are the local market shares of customers and expenditures, p_{ij} , respectively. The names are set automatically based on the original variable names (e.g. `POS_expen`).

The next step is the zoning of the empirical market areas by driving time using the function `shares.segm()` and the travel time stored in the `shopping2` dataset. The routes were calculated in R using the package `ggmap` (Kahle and Wickham, 2013). We want to know how much of the customers and expenditures come from origins with a maximal travel time of 10, 20, 30 and more than 30 minutes:

```
ijmatrix_dist <- merge(ijmatrix, shopping2, by.x = "interaction", by.y = "route",
all.x = TRUE)
# Adding the distances and travel times

visit <- shares.segm(ijmatrix_dist, "resid_code", "POS", "d_time", "freq_ij_abs",
0, 10, 20, 30)
# Segmentation by travel time using the number of customers/visitors
# Parameters: interaction matrix (data frame), columns with origins and destinations,
# variable to divide in classes, absolute frequencies/expenditures, class segments

expen <- shares.segm(ijmatrix_dist, "resid_code", "POS", "d_time",
"freq_ij_abs_POS_expen", 0, 10, 20, 30)
# Segmentation by travel time using the POS expenditures

visit
  d_time_class POS1_abs POS1_rel POS2_abs POS2_rel
1 0-10 108 72.483221 58 40.277778
2 10-20 24 16.107383 62 43.055556
3 20-30 10 6.711409 12 8.333333
4 Other 7 4.697987 12 8.333333

expen
  d_time_class POS1_abs POS1_rel POS2_abs POS2_rel
1 0-10 2858.25 76.418689 2900.0 34.354491
2 10-20 541.00 14.464274 4052.4 48.006255
3 20-30 283.00 7.566339 478.0 5.662568
4 Other 58.00 1.550698 1011.0 11.976686
```

The "data.frame" output of the used function contains the segment classes, named based on the input variable name (d_time_class), e.g. d_time_class="0-10" represents the zone up to 10 minutes of travel time. The further columns contain the sums and the percentage shares for each location, respectively, both named based on the original values (e.g. POS1_abs, POS1_re1).

We see that, at "POS1" (town centre) 108 surveyed customers come from an origin up to 10 minutes of driving time what corresponds to 72.48% of all customers. At POS2 (out-of-town planned shopping centre), only 40.28% of the visitors are generated from places of residence where to drive less or equal to 10 minutes. The difference is more clear when looking at the expenditures: at the town centre, 76.42% of the observed expenditures are spent by customers from origins of the first driving time class, while the share in the corresponding class at the shopping centre is 34.35%.

Since the survey was only conducted at these two supply locations but not at other possible competitive locations, it would not make any sense to use these empirical market areas in an econometric market area model. But, of course, it is possible to analyze the distance decay on the level of single locations.

Before fitting and plotting distance decay functions, some corrections of the data have to be made and, since we don't have local market shares, the dependent variable reflecting the intensity of interaction (surveyed visitors per 1,000 inhabitants) has to be calculated:

```
ijmatrix_dist$freq_ij_abs_cor <- var.correct(ijmatrix_dist$freq_ij_abs,
                                             corr.mode = "inc", incby = 0.1)
# Correcting the absolute values (frequencies) by increasing by 0.1

data(shopping3)
ijmatrix_alldata <- merge(ijmatrix_dist, shopping3)
# Adding the information about the origins (places of residence) stored in shopping3

ijmatrix_alldata$visitper1000 <- (ijmatrix_alldata$freq_ij_abs_cor /
                                  ijmatrix_alldata$resid_pop2015) * 1000
# Calculating the dependent variable
# visitper1000: surveyed customers per 1,000 inhabitants of the origin

ijmatrix_alldata <- ijmatrix_alldata[(!is.na(ijmatrix_alldata$visitper1000))
& (!is.na(ijmatrix_alldata$d_time)),]
# Removing NAs (data for some outlier origins and routes not available)

POS1 <- ijmatrix_alldata[ijmatrix_alldata$POS == "POS1",]
# Dataset for POS1 (town centre)
POS2 <- ijmatrix_alldata[ijmatrix_alldata$POS == "POS2",]
# Dataset for POS2 (out-of-town shopping centre)
```

A fit of distance decay functions can be done with the package function `huff.decay()` which compares four types of possible distance decay functions (linear, power, exponential, logistic). For both locations, "POS1" and "POS2", we test the influence of distance in km (d_km) and travel time in minutes (d_time) on the dependent variable (visitper1000):

```
huff.decay(POS1, "d_km", "visitper1000")
  Model type Intercept p Intercept Slope p Slope R-Squared Adj. R-squared
1 Linear 0.7354 2e-04 -0.0455 0.0038 0.216 0.1936
2 Power 1.9121 0.4434 -1.5267 2e-04 0.333 0.3139
3 Exponential 0.2788 0.0222 -0.1353 0.0044 0.2092 0.1866
4 Logistic 1.6999 0.0164 0.1823 0.0026 0.2311 0.2092

huff.decay(POS1, "d_time", "visitper1000")
  Model type Intercept p Intercept Slope p Slope R-Squared Adj. R-squared
1 Linear 1.2112 0 -0.0585 1e-04 0.3516 0.3331
2 Power 34.6019 0.0289 -2.2968 3e-04 0.3211 0.3017
3 Exponential 0.691 0.6322 -0.1432 0.0027 0.23 0.208
4 Logistic 0.2659 0.7795 0.2058 6e-04 0.2893 0.2689

huff.decay(POS2, "d_km", "visitper1000")
  Model type Intercept p Intercept Slope p Slope R-Squared Adj. R-squared
1 Linear 0.6734 0 -0.0316 7e-04 0.2812 0.2606
2 Power 5.7372 0.0348 -1.5978 0 0.3863 0.3688
3 Exponential 0.9161 0.8318 -0.1605 0 0.4086 0.3917
```

4	Logistic	-0.4868	0.4212	0.2043	1e-04	0.3444	0.3256
---	----------	---------	--------	--------	-------	--------	--------

```

huff.decay(POS2, "d_time", "visitper1000")
Model type Intercept p Intercept Slope p Slope R-Squared Adj. R-squared
1 Linear 0.9353 0 -0.0411 1e-04 0.3706 0.3526
2 Power 213.9932 7e-04 -2.7363 0 0.4213 0.4048
3 Exponential 2.3946 0.14 -0.184 0 0.4191 0.4025
4 Logistic -1.8572 0.031 0.2441 0 0.3835 0.3659
    
```

The `huff.decay()` function returns a "data.frame" containing a summary of the regression results and a plot of the four estimated functions and the observed values (see Figure 1). The output shows the model estimators (Intercept and Slope), their p-values (p Intercept, p Slope) and the goodness of fit measures (R-Squared, Adj. R-Squared) for every model type.

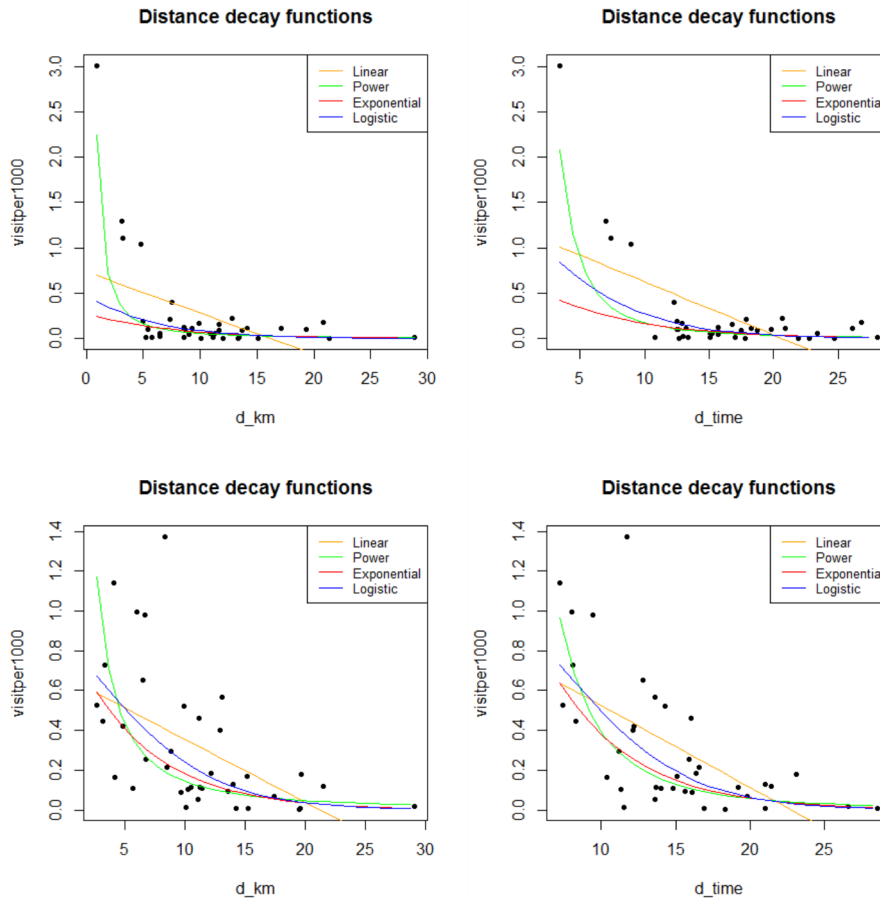


Figure 1: Distance decay functions for POS1 (top) and POS2 (bottom).

Note that the nonlinear models are not fitted via nonlinear regression but by linearization and retransformation, thereafter, since the usually used distance decay functions mentioned above are intrinsically linear (Rawlings et al., 1998). Internally, the linearized models are fitted via `lm()`. It is important to say that the usage of logarithmic transformations has, apart from many advantages of linearization, also serious drawbacks: with respect to gravity models, Silva and Tenreiro (2006) point out that OLS fitting of log-transformed data produces significant biases depending on heteroscedasticity. The same issue is also addressed by Manning and Mullahy (2001).

Huff (1962) assumed that a power function fits best to describe the distance decay. Since other types of distance decay functions are possible, Kanhäußer (2007) uses the explained variance as a criterion for choosing the best function type. Note that, strictly speaking, the R^2 values of the models can not be compared directly since they rest on different dependent variables. In the practical application, the choice for the preferred function type should be based on examining the model results and the plot as a case-by-case decision. With respect to R^2 the best distance decay functions for the two locations are:

$$\hat{I}_{ij_{POS1}} = 1.91 d_{km_{ij}}^{-1.53}, \tag{20}$$

$$\hat{I}_{ijPOS1} = 1.21 - 0.06 d_time_{ij}, \quad (21)$$

$$\hat{I}_{ijPOS2} = 0.92 e^{-0.16 d_km_{ij}}, \quad (22)$$

$$\hat{I}_{ijPOS2} = 213.99 d_time_{ij}^{-2.74}, \quad (23)$$

where \hat{I}_{ijPOS1} and \hat{I}_{ijPOS2} is the expected value of interaction intensity from i to j (visitors per 1,000 inhabitants) for the supply locations POS1 and POS2, respectively, d_km_{ij} is the distance from i to j in km, d_time_{ij} is the travel time from i to j and e is Euler's number ≈ 2.71828 .

As expected, the distance/travel time influences the interaction intensity significantly and, as Huff (1962) states, mostly exponentially. In the second case, the regression regarding POS1 with the travel time, surprisingly, the best fit can be achieved by a linear function. But, all in all, the model fits can be regarded as rather poor, since the highest R^2 is equal to 0.42 in the fourth function (POS2 with travel time). Of course, that is because the distance/travel time is just one important determinant of store choice, while the attraction and other attributes of the retail locations are not considered here.

Econometric market area analysis using the MCI Model

Since, in the Huff Model, the store choices are explained by sales area and travel time, we want to test these hypotheses empirically with respect to grocery shopping trips. First, we have to calculate the dependent variable (market shares) and to link it to the explanatory variables in an interaction matrix, then we need to apply the log-centering transformation and, finally, we have to fit the MCI Model.

As in the former example, we use the shopping1 dataset which also contains questions about the general shopping behaviour. The respondents were asked about the destination of their last grocery shopping trip (gro_purchase_code) and the related expenditures (gro_purchase_expen).

We have to clean our working dataset from respondents living outside our study area, since we want to analyze the shopping patterns of the grocery shoppers in the eastern districts of Karlsruhe:

```
data(shopping1)
# Survey dataset
data(shopping3)
# Dataset containing information about the city districts
data(shopping4)
# Dataset containing the grocery stores
shopping1_KAeast <- shopping1[shopping1$resid_code %in%
  shopping3$resid_code[shopping3$KA_east == 1],]
# Extracting only inhabitants of the eastern districts of Karlsruhe
```

From the adjusted dataset, we create an interaction matrix using the function `ijmatrix.create()` with default values (no further adjusting except ignoring NA values). The calculation includes the shopping trip frequency that is counted automatically and the expenditures that must be specified separately by stating the variable containing the individual trip expenditures, `gro_purchase_expen`. The interaction matrix is stored in a new "data.frame", `ijmatrix`.

```
ijmatrix <- ijmatrix.create(shopping1_KAeast, "resid_code", "gro_purchase_code",
  "gro_purchase_expen")

ijmatrix
  interaction resid_code gro_purchase_code freq_ij_abs freq_i_total p_ij_obs
1 resid1-ALDI1 resid1 ALDI1 10 186 0.053763441
2 resid1-ALDI11 resid1 ALDI11 0 186 0.000000000
3 resid1-ALDI2 resid1 ALDI2 0 186 0.000000000
...
freq_ij_abs_gro_purchase_expen freq_i_total_gro_purchase_expen
1 420.0 5270.0
2 0.0 5270.0
3 0.0 5270.0
...
p_ij_obs_gro_purchase_expen
1 0.0796963947
2 0.0000000000
3 0.0000000000
...
```

The interaction matrix (rows 1-3 are displayed) shows the interaction (origin-destination) in the first column interaction and the origins and destinations in the columns `resid_code` and `gro_purchase_code`, respectively, adopted from the column names in the input dataset `shopping1_KAeast`. The absolute number of respondents and the total value of expenditures from the origins, i , at the destinations, j , is stored in `freq_ij_abs` and `freq_ij_abs_POS_expen`, respectively. The names are set automatically based on the original variable names (e.g. `POS_expen`). The `freq_i_total*` columns contain the total number/sum of observed customers/expenditures from each origin, while the `p_ij_obs*` are the local market shares of customers and expenditures, p_{ij} , respectively.

Next, we calculate the total customers and expenditures and the corresponding over-all market shares, respectively, using the function `shares.total()`:

```
shares.total(ijmatrix, "resid_code", "gro_purchase_code", "p_ij_obs",
            "freq_i_total")
# Total values for the shopping trips

  suppliers_single sum_E_j   share_j
1             ALDI1      11 0.039426523
2             ALDI11       1 0.003584229
3             ALDI2       3 0.010752688
4             ALDI4       1 0.003584229
...

shares.total(ijmatrix, "resid_code", "gro_purchase_code",
            "p_ij_obs_gro_purchase_expen", "freq_i_total_gro_purchase_expen")
# Total values for the shopping expenditures

  suppliers_single sum_E_j   share_j
1             ALDI1    470.0 0.0492378608
2             ALDI11    60.0 0.0062856844
3             ALDI2    87.0 0.0091142423
4             ALDI4    80.0 0.0083809125
...
```

The resulting tables (rows 1-4 are displayed) contain 42 suppliers: in the first row, the store "ALDI1" has a share of about 3.94% of the obtained shopping trips and about 4.92% of the related expenditures. Obviously, the survey contains several singular instances and outliers: as can be seen in the second and third row of the interaction matrix, there are observed market shares equal to zero. The table with the total market areas show several stores only observed once (e.g. "ALDI11").

Thus, this interaction matrix cannot be processed in the MCI Model since it would not pass the log-centering transformation. Consequently, the data has to be "cleaned" distinctly by creating a corrected and simplified interaction matrix. Only stores obtained more than twice are incorporated and the absolute values are increased by a small constant of 0.1 before calculating the shares.

```
ijmatrix_adj <- ijmatrix.create(shopping1_KAeast, "resid_code",
                               "gro_purchase_code", "gro_purchase_expen",
                               remSing = TRUE, remSing.val = 1,
                               remSingSupp.val = 2, correctVar = TRUE,
                               correctVar.val = 0.1)
# Removing singular instances/outliers (remSing = TRUE) incorporating
# only suppliers which are at least obtained three times (remSingSupp.val = 2)
# Correcting the values (correctVar = TRUE)
# by adding 0.1 to the absolute values (correctVar.val = 0.1)
```

There are still some observations that have to be excluded because, in the survey, any kind of grocery shopping trip was inquired, including non-relevant stores and shopping channels (such as bakeries, health food shops and even the local weekly market). There has been some incomplete answers as well (`gro_purchase_code="X_INCOMPLETE_STORE"`.) Thus, the interaction matrix has to be adjusted again:

```
ijmatrix_adj <- ijmatrix_adj[(ijmatrix_adj$gro_purchase_code != "REFORMHAUSBOESER") &
                             (ijmatrix_adj$gro_purchase_code != "WMARKT_DURLACH") &
                             (ijmatrix_adj$gro_purchase_code != "X_INCOMPLETE_STORE"),]
# Remove non-regarded observations
```

Now, the interaction matrix (rows 1-3 are displayed) consists of 11 customer origins and 11

suppliers (121 rows). Structured as mentioned above, there are no zero values anymore and the absolute values (`freq_ij_abs` and `freq_ij_abs_POS_expen`) are increased by 0.1:

```
ijmatrix_adj
  interaction resid_code gro_purchase_code freq_ij_abs freq_i_total
1 resid1-ALDI1 resid1 ALDI1 10.1 172.4
2 resid1-ALDI2 resid1 ALDI2 0.1 172.4
3 resid1-CAP1 resid1 CAP1 18.1 172.4
...
  p_ij_obs freq_ij_abs_gro_purchase_expen freq_i_total_gro_purchase_expen
1 0.0585846868 420.1 4745.4
2 0.0005800464 0.1 4745.4
3 0.1049883991 224.1 4745.4
...
  p_ij_obs_gro_purchase_expen
1 8.852784e-02
2 2.107304e-05
3 4.722468e-02
...
```

In the next step, we have to add the travel times stored in the dataset `shopping2` (the route calculation was made in R using the `ggmap` package) and the grocery store characteristics (sales area in `sqm` and store chain, collected in June 2016 subsequent to the survey) from the `shopping4` dataset:

```
ijmatrix_dist <- merge(ijmatrix_adj, shopping2, by.x="interaction", by.y="route")
# Include the distances and travel times (shopping2)

ijmatrix_alldata <- merge(ijmatrix_dist, shopping4, by.x = "gro_purchase_code",
  by.y = "location_code")
# Adding the store information (shopping4)
```

The next step is to apply the necessary log-centering transformation to the interaction matrix. The function `mci.transmat()` processes this transformation with a given number of MCI variables. The functions output is a "data.frame" containing the transformed variables (ready for OLS regression):

```
ijmatrix_transf <- mci.transmat(ijmatrix_alldata, "resid_code", "gro_purchase_code",
  "p_ij_obs", "d_time", "salesarea_qm")

ijmatrix_transf
  resid_code gro_purchase_code p_ij_obs_t d_time_t salesarea_qm_t
1 resid1 ALDI1 0.5586409 0.060847455 -0.193400118
14 resid1 ALDI2 -1.4456805 -0.008788473 -0.193400118
23 resid1 CAP1 0.8119981 -0.091762709 -0.545582636
...
109 resid29 REWE1 -0.1893441 -0.060611802 -0.001514591
119 resid29 TREFF1 -0.1893441 -0.027694956 -0.169919022
6 resid3 ALDI1 -0.2768831 0.058442499 -0.193400118
...
```

In the transformed interaction matrix (2 x 3 rows are displayed), the column names of the origins (`resid_code`) and destinations (`gro_purchase_code`) are adopted from the columns in the input dataset. The metric MCI variables are marked with a "_t" to indicate that they were transformed (e.g. `d_time_t` is the log-centering transformation of `d_time` which contains the travel time, d_{ij}). The transformed interaction matrix is in alphabetical order with respect to the origins (first) and the locations (second). If stated, the function recognizes dummy variables which are, of course, not transformed.

To combine transformation and fitting in one step, we use the function `mci.fit()` whose parameters are equal to those in `mci.transmat()` except that the column containing the market shares must be stated as the first variable. The default is a no-intercept model (to be set by the logical argument `origin` with default `TRUE`). The function returns an object of type "lm" which can be accessed via `summary()`:

```
mci_trips <- mci.fit(ijmatrix_alldata, "resid_code", "gro_purchase_code", "p_ij_obs",
  "d_time", "salesarea_qm")
# shares: "p_ij_obs", explanatory variables: "d_time", "salesarea_qm"

summary(mci_trips)
```

```

Call:
lm(formula = mci_formula, data = mciworkfile)

Residuals:
    Min       1Q   Median       3Q      Max
-1.27457 -0.28725 -0.02391  0.32163  1.29351

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
d_time_t      -1.2443     0.2319  -5.367 4.02e-07 ***
salesarea_qm_t  0.9413     0.1158   8.132 4.59e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4458 on 119 degrees of freedom
Multiple R-squared:  0.4603,    Adjusted R-squared:  0.4512
F-statistic: 50.74 on 2 and 119 DF,  p-value: < 2.2e-16

```

The output can be interpreted like any other linear model fitted by `lm()`. The results show a strong significant influence of the explanatory variables (travel time and sales area) on the observed local market shares (both $p < 0.001$). As the estimators are exponents in the original nonlinear model, the impact of travel time can be regarded as negative superlinear ($\lambda = -1.2443$) while the effect of the sales area is positive sublinear ($\gamma = 0.9413$). Thus, the model fit corresponds to the theoretical underpinning of the utility function in the Huff Model (Huff, 1962).

For the interpretation of the explained variance it is noteworthy that the common formulation of R^2 is only designed for models that include an intercept. For no-intercept models, in R, the function `summary()` returns a modified R^2 which reflects the explained variance of the no-intercept model but can *not* be compared to the related fit measure in a model including an intercept. Apart from this, the model fit seems to be in need of improvement ($R^2 = 0.4458$ and $Adj.R^2 = 0.4512$, respectively).

But the market shares with respect to customer flows are not to be confused with market shares of expenditures: especially in grocery shopping, there are different kinds of shopping trips, such as less frequent *major trips* with high efforts and expenditures and more frequent and fast *fill-in trips* with low expenditures (Reutterer and Teller, 2009). Thus, the MCI analysis is repeated with the shares of expenditures (`p_ij_obs_gro_purchase_expen`) as dependent variable:

```

mci_expen <- mci.fit(ijmatrix_alldata, "resid_code", "gro_purchase_code",
                    "p_ij_obs_gro_purchase_expen", "d_time", "salesarea_qm")

summary(mci_expen)

Call:
lm(formula = mci_formula, data = mciworkfile)

Residuals:
    Min       1Q   Median       3Q      Max
-2.07495 -0.61794 -0.07452  0.70263  2.60085

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
d_time_t      -2.3788     0.4517  -5.267 6.26e-07 ***
salesarea_qm_t  2.0409     0.2255   9.051 3.30e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8683 on 119 degrees of freedom
Multiple R-squared:  0.4954,    Adjusted R-squared:  0.487
F-statistic: 58.43 on 2 and 119 DF,  p-value: < 2.2e-16

```

As in the first model, both predictors are highly significant, but, in contrast, the impact of size (sales area) is also superlinear. The model fit is a little better than in the first model, what may also be explained by a smaller bias based on the variable correction above: due to their dimensions, the increase of the absolute values by 0.1 has a smaller impact on the expenditures than on the shopping trips.

The utility/attraction function (or, more precisely, its deterministic component) with respect to the shopping trips (first model) can be written as:

$$U_{ijtrips} = salesarea_qm_j^{0.9413} d_time_{ij}^{-1.2443}, \quad (24)$$

where $U_{ijtrips}$ is the utility/attraction of grocery store j for the customers in submarket/origin i , derived from the shares of empirically obtained shopping trips, $salesarea_qm_j$ is the sales area of store j and d_time_{ij} is the travel time from i to j .

In consequence, the utility function is included in the Huff/MCI probability function. Thus, the local market shares of shopping trips can be estimated by:

$$\hat{p}_{ijtrips} = \frac{salesarea_qm_j^{0.9413} d_time_{ij}^{-1.2443}}{\sum_{j=1}^n salesarea_qm_j^{0.9413} d_time_{ij}^{-1.2443}}, \quad (25)$$

where $\hat{p}_{ijtrips}$ is the expected market share of shopping trips of the grocery store j in the submarket i .

Analogously, the utility/attraction function and the market share/probability function of the expenditures (second model) are:

$$U_{ijexpen} = salesarea_qm_j^{2.0409} d_time_{ij}^{-2.3788}, \quad (26)$$

$$\hat{p}_{ijexpen} = \frac{salesarea_qm_j^{2.0409} d_time_{ij}^{-2.3788}}{\sum_{j=1}^n salesarea_qm_j^{2.0409} d_time_{ij}^{-2.3788}}, \quad (27)$$

where $\hat{p}_{ijexpen}$ is the expected market share of expenditures of the grocery store j in the submarket i .

A market share prediction can be done using the functions `mci.shares()` and `huff.shares()`. These functions can be used similarly but differ in the formulation of the utility/attraction function: according to the Huff Model, `huff.shares()` allows only two explanatory variables (attraction/size and transport costs) but three types of weighting function for each variable (power, exponential, logistic). The function `mci.shares()` is able to process any number of variables but only using the power function from the MCI Model either retransformed or transformed (inverse log-centering transformation).

We predict market shares using the function `mci.shares()` and the estimations from the second model (expenditures). The variables and their weightings are function arguments which have to be stated one after another (`variable1, weighting1, variable2, weighting2, ...`). The estimated interaction matrix containing the predicted shares is stored in a new dataset, `expen`:

```
expen <- mci.shares(ijmatrix_alldata, "resid_code", "gro_purchase_code",
                  "salesarea_qm", 2.0409, "d_time", -2.3788)
# MCI market share prediction with two variables
# salesarea_qm (weighting power function with exponent equal to 2.0409)
# d_time (weighting power function with exponent equal to -2.3788)

expen
  gro_purchase_code  interaction resid_code
1                ALDI1  resid1-ALDI1  resid1
14               ALDI2  resid1-ALDI2  resid1
23                CAP1  resid1-CAP1   resid1
...
  d_time salesarea_qm storetype_dc store_chain
1      5.4          900           1         Aldi
14     4.6          900           1         Aldi
23     3.8          400           0         Edeka
...
      U_ij sum_U_ij      p_ij
1  1.936847e+04  9682538  2.000350e-03
14  2.836255e+04  9682538  2.929247e-03
23  8.537978e+03  9682538  8.817913e-04
...
```

The interaction matrix (rows 1-3 are displayed) contains the submarkets (`resid_code`), the suppliers (`gro_purchase_code`), the interaction between them (`interaction`), the explanatory variables (`d_time`, `salesarea_qm`) and the steps of calculation, named according to the Huff Model (`U_ij`, `sum_U_ij`) and resulting in the local market shares stored in the last column (`p_ij`). The sum of `p_ij` is equal to the

sum of submarkets, since, on the submarket level, the local market shares sum up to one.

Before using the MCI Model for further market share predictions, the validity of the model should be improved since store choices are, of course, not only influenced by size and transport costs. Especially in grocery retailing, there is a great heterogeneity between the store formats (supermarket, discounter etc.) and the store chains, such as concerning the image or price level of a chain. These more qualitative differences should be reflected in a grocery store market area model. According to [Wieland \(2015a\)](#), the model above is extended with dummy variables reflecting the grocery store chain.

The chains are already stored in the interaction matrix (column `store_chain`, adopted from `shopping4`). As they are nominal scaled variables in character format, they have to be converted to dummy variables using the function `var.asdummy()` which returns a new "data.frame" containing corresponding dummy variables (0, 1) named automatically based on the original characteristics and marked with "`_DUMMY`". Since they are in the same order, they can be directly attached:

```
chain <- var.asdummy(ijmatrix_alldata$store_chain)
# Converting the character vector (column store_chain) to dummy variables
# and storing in a new data frame

chain
  Aldi_DUMMY Edeka_DUMMY Lidl_DUMMY Netto_DUMMY Real_DUMMY Rewe_DUMMY Treff 3000_DUMMY
1           1           0           0           0           0           0           0
2           1           0           0           0           0           0           0
...
66          0           1           0           0           0           0           0
67          0           0           1           0           0           0           0
...

ijmatrix_alldata <- cbind(ijmatrix_alldata, chain)
# Add dummy dataset to interaction matrix
```

In the next step, we repeat the fitting of the second MCI Model (expenditures) including the new dummy variables. Since one dummy is explained by the content of all the others, the last dummy is not used:

```
mci_expen2 <- mci.fit(ijmatrix_alldata, "resid_code", "gro_purchase_code",
                     "p_ij_obs_gro_purchase_expen", "d_time", "salesarea_qm",
                     "Aldi_DUMMY", "Edeka_DUMMY", "Lidl_DUMMY", "Netto_DUMMY",
                     "Real_DUMMY", "Rewe_DUMMY")
# Same model as above with additional dummy variables

summary(mci_expen2)
Call:
lm(formula = mci_formula, data = mciworkfile)

Residuals:
    Min       1Q   Median       3Q      Max
-2.1601 -0.4338 -0.1041  0.2561  2.5342

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
d_time_t      -2.56243    0.41753  -6.137 1.28e-08 ***
salesarea_qm_t 1.31622    0.30754   4.280 3.94e-05 ***
Aldi_DUMMY    -0.05658    0.17995  -0.314 0.753763
Edeka_DUMMY     0.11873    0.12051   0.985 0.326637
Lidl_DUMMY    -0.59177    0.24441  -2.421 0.017060 *
Netto_DUMMY   -0.19785    0.25719  -0.769 0.443330
Real_DUMMY     1.32882    0.33093   4.015 0.000107 ***
Rewe_DUMMY    -0.38429    0.24012  -1.600 0.112299
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7955 on 113 degrees of freedom
Multiple R-squared:  0.5978,    Adjusted R-squared:  0.5694
F-statistic: 21 on 8 and 113 DF,  p-value: < 2.2e-16
```

The independent variables `d_time_t` and `salesarea_qm_t` are still statistically significant (both

$p < 0.001$) but especially the latter estimator has strikingly decreased. Two dummy variables are also significant: *Lidl_DUMMY* ($p < 0.05$) and *Real_DUMMY* ($p < 0.001$). Regarding R^2 and $Adj.R^2$, the model validity could be improved compared to the former model.

As this new model includes dummies, it would not make any sense to insert the estimated parameters in the multiplicative MCI function: if one dummy variable is equal to zero, the complete term is equal to zero. Thus, in this case, it is necessary to use the inverse log-centering transformation which leads to the following transformed utility/attraction function:

$$\hat{y}_{ij_{expen}} = -2.56243 \log \left(\frac{\widetilde{d_time_t_{ij}}}{\widetilde{d_time_t_i}} \right) + 1.31622 \log \left(\frac{\widetilde{salesarea_qm_t_j}}{\widetilde{salesarea_qm_t_i}} \right) - 0.59177Lidl_DUMMY_j + 1.32882Real_DUMMY_j, \quad (28)$$

where $\hat{y}_{ij_{expen}}$ is the log-centering transformed utility/attraction of store j for the customers in i , $\widetilde{d_time_t_i}$ and $\widetilde{salesarea_qm_t_j}$ are the geometric means of $d_time_t_{ij}$ and $salesarea_qm_t_j$, respectively, and *Lidl_DUMMY_j* and *Real_DUMMY_j* are dummy variables reflecting if the store chain of j is Lidl or Real, respectively. Now, the local market shares are defined by:

$$\hat{p}_{ij_{expen}} = \frac{e^{\hat{y}_{ij_{expen}}}}{\sum_{j=1}^n e^{\hat{y}_{ij_{expen}}}}, \quad (29)$$

where $\hat{p}_{ij_{expen}}$ is the expected market share of store j in i and e is Euler's number ≈ 2.71828 .

We repeat the market shares prediction including the dummies and their weights. The inverse log-centering transformation, which is required here, can be used in the function `mci.shares()` by stating the function parameter `mcitrans` to "ilc" (default: "lc"). In the next step, the total expenditures for each grocery store and the corresponding over-all shares are computed by the function `shares.total()`:

```
expen2 <- mci.shares(ijmatrix_alldata, "resid_code", "gro_purchase_code",
                    "salesarea_qm", 1.31622, "d_time", -2.56243, "Lidl_DUMMY", -0.59177,
                    "Real_DUMMY", 1.32882, mcitrans = "ilc")
# MCI market share prediction with four variables
# (ratio-scaled variables are log-centering transformed)
# salesarea_qm (multiplicative weighting with factor equal to 1.31622)
# d_time (multiplicative weighting with factor equal to -2.56243)
# Lidl_DUMMY (multiplicative weighting with factor equal to -0.59177)
# Real_DUMMY (multiplicative weighting with factor equal to 1.32882)

shares.total(expen2, "resid_code", "gro_purchase_code", "p_ij",
             "freq_ij_abs_gro_purchase_expen")
# Expected total sales and shares based on the observed local
# market potential (sum of all obtained expenditures for each origin)

  suppliers_single  sum_E_j  share_j
1          ALDI1    16.029089 0.008058517
2          ALDI2     2.157771 0.001084805
3           CAP1     7.101650 0.003570307
4          EDEKA1   790.385662 0.397361072
5          EDEKA2    34.432270 0.017310592
6          EDEKA3    25.098495 0.012618099
7           LIDL1    40.925354 0.020574946
8          NETTO1     2.063759 0.001037541
9          REAL1  1019.935563 0.512765741
10         REWE1    35.451053 0.017822778
11         TREFF1    15.506130 0.007795602
```

The result of `shares.total` is a "data.frame" containing the suppliers (`suppliers_single`), the total values (`sum_E_j`) and the corresponding over-all market shares `share_j`, where the sum of all `share_j` is equal to one. Here, the stores with the biggest shares of customer expenditures are "REAL1" (51.28%) and "EDEKA1" (39.74%).

Now, we analyze the impact of a change in the competitive environment: the biggest grocery store in the study area is Real (`gro_purchase_code = "REAL1"`). On condition that the sales area of this store is increased by 10 % (because of an increase of offered goods), how will, *ceteris paribus*, the

purchasing power flows with respect to this supplier and to all other stores change? First, we update the regarded variable in the existing interaction matrix:

```
ijmatrix_alldata[ijmatrix_alldata$gro_purchase_code == "REAL1",]$salesarea_qm <- 8525
# Replacing the sales area of REAL1 with a new value: 8525 sqm (increase of 10 %)
```

Now, we only have to repeat the MCI market share prediction above:

```
expen2_new <- mci.shares(ijmatrix_alldata, "resid_code",
                        "gro_purchase_code", "salesarea_qm", 1.31622,
                        "d_time", -2.56243,
                        "Lidl_DUMMY", -0.59177, "Real_DUMMY", 1.32882,
                        mcitrans = "ilc")

shares.total(expen2_new, "resid_code", "gro_purchase_code", "p_ij",
             "freq_ij_abs_gro_purchase_expen")
```

	suppliers_single	sum_E_j	share_j
1	ALDI1	15.787692	0.007886640
2	ALDI2	2.097968	0.001048026
3	CAP1	6.995287	0.003494451
4	EDEKA1	778.019361	0.388654556
5	EDEKA2	33.873811	0.016921444
6	EDEKA3	24.365708	0.012171732
7	LIDL1	40.262537	0.020112891
8	NETTO1	2.032720	0.001015432
9	REAL1	1048.206517	0.523624808
10	REWE1	34.910285	0.017439208
11	TREFF1	15.275568	0.007630811

The results in the total sales/shares table show an impact of the trading-up in the regarded hypermarket: the over-all share of Real increases from 51.28% to 52.36%. The other stores are affected differently: the over-all share of EDEKA1 decreases from 39.74% to 38.87%. Since this model does not consider any kind of agglomeration economies, the market shares of the competitors decrease dependent on their proximity to the hypermarket and the spatial distribution of customers.

It should be noticed that the survey in the shopping1 dataset is *not* statistically representative, since it is a POS survey not regarding customers shopping at other supply locations. Thus, the analyses and results shown here should not be overinterpreted but regarded as an example how to use the MCI-related functions in the MCI package.

Huff Model optimization

The final example deals with the problem of fitting the Huff Model on condition that no empirically observed store choices, market shares and market areas, respectively, are available. The aim of the analysis is to estimate the market areas of the grocery stores in Freiburg, Germany, based on their total annual sales and their "attraction" (size).

The dataset Freiburg1 contains the preliminary stage of a Huff interaction matrix, containing the origins (statistical districts of Freiburg, column district), codes representing the grocery stores (store), the sales area of these stores in sqm (salesarea) and the street distances between the origins and the stores (distance). The grocery stores and their characteristics were obtained in spring 2015 (Wieland, 2015b), while the street distances were calculated via network analysis in GRASS (GRASS Development Team, 2015) using OpenStreetMap vector data.

```
data(Freiburg1)
# Distance matrix and sales area
```

First, we try to approximate the market areas using a conventional Huff Model calculation using the function `huff.shares()`. This function is similar to the `mci.shares()` function and allows two explanatory variables (size, transport costs) which can be weighted by given parameters in a power, exponential or logistic function. The function type (power) and the exponents are set corresponding to the default parameters of the Huff Model: $\gamma = 1$ and $\lambda = -2$ (Güssefeldt, 2002):

```
huff_mat <- huff.shares (Freiburg1, "district", "store", "salesarea", "distance")
# Market area estimation using the Huff Model with standard parameters
# (gamma = 1, lambda = -2)
```

In the next step, the total annual sales of the grocery stores are computed based on the estimated interaction probabilities/local market shares and the grocery purchasing power potential on the district level in EUR (calculated based on the local population size and the national average expenditures for groceries), stored in the dataset Freiburg2:

```
data(Freiburg2)
# Grocery purchasing power on the city district level

huff_mat_pp <- merge (huff_mat, Freiburg2)
# Adding the purchasing power data for the city districts

huff_total <- shares.total (huff_mat_pp, "district", "store", "p_ij", "ppower")
# Total expected sales and shares

huff_total
  suppliers_single sum_E_j   share_j
1                1 4057591 0.010759819
2                10 5809861 0.015406444
3                11 1289847 0.003420383
4                12 7103210 0.018836115
5                13 3476313 0.009218400
...
```

The new dataset `huff_total` contains the expected total annual sales in EUR (`sum_E_j`) and the corresponding over-all market shares (`share_j`). Since the “real” annual sales are known (calculated by chain-based average retail space productivity, stored in the dataset Freiburg3), we compare the expected values to the observed values using the help function `model.fit()`:

```
data(Freiburg3)
# Annual sales of the grocery stores

huff_total_control <- merge(huff_total, Freiburg3, by.x = "suppliers_single",
                           by.y = "store")

model.fit(huff_total_control$annualsales, huff_total_control$sum_E_j, plotVal = TRUE)
$resids_sq_sum
[1] 2.125162e+15

$pseudorsq
[1] 0.5128422

$globerr
[1] 0.5210329

$mape
[1] 0.6383766
```

The function `model.fit()` returns a “list” with four entries containing the following goodness-of-fit measures: the sum of the squared residuals (`resids_sq_sum`), a Pseudo- R^2 measure (`pseudorsq`), the global error (`globerr`) and the MAPE (`mape`), as described in the Huff Model section. Note that these fit measures are closely related to each other. Optionally, the function returns a plot to compare the observed and expected values graphically (see the plot in Figure 2, top left).

Obviously, the fit in this Huff Model market area estimation using the default values is quite poor as can be seen from the fit measures and, of course, from the plot. E.g. the sales of the most high-selling grocery store is extremely underestimated. From this, one can conclude that also the estimated market shares/market areas will not reflect the reality to some extent.

Thus, we have to use the optimization algorithm as discussed in the Huff Model section. The function `huff.attrac()` provides one iteration of this algorithm, requiring the interaction matrix, the local market potential and the total values (e.g. sales) of the suppliers. The tolerance value to accept a difference of individual total sales or not is set equal to 5, while the function output contains the estimated total values (Internally, `huff.attrac()` uses the function `shares.total()`). We apply the function to the three datasets (Freiburg1, Freiburg2 and Freiburg3):

```
huff_total_opt1 <- huff.attrac(Freiburg1, "district", "store", "salesarea", "distance",
                              lambda = -2, dtype = "pow", lambda2 = NULL,
```

```

        Freiburg2, "district", "ppower",
        Freiburg3, "store", "annualsales",
        tolerance = 5, output = "total")
# One-time optimization (one iteration) with an accepted difference of +/- 5 %
# Output of total sales/shares

```

Note that this calculation includes a great many calculation steps while working with 63 stores and 42 statistical districts. In a test environment (computer with CPU: Intel Core i3-2100, RAM: 4.00 GB, OS: Windows 7 64bit), the command above takes about 43 seconds.

```

huff_total_opt1
  suppliers_single  sum_E_j    share_j  total_obs    diff  attrac_new_opt
1                1  7097226  0.018820246  7210720  113494.39   1329.26622
2                10  5043058  0.013373057  5600000  556941.50  1400.00000
3                11  1136322  0.003013270  1000000 -136321.73   193.82148
4                12  2849360  0.007555862  2776000  -73360.04   271.22161
...

```

Since the option `output` was set to `"total"`, the `huff.attrac()` function returns a `"data.frame"` (rows 1-4 are displayed) that contains a comparison between the observed (`total_obs`) and the expected total values (`sum_E_j`) as well as the corresponding difference (`diff`) and the estimated new attractions (`attrac_new_opt`). If output is set to `"matrix"`, the function returns an interaction matrix equal to the output of `mci.shares()` or `huff.shares()`.

Next, the validity analysis of the new model using `model.fit()` is repeated as described above:

```

model.fit(huff_total_opt1$total_obs, huff_total_opt1$sum_E_j, plotVal = TRUE)
# total_obs = observed total values, originally from dataset Freiburg3
# sum_E_j = expected total values

$resids_sq_sum
[1] 2.901841e+14

$pseudorsq
[1] 0.9334801

$globerr
[1] 0.1564878

$mape
[1] 0.1620126

```

The goodness-of-fit measures and the plot (see the plot in Figure 2, top right) reveal a much better fit: the Pseudo- R^2 increases from 0.51 to 0.93 while the error measures decrease accordingly.

To extend this optimization algorithm to a given number of iterations, the **MCI** package provides the function `huff.fit()`. First, we run two iterations decreasing the tolerance value equal to one which means a more strict check. Since this optimization takes some time, we enable the printing of status messages. The output of the function `huff.fit()` is equal to the output of `huff.attrac()` and can be processed in the same way. As above, the estimated total sales are compared to the observed sales using `model.fit()`:

```

huff_total_opt2 <- huff.fit(Freiburg1, "district", "store", "salesarea", "distance",
                          lambda = -2, dtype = "pow", lambda2 = NULL,
                          Freiburg2, "district", "ppower",
                          Freiburg3, "store", "annualsales",
                          tolerance = 1, iterations = 2, output = "total",
                          show_proc = TRUE)
# 2 iterations of the optimization algorithm with an accepted difference of +/- 1 %
# Output of total sales/shares, stored in dataset huff_total_opt2

# printing of status messages:
Iteration 1 of 2 ...
Processing location 1 ...
...

model.fit(huff_total_opt2$total_obs, huff_total_opt2$sum_E_j, plotVal = TRUE)

```

```
# total_obs = observed total values, originally from dataset Freiburg3
# sum_E_j = expected total values
```

```
$resids_sq_sum
[1] 4.806282e+13
```

```
$pseudorsq
[1] 0.9889824
```

```
$globerr
[1] 0.05946104
```

```
$mape
[1] 0.0618133
```

To run the algorithm twice with a smaller tolerance leads to a much better fit: while Pseudo- R^2 increases from 0.93 to 0.98, the global error and the MAPE reduce from about 15 and 16% to about 6%, respectively. This improvement is also obvious when looking at the plot (Figure 2, bottom left).

In the test environment mentioned above, this `huff.fit` operation takes about one and a half minute (89 seconds). Consequently, to extend the procedure to more iterations means also an increase of calculating time. We repeat the algorithm with 10 iterations and compare the results once more:

```
huff_total_opt10 <- huff.fit(Freiburg1, "district", "store", "salesarea", "distance",
                           lambda = -2, dtype= "pow", lambda2 = NULL,
                           Freiburg2, "district", "ppower",
                           Freiburg3, "store", "annualsales",
                           tolerance = 1, iterations = 10, output = "total",
                           show_proc = TRUE)
```

```
# 10 iterations of the optimization algorithm with an accepted difference of +/- 1 %
# Output of total sales/shares, stored in dataset huff_total_opt10
# with printing of status messages
```

```
model.fit(huff_total_opt10$total_obs, huff_total_opt10$sum_E_j, plotVal = TRUE)
```

```
$resids_sq_sum
[1] 185646134781
```

```
$pseudorsq
[1] 0.9999574
```

```
$globerr
[1] 0.004508996
```

```
$mape
[1] 0.004405252
```

This operation is, of course, very time-consuming: in the test environment mentioned above, ten iterations of the algorithm took about five minutes (308 seconds), while the time of an iteration decreases since the local fits (for each store/locations) also gets better with each iteration. The final result is a nearly perfect fit with a MAPE of about 0.4% and a global error of about 0.5% while the Pseudo- R^2 approaches the maximum (0.99). The expected annual sales are highly accurate as can be seen in the corresponding plot (Figure 2, bottom right).

The function `huff.fit()` allows an evaluation of the iterations by returning the corresponding global fit measures for each step (Internally, `huff.fit()` uses the `model.fit()` function). We set the function parameter `output = "diag"` to make the function return a "data.frame" containing the iteration statistics:

```
huff.fit(Freiburg1, "district", "store", "salesarea", "distance", lambda = -2,
         dtype= "pow", lambda2 = NULL,
         Freiburg2, "district", "ppower",
         Freiburg3, "store", "annualsales",
         tolerance = 1, iterations = 10, output = "diag", show_proc = TRUE)
```

```
  iterations_count resids_sq_sum pseudorsq   globerr   mape
1                1  2.886177e+14 0.9338392 0.155826392 0.162228371
```

2	2	4.806282e+13	0.9889824	0.059461039	0.061813302
3	3	9.936652e+12	0.9977222	0.031229899	0.032170472
4	4	2.318130e+12	0.9994686	0.018162793	0.019737125
5	5	6.378060e+11	0.9998538	0.010563610	0.011844831
6	6	5.567110e+11	0.9998724	0.007654992	0.007507305
7	7	1.776295e+11	0.9999593	0.005357527	0.005444095
8	8	1.778885e+11	0.9999592	0.004689937	0.004669445
9	9	1.828774e+11	0.9999581	0.004587815	0.004541543
10	10	1.856461e+11	0.9999574	0.004508996	0.004405252

Obviously, all goodness-of-fit measures show an improvement of the model fit with each iteration. But there is no noticeable improvement after the eighth iteration. All in all, the accuracy of the results are determined by the error tolerance (parameter tolerance) and the number of iterations (parameter iterations) which are stated by the user who has to trade off accuracy against computing time.

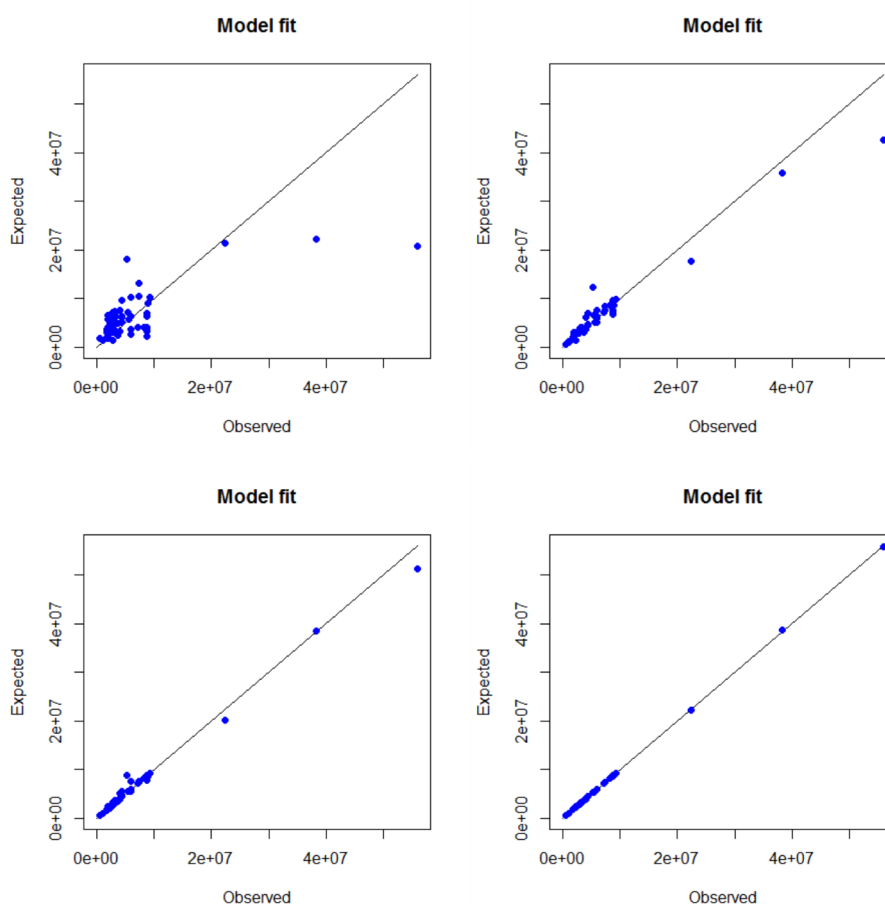


Figure 2: Huff Model fit algorithm.

Conclusions and limitations

Model-based market area analysis for retail and service locations includes 1) theoretical considerations, 2) collection of empirical data regarding the locations and the customers, 3) data processing and, in some cases, data transformation, 4) fitting of the used models and 5) the market area prediction itself. While the first two aspects can not be offered by a statistical software, the presented package *MCI* provides several functions to process the other mentioned steps by implementing the Huff Model and the Multiplicative Competitive Interaction (MCI) Model into R.

The focus of the package is 1) on model fitting via OLS regression (MCI Model) or iterative optimization (Huff Model) and 2) on data preparation, while both aspects are closely related (especially in the MCI Model). As a consequence, some other related substeps are not considered.

Though the package is designed to allow a data exchange with a GIS (e.g. `ijmatrix.crosstab()`) to

prepare an interaction matrix for a map visualization), it does not provide any GIS functions. Thus, the GIS-related steps of market area analysis have to be borrowed from other packages: especially distance or travel time calculations are needed what can be done using GIS-orientated packages like **ggmap** which utilizes the Google API for routing (and was used in the example dataset `shopping2`), **osmar** (Eugster and Schlesinger, 2013) and **osrm** (Giraud, 2016) providing similar functions with respect to OpenStreetMap, or the **huff-tools** package mentioned above.

In econometric market area analyses using the MCI Model, also further diagnostics of the model are recommended, especially with respect to the possible violations of the OLS-related assumptions, such as heteroscedasticity or multicollinearity. A discussion of specific problems and opportunities regarding logarithmic transformations can be found in Silva and Tenreyro (2006) and Manning and Mullahy (2001). The violation of OLS assumptions and possible solutions have been addressed in MCI studies several times (Nakanishi and Cooper, 1974; Kubis and Hartmann, 2007; Tihl and Oruc, 2012; Wieland, 2015a), but there are no corresponding extensions implemented in MCI yet. Other packages may be helpful for this analyses: since the implemented functions return "data.frame" (`mci.transmat()`) and "lm" objects (`mci.fit()`), respectively, this data can be processed e.g. in **car** (Fox and Weisberg, 2011) for further diagnostics.

Another remaining step not yet provided by the presented package is the combination of the MCI Model with the *Geographically Weighted Regression* (GWR) which has been already used to identify spatial nonstationarity in the estimated parameters (Suárez-Vega et al., 2015; Wieland, 2015a). The GWR is implemented in R by the package **spgwr** (Bivand and Yu, 2015).

Acknowledgements

The author would like to thank the two anonymous reviewers for providing useful feedback that helped to improve the paper.

Bibliography

- S. Baecker-Neuchl and H. Wagenseil. Das Ganze sehen: Räumliche Analysen zur Ermittlung des Zusammenhangs zwischen Umsatz- und Nachfragepotenzial. In O. Gansser and B. Krol, editors, *Markt- und Absatzprognosen. Modelle - Methoden - Anwendung*, pages 263–281. Springer-Verlag, 2015. URL https://doi.org/10.1007/978-3-658-04492-3_14. [p300]
- B. R. Berman and J. R. Evans. *Retail Management: A Strategic Approach*. Pearson, 12 edition, 2013. [p298]
- R. Bivand and D. Yu. *Spgwr: Geographically Weighted Regression*, 2015. URL <https://CRAN.R-project.org/package=spgwr>. R package version 0.6-28. [p321]
- G. Cliquet. Retail Location Models. In G. Cliquet, editor, *Geomarketing: Methods and Strategies in Spatial Marketing*, pages 137–163. John Wiley & Sons, 2013. URL <https://doi.org/10.1002/9781118614020.ch6>. [p302, 303]
- P. D. Converse. New laws of retail gravitation. *Journal of Marketing*, 14(3):379–384, 1949. URL <https://doi.org/10.2307/1248191>. [p298]
- L. G. Cooper and M. Nakanishi. Standardizing variables in multiplicative choice models. *Journal of Consumer Research*, 10(1):96–108, 1983. URL <https://doi.org/10.1086/208948>. [p304]
- L. G. Cooper and M. Nakanishi. *Market-Share Analysis: Evaluating Competitive Marketing Effectiveness*. International series in quantitative marketing. First published in 1988 by Kluwer, E-Book edition, 2010. URL http://www.anderson.ucla.edu/faculty/lee.cooper/MCI_Book/BOOKI2010.pdf. [p301, 302]
- M. De Beule, D. Van den Poel, and N. Van de Weghe. An extended Huff-model for robustly benchmarking and predicting retail network performance. *Applied Geography*, 46(1):80–89, 2014. URL <https://doi.org/10.1016/j.apgeog.2013.09.026>. [p300, 301]
- M. J. A. Eugster and T. Schlesinger. Osmar: OpenStreetMap and R. *R Journal*, 5(1):53–63, 2013. URL <https://journal.r-project.org/archive/2013-1/eugster-schlesinger.pdf>. [p321]
- 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>. [p321]
- T. Giraud. *Osmr: Interface Between R and the OpenStreetMap-Based Routing Service OSRM*, 2016. URL <https://CRAN.R-project.org/package=osrm>. R package version 3.0.0. [p321]

- T. Giraud and H. Commenges. *SpatialPosition: Spatial Position Models*, 2016. URL <https://CRAN.R-project.org/package=SpatialPosition>. R package version 1.1.1. [p298]
- O. González-Benito, M. Greatorrex, and P. A. Muñoz-Gallego. Assessment of potential retail segmentation variables: An approach based on a subjective MCI resource allocation model. *Journal of Retailing and Consumer Services*, 7(3):171–179, 2000. URL [https://doi.org/10.1016/S0969-6989\(99\)00026-0](https://doi.org/10.1016/S0969-6989(99)00026-0). [p303]
- GRASS Development Team. *Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.0*. Open Source Geospatial Foundation, 2015. URL <http://grass.osgeo.org>. [p316]
- J. Güssefeldt. Zur Modellierung von räumlichen Kaufkraftströmen in unvollkommenen Märkten. *Erdkunde*, 56(4):351–370, 2002. URL <https://doi.org/10.3112/erdkunde.2002.04.02>. [p299, 300, 301, 316]
- D. L. Huff. *Determination of Intra-Urban Retail Trade Areas*. University of California, 1962. [p299, 300, 303, 308, 309, 312]
- D. L. Huff. A probabilistic analysis of shopping center trade areas. *Land Economics*, 39(1):81–90, 1963. URL <https://doi.org/10.2307/3144521>. [p299]
- D. L. Huff. Defining and estimating a trading area. *Journal of Marketing*, 28(4):34–38, 1964. URL <https://doi.org/10.2307/1249154>. [p299]
- D. L. Huff and R. R. Batsell. Conceptual and operational problems with market share models of consumer spatial behavior. *Advances in Consumer Research*, 2:165–172, 1975. [p299]
- D. L. Huff and B. M. McCallum. Calibrating the Huff Model using ArcGIS Business Analyst. ESRI White Paper, September 2008, 2008. URL <https://www.esri.com/library/whitepapers/pdfs/calibrating-huff-model.pdf>. [p298, 302, 303, 304]
- P. Jia, I. M. Xierali, and F. Wang. Evaluating and re-demarcating the Hospital Service Areas in Florida. *Applied Geography*, 60(4):248–253, 2015. URL <https://doi.org/10.1016/j.apgeog.2014.10.008>. [p298]
- 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>. [p306]
- C. Kanhäußer. Modellierung und Prognose von Marktgebieten am Beispiel des Möbeleinzelhandels. In R. Klein and J. Rauh, editors, *Analysemethodik und Modellierung in der geographischen Handelsforschung*, volume 13 of *Geographische Handelsforschung*, pages 75–109. L.I.S., 2007. [p299, 308]
- R. Klein. *Der Lebensmittel-Einzelhandel im Raum Verden - Räumliches Einkaufsverhalten unter sich wandelnden Bedingungen*, volume 8 of *Flensburger Arbeitspapiere zur Landeskunde und Raumordnung*. Universität Flensburg, 1988. [p300, 301]
- A. Kubis and M. Hartmann. Analysis of location of large-area shopping centres. a probabilistic gravity model for the Halle–Leipzig Area. *Jahrbuch für Regionalwissenschaft*, 27(1):43–57, 2007. URL <https://doi.org/10.1007/s10037-006-0010-3>. [p303, 304, 321]
- R. D. Luce. *Individual Choice Behavior: A Theoretical Analysis*. John Wiley & Sons, 1959. [p299]
- G. Löffler. Market areas - a methodological reflection on their boundaries. *GeoJournal*, 45(4):265–272, 1998. URL <https://doi.org/10.1023/A:1006980420829>. [p298]
- W. G. Manning and J. Mullahy. Estimating log models: To transform or not to transform? *Journal of Health Economics*, 20(4):461–494, 2001. URL [https://doi.org/10.1016/S0167-6296\(01\)00086-8](https://doi.org/10.1016/S0167-6296(01)00086-8). [p308, 321]
- M. Marinov and D. Czamanski. Normative issues in the organization of modern retailers in Israel. *GeoJournal*, 77(3):383–398, 2012. URL <https://doi.org/10.1007/s10708-010-9394-2>. [p300]
- M. Nakanishi and L. G. Cooper. Parameter estimation for a Multiplicative Competitive Interaction Model: Least squares approach. *Journal of Marketing Research*, 11(3):303–311, 1974. URL <https://doi.org/10.2307/3151146>. [p299, 300, 302, 303, 321]
- M. Nakanishi and L. G. Cooper. Simplified estimation procedures for MCI models. *Marketing Science*, 1(3):314–322, 1982. URL <https://doi.org/10.1287/mksc.1.3.314>. [p299, 303]

- T. Orpana and J. Lampinen. Building spatial choice models from aggregate data. *Journal of Regional Science*, 43(2):319–347, 2003. URL <https://doi.org/10.1111/1467-9787.00301>. [p300]
- M. Pavlis, L. Dolega, and A. Singleton. The huff-tools package (version 0.6). Technical Report (PDF), 2014. URL <https://github.com/alexsingleton/Huff-Tools/blob/master/huff-toolsvignette.pdf>. [p298]
- R. C. Perales. *Consumer Choice in Competitive Location Models*. Universitat Pompeu Fabra, 2002. URL <http://www.tdx.cat/bitstream/handle/10803/7330/trcp1de1.pdf>. [p304]
- J. O. Rawlings, S. G. Pantula, and D. A. Dickey. *Applied Regression Analysis*. Springer-Verlag, 2nd edition, 1998. [p303, 308]
- W. J. Reilly. *Methods for the Study of Retail Relationships*, volume 4 of *Studies in Marketing*. Bureau of Business Research, The University of Texas, 1929. [p298]
- T. Reutterer and C. Teller. Store format choice and shopping trip types. *International Journal of Retail & Distribution Management*, 37(8):695–710, 2009. URL <https://doi.org/10.1108/09590550910966196>. [p312]
- J.-P. Rodrigue, C. Comtois, and B. Slack. *The Geography of Transport Systems*. Routledge, 2006. [p298]
- J. M. C. S. Silva and S. Tenreiro. The Log of Gravity. *The Review of Economics and Statistics*, 88(4): 641–658, 2006. URL <https://doi.org/10.1162/rest.88.4.641>. [p308, 321]
- R. Suárez-Vega, J. L. Gutiérrez-Acuña, and M. Rodríguez-Díaz. Locating a supermarket using a locally calibrated Huff model. *International Journal of Geographical Information Science*, 29(2):171–179, 2015. URL <http://doi.org/10.1080/13658816.2014.958154>. [p303, 321]
- B. Tihi and N. Oruc. Competitive location assessment – the MCI approach. *South East European Journal of Economics and Business*, 7(2):379–384, 2012. URL <https://doi.org/10.2478/v10033-012-0013-7>. [p303, 304, 321]
- T. Wieland. *Räumliches Einkaufsverhalten und Standortpolitik im Einzelhandel unter Berücksichtigung von Agglomerationseffekten. Theoretische Erklärungsansätze, modellanalytische Zugänge und eine empirisch-ökonomische Marktgebietsanalyse anhand eines Fallbeispiels aus dem ländlichen Raum Ostwestfalens/Südniedersachsens*, volume 23 of *Geographische Handelsforschung*. MetaGIS, 2015a. [p298, 300, 302, 303, 304, 314, 321]
- T. Wieland. *Nahversorgung im Kontext raumökonomischer Entwicklungen im Lebensmitteleinzelhandel - Konzeption und Durchführung einer GIS-gestützten Analyse der Strukturen des Lebensmitteleinzelhandels und der Nahversorgung in Freiburg im Breisgau*. Technical report, Georg-August-Universität Göttingen, Geographisches Institut, Abteilung Humangeographie, 2015b. URL <http://webdoc.sub.gwdg.de/pub/mon/2015/5-wieland.pdf>. [p316]
- T. Wieland. *MCI: Multiplicative Competitive Interaction (MCI) Model*, 2016. URL <https://CRAN.R-project.org/package=MCI>. R package version 1.3.0. [p298]
- M. Wolf. Anforderungen an Einzelhandelsgutachten. In H. Konze and M. Wolf, editors, *Einzelhandel in Nordrhein-Westfalen planvoll steuern!*, volume 2 of *Arbeitsberichte der ARL [Akademie für Raumforschung und Landesplanung]*, pages 114–134. ARL, 2012. [p298]
- L. Yingru and L. Lin. Assessing the impact of retail location on store performance: A comparison of Wal-Mart and Kmart stores in Cincinnati. *Applied Geography*, 32(2):591–600, 2012. URL <https://doi.org/10.1016/j.apgeog.2011.07.006>. [p300]

Thomas Wieland
 Karlsruhe Institute of Technology
 Institute of Geography and Geoecology
 Kaiserstr. 12, 76131 Karlsruhe
 Germany
thomas.wieland@kit.edu

PSF: Introduction to R Package for Pattern Sequence Based Forecasting Algorithm

by Neeraj Bokde, Gualberto Asencio-Cortés, Francisco Martínez-Álvarez and Kishore Kulat

Abstract This paper introduces the R package that implements the Pattern Sequence based Forecasting (PSF) algorithm, which was developed for univariate time series forecasting. This algorithm has been successfully applied to many different fields. The PSF algorithm consists of two major parts: clustering and prediction. The clustering part includes selection of the optimum number of clusters. It labels time series data with reference to such clusters. The prediction part includes functions like optimum window size selection for specific patterns and prediction of future values with reference to past pattern sequences. The PSF package consists of various functions to implement the PSF algorithm. It also contains a function which automates all other functions to obtain optimized prediction results. The aim of this package is to promote the PSF algorithm and to ease its usage with minimum efforts. This paper describes all the functions in the PSF package with their syntax. It also provides a simple example. Finally, the usefulness of this package is discussed by comparing it to `auto.arima` and `ets`, well-known time series forecasting functions available on CRAN repository.

Introduction

PSF stands for Pattern Sequence Forecasting algorithm. PSF is a successful forecasting technique based on the assumption that there exist pattern sequences in the target time series data. For the first time, it was proposed in Martínez-Álvarez et al. (2008), and an improved version was discussed in Martínez-Álvarez et al. (2011a).

Martínez-Álvarez et al. (2011a) improved the label based forecasting (LBF) algorithm proposed in Martínez-Álvarez et al. (2008) to forecast the electricity price and compared it to other available forecasting algorithms such as ANN (Catalão et al., 2007), ARIMA (Conejo et al., 2005), mixed models (García-Martos et al., 2007) and WNN (Troncoso et al., 2007). These comparisons concluded that the PSF algorithm is able to outperform all these forecasting algorithms, at least in the electricity price/demand context.

Many authors have proposed improvements for PSF. In particular, Jin et al. (2014) highlighted the PSF algorithm limitations and suggested minute modification to minimize the computation delay. They suggested that, instead of using multiple indexes, a single index could make computation simpler and they used the Davies Bouldin index to obtain the optimum number of clusters.

Majidpour et al. (2014) compared PSF to kNN and ARIMA, and observed that PSF can be used for electric vehicle charging energy consumption. It also proposed three modifications in the existing PSF algorithm. First, instead of taking average of all the matched template, it only uses the most recent matched template. Second, if no match was found in the training data (which is possible), MPSF outputs the cluster center of the largest cluster as output. Third, instead of finding the optimum number of clusters starting at two, it starts k from 10% of the total number of samples to avoid degenerate clusters.

Shen et al. (2013) proposed an ensemble of PSF and five variants of the same algorithm, showing better joint performance when applied to electricity-related time series data.

Koprinska et al. (2013) attempted to propose a new algorithm for electricity demand forecast, which is a combination of PSF and Neural networks (NN) algorithms. The results concluded that PSF-NN is performing better than original PSF algorithm.

Fujimoto and Hayashi (2012) modified the clustering method in PSF algorithm. It used a cluster method based on non-negative tensor factorization instead of k-means technique and forecasted energy demand using photovoltaic energy records.

Martínez-Álvarez et al. (2011b) also modified the original PSF algorithm to be used for outlier occurrence forecasting in time series. The metaheuristic searches for motifs or pattern sequences preceding certain data, marked as anomalous in the training set. Then, outlying and regular data are separately processed. Once again, this version was shown to be useful in electricity prices and demand.

The analysis of these works highlights the fact that the PSF algorithm can be applied to many different fields for time series forecasting, outperforming many existing methods. Since the PSF algorithm consists of many dependent functions and the authors did not originally publish their code, the package here described aims at making PSF handy and with minimum efforts for coding.

The rest of the paper is structured as follows. Section 2 provides an overview of the PSF algorithm.

Section 3 explains how the package in R has been developed. Illustrative examples are described in Section 4. Finally, the conclusions drawn from this work are summarized in Section 5.

Brief description of PSF

The PSF algorithm consists of many processes, which can be divided, broadly, in two steps. The first step is clustering of data and the second step is forecasting based on clustered data in earlier step. The block diagram of PSF algorithm shown in Figure 1 was proposed by Martínez-Álvarez et al. (2008). The PSF algorithm is a closed loop process, hence it adds an advantage since it can attempt to predict the future values up to long duration by appending earlier forecasted value to existing original time series data. The block diagram for PSF algorithm shows the close loop feedback characteristics of the algorithm. Although there are various strategies for multiple-step ahead forecasting as described by Bontempi et al. (2013), this strategy turned out to be particularly suitable for electricity prices and demand forecasting, as the original work discussed.

Another interesting feature lies in its ability to simultaneously forecast multiple values, that is, it deals with arbitrary lengths for the horizon of prediction. However, it must be noted that this algorithm is particularly developed to forecast time series exhibiting some patterns in the historical data, such as weather, electricity load or solar radiation, just to mention few examples of usage in reviewed literature. The application of PSF to time series without such kind of inherent patterns might lead to the generation of not particularly competitive results.

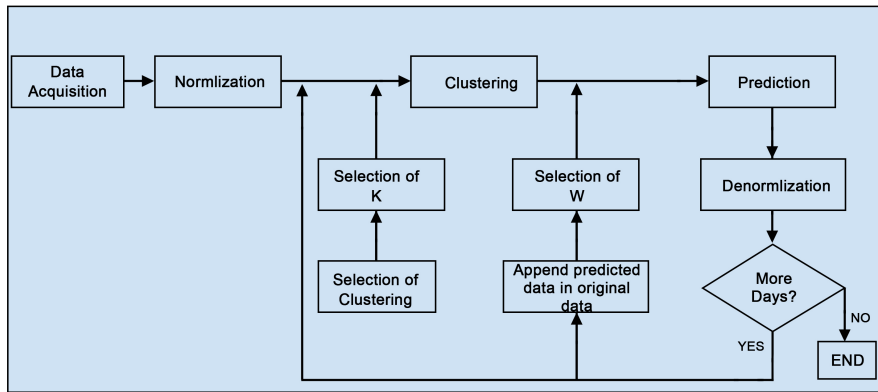


Figure 1: Block diagram of PSF algorithm methodology.

The clustering part consists of various tasks, including data normalization, optimum number of clusters selection and application of k-means clustering itself. The ultimate goal of this step is to discover clusters of time series data and label them accordingly.

Normalization is one of the essential processes in any time series data processing technique. Normalization is used to scale data. The algorithm (Martínez-Álvarez et al., 2011a) used the following transformation to normalize the data.

$$X_j = \frac{X_j}{\frac{1}{N} \sum_{i=1}^N X_i} \tag{1}$$

where X_j is the input time series data and N is the total length of the time series.

However, the original PSF original algorithm used $N = 24$ hours instead of N as the total length of the time series. This presents the problem of knowing all hours for each day to assess the mean. For such reason, the original formula has been replaced in this implementation by the standard feature scaling formula (also called unity-based normalization), which bring all values into the range $[0, 1]$ Dodge (2003):

$$X'_j = \frac{X_j - \min(X_i)}{\max(X_i) - \min(X_i)} \tag{2}$$

where X'_j denotes the normalized value for X_j , and $i = 1, \dots, N$.

The reference articles (Martínez-Álvarez et al., 2008) and (Martínez-Álvarez et al., 2011a) used the k-means clustering technique to assign labels to sets of consecutive values. In the original manuscript, since daily energy was analyzed, clustering was applied to every 24 consecutive values, that is, to

every day. The advantage of k-means is its simplicity, but it requires the number of clusters as input. In Martínez-Álvarez et al. (2008), the Silhouette index was used to decide the optimum numbers of clusters, whereas in the improved version (Martínez-Álvarez et al., 2011a), three different indexes were used, which include the Silhouette index (Kaufman and Rousseeuw, 2008), the Dunn index (Dunn, 1974) and the Davies Bouldin index (Davies and Bouldin, 1979). However, the number of groups suggested by each of these indexes is not necessarily the same. Hence, it was suggested to select the optimum number of clusters by combining more than one index, thus proposing a majority voting system.

As output of the clustering process, the original time series data is converted into a series of labels, which is used as input in the prediction block of the second phase of the PSF algorithm. The prediction technique consists of window size selection, searching for pattern sequences and estimation processes.

Let $x(t)$ be the vector of time series data such that $x(t) = [x_1(t), x_2(t), \dots, x_N(t)]$. After clustering and labeling, the vector converted to $y(t) = [L_1, L_2, \dots, L_N]$, where L_i are labels identifying the cluster centers to which data in vector $x(t)$ belongs to. Note that every $x_i(t)$ can be of arbitrary length and must be adjusted to the pattern sequence existing in every time series. For instance, in the original work, $x(i)$ was composed of 24 values, representing daily patterns.

Then the searching process includes the last W labels from $y(t)$ and it searches for these labels in $y(t)$. If this sequence of last W labels is not found in $y(t)$, then the search process is repeated for last $(W-1)$ labels. In PSF, the length of this label sequence is named as window size. Therefore, the window size can vary from W to 1, although it is not usual that this event occurs. The selection of the optimum window size is very critical and important to make accurate predictions. The optimum window size selection is done in such a way that the forecasting error is minimized during the training process. Mathematically, the error function to be minimized is:

$$\sum_{t \in TS} \|\bar{X}(t) - X(t)\| \tag{3}$$

where $\bar{X}(t)$ are predicted values and $X(t)$ are original values of time series data. In practice, the window size selection is done with cross validation. All possible window sizes are tested on sample data and corresponding prediction errors are compared. The window size with minimum error considered as the optimum window size for prediction.

Once the optimum window size is obtained, the pattern sequence available in the window is searched for in $y(t)$ and the label present just after each discovered sequence is noted in a new vector, called ES. Finally, the future time series value is predicted by averaging the values in vector ES as given below:

$$\bar{X}(t) = \frac{1}{size(ES)} \times \sum_{j=1}^{size(ES)} ES(j) \tag{4}$$

where, $size(ES)$ is the length of vector ES. The procedure of prediction in the PSF algorithm is described in Figure 2. Note that the average is calculated with real values and not with labels.

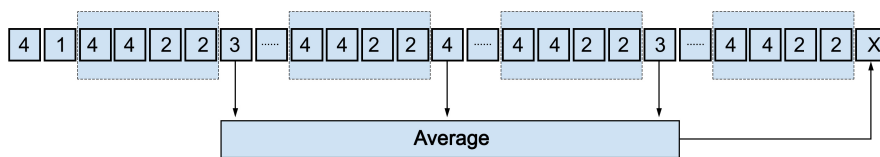


Figure 2: Prediction with PSF algorithm.

The algorithm can predict the future value for next interval of the time series. But, to make it applicable for long term prediction, the predicted short term values will get linked with original data and the whole procedure will be carried out till the desired length of time series prediction is obtained.

PSF package description

This section introduces the PSF package in R language (Bokde et al., 2017). With reference to dependencies, this package imports package `cluster` (Maechler et al., 2017) and suggests packages `knitr` and `rmarkdown`. The package also needs the `data.table` package to improve data wrangling. PSF package consists of various functions. These functions are designed such that these can replace the block diagrams of methodology used in PSF. The block diagram in Figure 3 represents the methodology

mentioned in Figure 1 with the replacement of equivalent functions used in the proposed package.

The various tasks including the optimum cluster size, window size selection, pattern searching and prediction processes are performed in the package with different functions including `psf()`, `predict.psf()`, `plot.psf()`, `optimum_k()`, `optimum_w()`, `psf_predict()` and `convert_datatype()`. All functions in the PSF package version 0.4 onwards are made private except for `psf()`, `predict.psf()` and `plot.psf()` functions, so that users could not use it directly. Moreover, if users need to change or modify clustering techniques or procedures in private functions, the code is available at GitHub (<https://github.com/neerajdhanraj/PSF>).

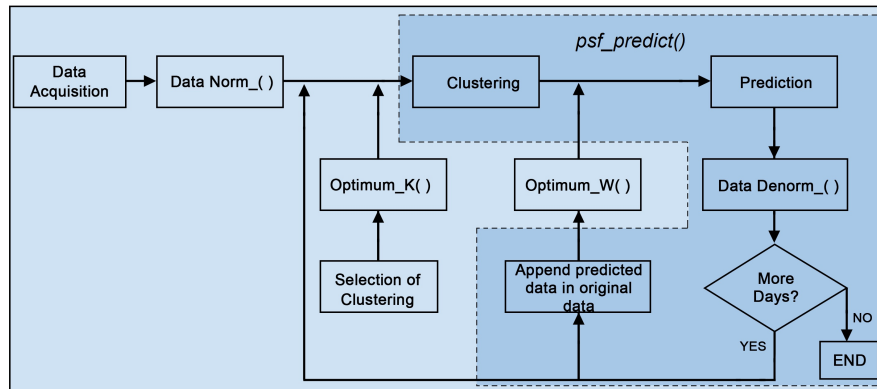


Figure 3: Block diagram with PSF package functions replacement for PSF algorithm.

This section discusses each of these functions and their functionalities along with simple examples. All functions in the PSF package are loaded with:

```
library(PSF)
```

Optimum cluster size selection

As aforementioned, clustering of data is one of the initial phases in PSF. The reference articles (Martínez-Álvarez et al., 2008) and (Martínez-Álvarez et al., 2011a) chose the k-means clustering technique for generating data clusters according to the time series data properties. The limitation of k-means clustering technique is that the adequate number of clusters must be provided by users. Hence, to avoid such situations, the proposed package contains a function `optimum_k()` which calculates the optimum value for the number of clusters, (k), according to the Silhouette index. This function generates the optimum cluster size as output. In Martínez-Álvarez et al. (2011a), multiple indexes (Silhouette index, Dunn index and the Davies–Bouldin index) were considered to determine the optimum number of clusters. For the sake of simplicity and to save the calculation time, only the Silhouette index is considered in this package, as suggested in Martínez-Álvarez et al. (2008).

This `optimum_k()` is a private function which is not directly accessible by users. This function takes data as input time series, in any format be it matrix, data frame, list, time series or vector. Note that the input data type must be strictly numeric. This function returns the optimum value of number of clusters (k) in numeric format.

Optimum window size selection

Once data clustering is performed, the optimum window size needs to be determined. This is an important but tedious and time consuming process, if it is manually done. In Martínez-Álvarez et al. (2008) and Martínez-Álvarez et al. (2011a), the selection of optimum window size is done through cross-validation, in which data is partitioned into two subsets. One subset is used for analysis and the other one for validating the analysis. Since the window size will always be dependent on the pattern of the experimental data, it is necessary to determine the optimum window size for every time series data.

In the PSF package, the optimum window size selection is done with the function `optimum_w()`, which takes as input the time series data, the previous estimated k value, a set candidate w values to search in and the cycle of the input time series. This function estimates the optimum value for the window size such that the error between predicted and actual values is minimum. Internally, this function divides the input time series data into two sub-parts. One of them is the last cycle data

values which will be taken as reference to compare to the predicted values and to calculate RMSE values. The other sub-part is the remaining data set which is used as training part of the data set. The predicted values for the window size with minimum RMSE value is then taken as optimum window size. If more than one window size values are obtained with the same RMSE values, the maximum window size is preferred by the function `optimum_w()`. Like the `optimum_k()` function, the `optimum_w()` function is also a private function and users cannot directly access it.

Prediction with PSF

The PSF package exposes three functions to the user. The first one, `psf()`, can build a PSF model from an univariate time series. The value returned by such function is a S3 object of class 'psf', which contains both original and normalized input time series along with other internal model adjustments. Once the PSF model is trained and returned, the user can invoke the S3 method `predict()` over the the 'psf' object specifying the desired number of forecasted values via the `n.ahead` parameter. This method returns a numeric vector. Finally, the third function exposed is the S3 method `plot()`, that produces a plot including both actual and predicted values from a PSF model and a numeric vector of predictions.

Internally, the prediction procedure is composed of data processing, optimum window size and cluster size selection. The prediction is done with the private function `psf_predict()`, which takes time series data, window size (`w`), cluster size (`k`) and an integer `n.ahead`. The value of `n.ahead` indicates the count up to what extend the forecasting is to be done by function `psf_predict()`. The time series data taken as input can be in any format supported by R language, be it time series, vector, matrix, list or data frame. The PSF package automatically converts it in suitable format using the private function `convert_datatype()` and proceeds further. All other input parameters `w`, `k` and `n.ahead` must be in integer format.

The function `psf_predict()` initiates the process of data normalization. Then, it selects the last `w` labels from the input time series data and searches for that number sequence in the entire data set. Additionally, it captures the very next integer value after each sequence and calculates the average of these values. The obtained averaged value along with denormalization is considered as raw predicted value. If the input parameter `n.ahead` is greater than unity, then the predicted value is appended to the original input data, and the procedure is repeated `n.ahead` times. Finally, the processed data is denormalized and returns a time series which replaces labels by the predicted values.

The `psf_predict()` function is also one of the private functions which takes input the dataset in `data.table` format, as well as another integer inputs including number of clusters (`k`), window size (`w`), horizon of prediction (`n.ahead`), and the `cycle` parameter, which discovers the cycle pattern followed by dataset. This function considers all input parameters and searches for the desired pattern in training data. While performing the searching process, if the desired pattern is obtained once or more times, it calculates the average of very next values for each repeated desired patterns in the whole dataset. Finally, this averaged value is considered as next predicted value and is appended to input time series data.

The `psf()` function uses `optimum_k()` and `optimum_w()` functions. The latter, in turn, uses the described `psf_predict()`. The function `psf()` returns the PSF trained model (S3 object of class 'psf') whose contents are described later in this section. The syntax for `psf()` function is shown below:

```
psf(data, k = seq(2, 10), w = seq(1, 10), cycle = 24)
```

Within the indicated syntax, the parameter `data` is an univariate time series in any format, e.g. time series, vector, matrix, list or data frame. Also, `data` must be strictly provided in numeric format. The parameter `k` is the number of clusters, whereas parameter `w` is the window size. Finally, the `cycle` parameter is the number of values that confirms a cycle in the time series. Usual values for the `cycle` parameter can be 24 hours per day, 12 months per year or so on and it is used only when input data is not in the time series format. If input data is given in time series format, the `cycle` is automatically determined by its internal frequency attribute.

If the user provides a single value for either `k` or `w`, then this value is used in the PSF algorithm and the search for their optimum is skipped. Furthermore, if the user provides a vector of values for either parameter `k` or `w`, then the search for optimum is limited to these values.

This `psf()` function returns a S3 object of class 'psf' which includes the 7 elements described below:

original_series Original time series stored to be used internally to build further plots.

train_data Adapted and normalized internal time series used to train the PSF model.

k The number of clusters optimized and used to train the model.

w The window size optimized and used to train the model.

cycle Determined cycle for the input time series.

dmin Minimum value of the input time series (used to denormalize internally in further predictions).

dmax Maximum value of the input time series (used to denormalize internally in further predictions).

The `psf()` function initiates the conversion of any type of data format into *data.table* which is an internal format for this function (this conversion is carried out by the private function `convert_datatype`). Secondly, it is checked whether the input dataset is multiple of `cycle` parameter or not. If not, it generates a warning to state that the dataset pattern is not suitable for further study. But if the dataset is multiple of the `cycle` value, the normalization of the dataset is carried out and subsequently reshaped according to the cycle of time series.

This reshaped dataset is then provided to `psf_predict()` function with other parameters including cluster size (`k`), window size (`w`), `n.ahead` and `cycle`, as mentioned before. In reply to this, the `psf_predict()` function returns a vector of future predicted values. Since the input dataset had been previously normalized, it is required to denormalize the predicted values. This is carried out by the S3 method `predict.psf()`. This function returns a numeric vector with the denormalized predicted values.

Plot function for PSF

The `plot.psf()` function allows users to plot both the actual values of a series and predicted values obtained by a PSF model. This function takes the trained PSF model, denoted by the first parameter named `x`, which includes internally the original time series, obtained by `psf()`, along with the predicted values obtained through `predict.psf()`, and optionally other plot describing variables, like plot title, legends, etc. This function generates the plot showing original time series data and predicted data with significant color changes. The plot obtained by `plot.psf()` possesses dynamic margin size such that it can include the input data set and all predicted values as per requirements. The syntax of `plot.psf()` function is as shown below. The usage of `plot.psf()` function is further discussed in the next section.

```
plot.psf(x, predictions, cycle = 24, ...)
```

Example

This section presents the examples to introduce the use of the PSF package and to compare it with `auto.arima()` and `ets()` functions, which are well accepted functions in the R community working over time series forecasting techniques. The data used in this example are `nottem` and `sunspots` which are standard time series datasets available in R. The `nottem` dataset is the average air temperatures at Nottingham Castle in degrees Fahrenheit, collected for 20 years, on monthly basis. Similarly, `sunspots` dataset is mean relative sunspot numbers from 1749 to 1983, measured on monthly basis.

For both datasets, all the recorded values except for the final year are considered as training data, and the last year is used for testing purposes. The predicted values for final year for both datasets are discussed in this section.

In the following examples, model training, forecasting and plotting were shown for the dataset `nottem`, but the procedure is the same for the dataset `sunspots`. In first place, the model must be trained using the PSF package. as is shown below.

```
library(PSF)
nottem_model <- psf(nottem)
nottem_model

## $original_series
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1920 40.6 40.8 44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8
## 1921 44.2 39.8 45.1 47.0 54.1 58.7 66.3 59.9 57.0 54.2 39.7 42.8
## ...
##
## $train_data
##           V1          V2          V3          V4          V5          V6          V7
## 1: 0.26420455 0.2698864 0.3721591 0.4375000 0.6477273 0.7727273 0.7500000
## 2: 0.36647727 0.2414773 0.3920455 0.4460227 0.6477273 0.7784091 0.9943182
## ...
```

```
##
## $k
## [1] 2
##
## $w
## [1] 1
##
## $cycle
## [1] 12
##
## $dmin
## [1] 31.3
##
## $dmax
## [1] 66.5
##
## attr(,"class")
## [1] "psf"
```

Once the model is trained, forecasted values for the time series can be obtained using the S3 method `predict()` function, as is shown below.

```
nottem_preds <- predict(nottem_model, n.ahead = 12)
nottem_preds

## [1] 38.97692 38.71538 42.49231 46.32308 52.91538 57.97692 61.87692
## [8] 60.19231 57.03846 49.42308 43.23846 40.21538
```

To represent the prediction performance in plot format, the S3 method `plot()` can be used, as shown in the following code.

```
plot(nottem_model, nottem_preds)
```

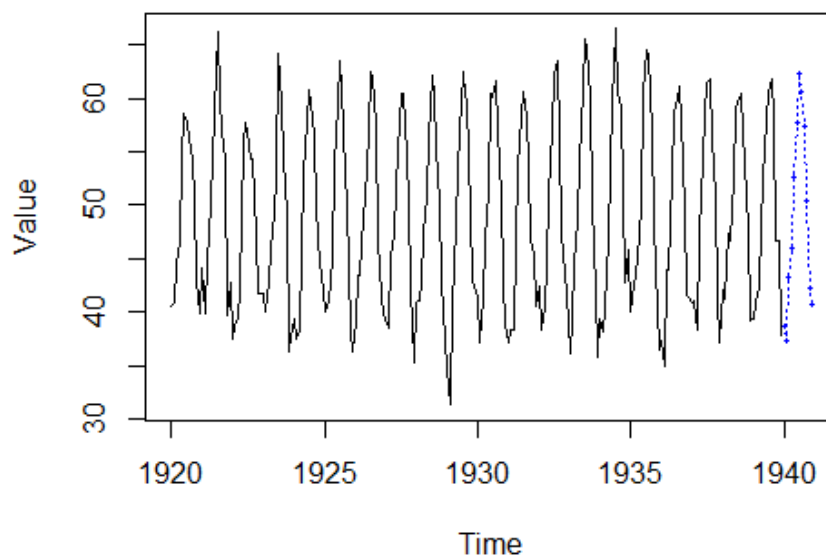


Figure 4: Plot showing forecasted results with function `plot.psf()` for nottem dataset.

Figures 4 and 5 show the prediction with the PSF algorithm for nottem and sunspots datasets, respectively. Such results are compared to those of `auto.arima()` and `ets()` functions from **forecast** R package (Hyndman et al., 2017). `auto.arima()` function compares either AIC, AICc or BIC value and suggests best ARIMA or SARIMA models for a given time series data. Analogously, the `ets()`

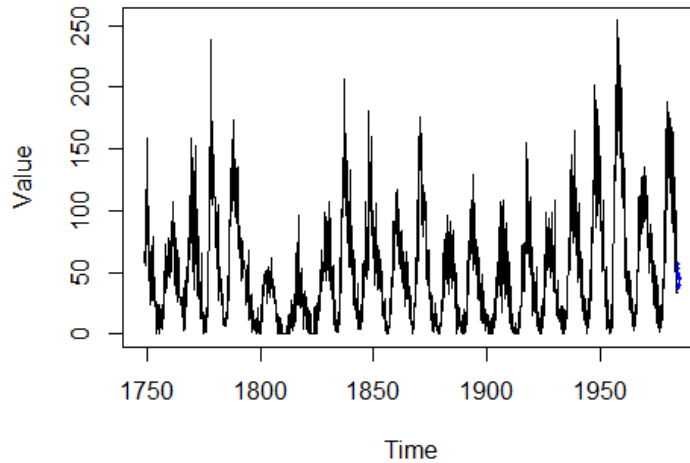


Figure 5: Plot showing forecasted results with function `plot.psf()` for sunspots dataset.

Functions	<code>psf()</code>	<code>auto.arima()</code>	<code>ets()</code>
RMSE	2.077547	2.340092	32.78256

Table 1: Comparison of time series prediction methods with respect to RMSE values for nottem dataset.

function considers the best exponential smoothing state space model automatically and predicts future values.

Tables 1 and 2 show comparisons for `psf()`, `auto.arima()` and `ets()` functions when using the Root Mean Square Error (RMSE) parameter as metric, for nottem and sunspots datasets, respectively. In order to avail more accurate and robust comparison results, error values are calculated for 10 times and the mean value of error values for methods under comparison are shown in Tables 1 and 2. These tables clearly state that `psf()` function is able to outperform the comparative time series prediction methods.

Additionally, the reader might want to refer to the results published in the original work [Martínez-Álvarez et al. \(2011a\)](#), in which it was shown that PSF outperformed many different methods when applied to electricity prices and demand forecasting.

Conclusions

This article is a thorough description of the PSF R package. This package is introduced to promote the algorithm Pattern Sequence based Forecasting (PSF) which was proposed by [Martínez-Álvarez et al. \(2008\)](#) and then modified and improved by [Martínez-Álvarez et al. \(2011a\)](#). The functions involved in the PSF package can be tested with time series data in any format like vector, time series, list, matrix or data frame. The aim of the PSF package is to simplify the calculations and to automate the steps involved in prediction while using PSF algorithm. Illustrative examples and comparisons to other methods are also provided.

Functions	<code>psf()</code>	<code>auto.arima()</code>	<code>ets()</code>
RMSE	22.11279	41.14366	52.29985

Table 2: Comparison of time series prediction methods with respect to RMSE values for sunspots dataset.

Acknowledgements

We would like to thank our colleague, Mr. Sudhir K. Mishra, for his valuable comments and suggestions to improve this article. Thanks also goes to CRAN maintainers for needful comments on the submitted package. Finally, this study has been partially funded by the Spanish Ministry of Economy and Competitiveness and by the Junta de Andalucía under projects TIN2014-55894-C2-R and P12-TIC-1728, respectively.

Bibliography

- N. Bokde, G. Asencio-Cortes, and F. Martínez-Alvarez. *PSF: Forecasting of Univariate Time Series Using the Pattern Sequence-Based Forecasting (PSF) Algorithm*, 2017. URL <https://CRAN.R-project.org/package=PSF>. R package version 0.4. [p326]
- G. Bontempi, S. B. Taieb, and Y. Le-Borgne. Machine learning strategies for time series forecasting. *Lecture Notes in Business Information Processing*, 138:62–77, 2013. URL https://doi.org/10.1007/978-3-642-36318-4_3. [p325]
- J. Catalão, S. Mariano, V. Mendes, and L. Ferreira. Short-term electricity prices forecasting in a competitive market: a neural network approach. *Electric Power Systems Research*, 77(10):1297–1304, 2007. URL <https://doi.org/10.1016/j.epsr.2006.09.022>. [p324]
- A. J. Conejo, M. A. Plazas, R. Espínola, and A. B. Molina. Day-Ahead Electricity Price Forecasting Using the Wavelet Transform and ARIMA Models. *Power Systems, IEEE Transactions on*, 20(2):1035–1042, 2005. URL <https://doi.org/10.1109/tpwrs.2005.846054>. [p324]
- D. L. Davies and D. W. Bouldin. A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2:224–227, 1979. URL <https://doi.org/10.1109/tpami.1979.4766909>. [p326]
- Y. Dodge. *The Oxford Dictionary of Statistical Terms*. The International Statistical Institute, 2003. URL <https://doi.org/10.1002/bimj.200410024>. [p325]
- J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974. URL <https://doi.org/10.1080/01969727408546059>. [p326]
- Y. Fujimoto and Y. Hayashi. Pattern sequence-based energy demand forecast using photovoltaic energy records. In *Renewable Energy Research and Applications (ICRERA), 2012 International Conference on*, pages 1–6. IEEE, 2012. URL <https://doi.org/10.1109/icrera.2012.6477299>. [p324]
- C. García-Martos, J. Rodríguez, and M. J. Sánchez. Mixed models for short-run forecasting of electricity prices: Application for the spanish market. *Power Systems, IEEE Transactions on*, 22(2):544–552, 2007. URL <https://doi.org/10.1109/tpwrs.2007.894857>. [p324]
- R. Hyndman, M. O’Hara-Wild, C. Bergmeir, S. Razbash, and E. Wang. *Forecast: Forecasting Functions for Time Series and Linear Models*, 2017. URL <https://CRAN.R-project.org/package=forecast>. R package version 8.0. [p330]
- C. H. Jin, G. Pok, H.-W. Park, and K. H. Ryu. Improved pattern sequence-based forecasting method for electricity load. *IEEE Transactions on Electrical and Electronic Engineering*, 9(6):670–674, 2014. URL <https://doi.org/10.1002/tee.22024>. [p324]
- L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 2008. URL <https://doi.org/10.2307/2290430>. [p326]
- I. Koprinska, M. Rana, A. Troncoso, and F. Martínez-Álvarez. Combining pattern sequence similarity with neural networks for forecasting electricity demand time series. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 940–947. IEEE, 2013. URL <https://doi.org/10.1109/ijcnn.2013.6706838>. [p324]
- M. Maechler, P. Rousseeuw, A. Struyf, and M. Hubert. *Cluster: “Finding Groups in Data”: Cluster Analysis Extended Rousseeuw et al.*, 2017. URL <https://CRAN.R-project.org/package=cluster>. R package version 2.0.6. [p326]
- M. Majidpour, C. Qiu, P. Chu, R. Gadh, and H. R. Pota. Modified pattern sequence-based forecasting for electric vehicle charging stations. In *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, pages 710–715. IEEE, 2014. URL <https://doi.org/10.1109/smartgridcomm.2014.7007731>. [p324]

- F. Martínez-Álvarez, A. Troncoso, J. C. Riquelme, and J. S. Aguilar-Ruiz. LBF: A Labeled-Based Forecasting Algorithm and Its Application to Electricity Price Time Series. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 453–461. IEEE, 2008. URL <https://doi.org/10.1109/icdm.2008.129>. [p324, 325, 326, 327, 331]
- F. Martínez-Álvarez, A. Troncoso, J. C. Riquelme, and J. S. Aguilar-Ruiz. Energy time series forecasting based on pattern sequence similarity. *Knowledge and Data Engineering, IEEE Transactions on*, 23(8): 1230–1243, 2011a. URL <https://doi.org/10.1109/tkde.2010.227>. [p324, 325, 326, 327, 331]
- F. Martínez-Álvarez, A. Troncoso, J. C. Riquelme, and J. S. Aguilar-Ruiz. Discovery of motifs to forecast outlier occurrence in time series. *Pattern Recognition Letters*, 32:1652–1665, 2011b. URL <https://doi.org/10.1016/j.patrec.2011.05.002>. [p324]
- W. Shen, V. Babushkin, Z. Aung, and W. L. Woon. An ensemble model for day-ahead electricity demand time series forecasting. In *Future Energy Systems, 2013 ACM International Conference on*, pages 51–62, 2013. URL <https://doi.org/10.1145/2487166.2487173>. [p324]
- A. Troncoso, J. M. R. Santos, A. G. Expósito, J. L. M. Ramos, and J. C. R. Santos. Electricity market price forecasting based on weighted nearest neighbors techniques. *Power Systems, IEEE Transactions on*, 22(3):1294–1301, 2007. URL <https://doi.org/10.1109/tpwrs.2007.901670>. [p324]

Neeraj Bokde

Visvesvaraya National Institute of Technology, Nagpur
North Ambazari Road, Nagpur

India

neeraj.bokde@students.vnit.ac.in

Gualberto Asencio-Cortés

Universidad Pablo de Olavide

ES-41013, Sevilla

Spain

guaasecor@upo.es

Francisco Martínez-Álvarez

Universidad Pablo de Olavide

ES-41013, Sevilla

Spain

fmaralv@upo.es

Kishore Kulat

Visvesvaraya National Institute of Technology, Nagpur

North Ambazari Road, Nagpur

India

kdkulat@ece.vnit.ac.in

flan: An R Package for Inference on Mutation Models.

by Adrien Mazoyer, Rémy Drouilhet, Stéphane Despréaux, and Bernard Ycart

Abstract This paper describes **flan**, a package providing tools for fluctuation analysis of mutant cell counts. It includes functions dedicated to the distribution of final numbers of mutant cells. Parametric estimation and hypothesis testing are also implemented, enabling inference on different sorts of data with several possible methods. An overview of the subject is proposed. The general form of mutation models is described, including the classical models as particular cases. Estimating from a model, when the data have been generated by another, induces different possible biases, which are identified and discussed. The three estimation methods available in the package are described, and their mean squared errors are compared. Finally, implementation is discussed, and a few examples of usage on real data sets are given.

Introduction

Mutation models are probabilistic descriptions of the growth of a population of cells, where mutations occur randomly during the process. Data are samples of integers, interpreted as final numbers of mutant cells. These numbers may be coupled with final numbers of cells (mutant and non mutant). The frequent appearance in the data of very large mutant counts, usually called “jackpots”, evidences heavy-tailed probability distributions. The parameter of interest is the mutation probability for a mutant cell to appear upon any given cell division, denoted by π . In practice, π is typically of order 10^{-9} – 10^{-11} . Computing robust estimates for π is of crucial importance in medical applications, like cancer tumor relapse or multidrug resistance of *Mycobacterium Tuberculosis* for instance.

Any mutation model can be interpreted as the result of the three following ingredients:

- a random number of mutations occurring with small probability among a large number of cell divisions. Due to the law of small numbers, the number of mutations approximately follows a Poisson distribution. The expectation of that distribution, denoted by α , is the product of the mutation probability π with the total number of divisions;
- from each mutation, a clone of mutant cells growing for a random time. Due to exponential growth, most mutations occur close to the end of the experiment, and the developing time of a random clone has exponential distribution. The rate of that distribution, denoted by ρ , is the relative fitness, i.e. the ratio of the growth rate of normal cells to that of mutants;
- the number of mutant cells that any clone developing for a given time will produce. The distribution of this number depends on the distribution of division times of mutants.

Using the theory of continuous time branching processes (Bellman and Harris, 1952; Athreya and Ney, 1972), and under specific modeling assumptions, it can be proved that the asymptotic distribution of the final number of mutants has an explicit form. A first mutation model with explicit distribution is the well known Luria-Delbrück model (Luria and Delbrück, 1943). Other mathematical models were introduced by Lea and Coulson (1949), followed by Armitage (1952) and Bartlett (1978). In these models, division times of mutant cells were supposed to be exponentially distributed. Thus a clone develops according to a Yule process, and its size at a given time follows a geometric distribution. The distribution of final mutant counts is also explicit when division times are supposed to be constant. This latter model is called Haldane model by Sarkar (1991); an explicit form of the asymptotic distribution is given in Ycart (2013). General division times have been studied by Ycart (2013), but no explicit distribution is available apart from the exponential and constant division times.

The first estimation method was given by Luria and Delbrück (1943). It is based on the simple relation between the probability of null counts in the sample, and the mutation probability, and it is called P0 method. Of course, if the sample does not contain null counts, the method cannot be applied. Apart from the P0 method, all other methods couple the estimation of π or α , with the estimation of ρ . When the distribution of final numbers has an explicit form, the Maximum Likelihood (ML) is an obvious optimal choice (Ma et al., 1992; Zheng, 2005). However, because of the jackpots, likelihood computation can be numerically unstable. There are several ways to reduce tail effects (Wilcox, 2012, Sec. 2.2), among which “Winsorization” consists in truncating the sample beyond some maximal value. Another estimation method (GF) uses the probability generating function (PGF) (Rémillard and Theodorescu, 2000; Hamon and Ycart, 2012). The estimators of α and ρ obtained with the GF method proved to be close to optimal efficiency, with a broad range of calculability, a good numerical stability, and a negligible computing time. For the three methods, P0, ML, and GF, the estimators of α and

ρ are asymptotically normal. Thus confidence intervals and p-values for hypothesis testing can be computed, for one sample and two sample tests.

The problem with classical mutation models, is that they are based on quite unrealistic assumptions: constant final number of cells (Angerer, 2001a; Komarova et al., 2007; Ycart and Veziris, 2014), no cell deaths (Angerer (2001a, Sec. 3.1); Dewanji et al. (2005); Komarova et al. (2007); Ycart (2014)), fully efficient plating (Stewart et al., 1990; Stewart, 1991; Angerer, 2001b), or, as mentioned above, exponential distribution of division times. Using a model for estimation, when the data have been generated by another one, necessarily induces a bias on estimates. For instance, if cell deaths are neglected, mutation probability will be underestimated.

Several informatic tools have already been developed for fluctuation analysis (Zheng, 2002; Hall et al., 2009; Gillet-Markowska et al., 2015). These tools are quite user-friendly. However, they do not take into account all the possible model assumptions mentioned above. The `flan` package described here, is dedicated to mutation models, and parameter estimation with the three methods P0, ML, and GF. It includes a set of functions for the distribution of mutant cell counts (`df1an`, `pf1an`, `qf1an`, `rf1an`) and a graphic function (`draw.clone`). They treat general models, with fluctuating final numbers, cell deaths, and other division time distributions than exponential and constant. The general estimation function is `mutestim`. It returns estimates for the parameters α , π and ρ , with the three estimation methods, constant or exponential division times, and cell deaths. As a wrapper, a hypothesis testing function (`flan.test`) is provided. In order to make the package user-friendly, the functions have been designed to resemble classical R functions, like `t.test` or `rnorm`.

The paper is organized as follows. Section L.2 is devoted to the probabilistic setting: the hypotheses of the different models are described, and the asymptotic results are explained. In Section L.3, the three estimation methods are exposed, and the biases described above are discussed. A comparison of the three methods in terms of mean squared errors is provided. The user interface and the `Rcpp` (Eddelbuettel, 2013) implementation is treated in Section L.4; examples of execution are shown in Section L.5.

Mutation models

In this section, probabilistic mutation models are described. The basic modeling hypotheses are the following:

- at time 0 a homogeneous culture of n_0 normal cells is given;
- the lifetime of any normal cell is a random variable with distribution function F ;
- upon completion of the generation time of a normal cell:
 - with probability π one normal and one mutant cell are produced;
 - with probability $1 - \pi$ two normal cells are produced;
- the lifetime of any mutant cell is a random variable with distribution function G ;
- upon completion of the lifetime of a mutant cell:
 - with probability δ the cell dies out;
 - with probability $1 - \delta$ two mutant cells are produced;
- all random variables and events (division times, mutations, and deaths) are mutually independent.

Consider that the initial number n_0 tends to infinity, the mutation probability $\pi = \pi_{n_0}$ tends to 0, and the time $t = t_{n_0}$ at which mutants are counted tends to infinity. The scale of time is supposed to be adjusted so that the exponential growth rate of mutants is 1; thus the exponential growth rate of normal cells is ρ . See Athreya and Ney (1972, Chap. IV Sec. 4) or Hamon and Ycart (2012) for the definition of the growth rate (also called “Malthusian parameter”). The expected number of mutations before t_{n_0} is proportional to $n_0 \pi_{n_0} e^{\rho t_{n_0}}$, and the asymptotics are assumed to be such that this number converges as n_0 tends to infinity to α , positive and finite.

Under the above hypotheses, as n_0 tends to $+\infty$, the final number of mutants converges in law to the distribution with PGF:

$$g(z) = \exp(-\alpha(1 - h(z))), \quad (1)$$

with

$$h(z) = \int_0^\infty \psi(z, t) \rho e^{-\rho t} dt, \quad (2)$$

where $\psi(z, t)$ is the PGF of the number of cells at time t in a mutant clone, starting from a single cell at time 0. Observe that it depends on the lifetime distribution of normal cells F only through ρ . The

above result is deduced from the theory of continuous time branching processes (Hamon and Ycart, 2012)). The expressions (1) and (2) translate the three ingredients described in the introduction:

1. the Poisson distribution with intensity α models the total number of mutations which occur during the process;
2. the exponential distribution with rate ρ is that of the time during which a random clone develops;
3. the distribution with PGF $\psi(\cdot, t)$ is that of the number of cells in a random clone developing during a time interval of length t . The PGF ψ is the solution of a Bellman-Harris equation (Bellman and Harris, 1952) in terms of δ and G .

Hence the expressions of h as an exponential mixture, and of g as a Poisson compound. In practice, the plating process can be less than 100% efficient. In that case, a random number of mutants will not be counted: if only a proportion ζ of the final population is plated, then each cell will be observed with probability ζ . Denote by M_{tot} and M the total and the observed numbers of mutants. Given $M_{\text{tot}} = m$, M follows the binomial distribution with parameters m and ζ . Thus, the PGF g of M is given by:

$$g(z) = \mathbb{E} \left[\mathbb{E} \left[z^M \mid M_{\text{tot}} \right] \right] = \exp(-\alpha(1 - h(1 - \zeta + \zeta z))) . \tag{3}$$

The PGF (3) defines a parametrized family of distributions, denoted hereafter by $MM(\alpha, \rho, \delta, \zeta, G)$ (Mutation Model). This is a family of heavy-tailed distributions, with tail exponent ρ : the higher the fitness, the heavier the tail. This directly influences the number and the amount of jackpots.

At this point, the PGF ψ can be given as an explicit expression only for two particular lifetime distributions of mutants: exponential, and Dirac (constant lifetimes). The corresponding mutation models will be denoted respectively by $LD(\alpha, \rho, \delta, \zeta)$ (Luria-Delbrück), and $H(\alpha, \rho, \delta, \zeta)$ (Haldane). The functions `df1an`, `pf1an`, and `qf1an` compute densities, probabilities, quantiles of LD and H distributions (with $\zeta = 1$).

Assuming that a consistent estimator of α has been defined, the problem in practice is to compute reliable estimates of the mutation probability, π . The simplest approach assumes that the final number of cells, denoted by N , is constant. An estimate of π is then obtained by dividing the estimate of α by N . However, even under close experimental monitoring, assuming that the final number of cells is a constant is quite unrealistic. Thus, N must be viewed as a random variable with a certain probability distribution function K on $[0, +\infty)$. By analogy with (1), the conditional PGF of the number of mutants given $N = n$, can be given by the following expression:

$$g(z \mid N = n) = \exp(-\pi n(1 - h(z))) .$$

Or else, the conditional distribution of the number of mutants given $N = n$ is the distribution $MM(\pi n, \rho, \delta, \zeta, G)$. The distribution function K is supposed to be known and its Laplace transform is denoted by \mathcal{L} :

$$\mathcal{L}(z) = \mathbb{E} \left[e^{-zN} \right] = \int_0^\infty e^{-zn} dK(n) ,$$

Thus the PGF of the final number of mutants is given by:

$$g(z) = \int_0^\infty g(z \mid N = n) dK(n) = \mathcal{L}(\pi(1 - h(z))) . \tag{4}$$

Remark that if N is constant, (4) reduces to (1) with $\alpha = \pi N$. In general, the PGF (4) defines a new parametrized family of mutation distributions, denoted hereafter by $MMFN(\pi, \rho, \delta, \zeta, G, K)$ (Mutation Models with Fluctuating Numbers of cells).

The two particular cases for the distribution G_X previously mentioned above (exponential and Dirac) will be denoted by $LDFN(\alpha, \rho, \delta, \zeta, K)$ (Luria-Delbrück with Fluctuating Number of cells) and $HFN(\alpha, \rho, \delta, \zeta, K)$ (Haldane with Fluctuating Number of cells). As will be shown in Section L.3, estimating π by the ratio of an estimate of α by the expectation of N induces a negative bias.

The function `rfl1an` outputs samples of pairs (mutant counts–final counts) following $MMFN$ distributions where G is an exponential, Dirac, log-normal or gamma distribution, and K is a log-normal or Dirac distribution.

Statistical inference

Here the three estimation methods P0, ML and GF are described. The main features and the limitations of each method are discussed. The three methods compute estimates of α and ρ , under the *LD* and *H* models. When couples (mutant counts–final numbers) are given, estimates of π and ρ are calculated under the *LDFN* or *HFN* models.

Even if the probabilities and their derivatives with respect to δ for *LD* and *H* distributions can be computed, the variations of the whole distribution as a function of δ are too small to enable estimation in practice (see [Ycart \(2014\)](#) for more details). Thus, the parameter δ is supposed to be known for the three methods.

In the rest of this section, the three estimators are described, their performances compared in terms of MSE, and the possible sources of biases discussed.

Estimators

P0 estimator: The first method was introduced by [Luria and Delbrück \(1943\)](#) when $\delta = 0$. In that case, the probability of null counts in the sample is $e^{-\alpha}$. Hence α can be estimated taking the negative logarithm of the relative frequency of zeros among mutant counts. Hence the method cannot be applied if the sample does not contain null counts.

If $\delta > 0$, the probability of null counts in the sample depends also on δ . Assuming $\delta < 1/2$, a fixed point of the PGF $\psi(\cdot, t)$ is the extinction probability of a mutant clone ([Athreya and Ney, 1972](#), Theorem 1, Chap.I):

$$\delta_* = \frac{\delta}{1 - \delta}.$$

By definition, δ_* is also a fixed point of the PGF (2). Then the probability of null counts in the sample is $e^{-\alpha(1-\delta_*)}$. A consistent and asymptotically normal estimator of α is given by:

$$\hat{\alpha}_0 = \frac{-\log(\hat{g}(\delta_*))}{1 - \delta_*}, \tag{5}$$

where \hat{g} denotes the empirical PGF of the final number of mutants.

Consider now that $\zeta < 1$. Ignoring the inefficient plating will induce a negative bias. A correction has been proposed by [Stewart et al. \(1990, eq. \(41\)\)](#). However, it can be used only under model *LD*($m, 1, 0$). Indeed, the general expression of the probability of null counts is $e^{-\alpha(1-h(1-\zeta))}$, which depends on the fitness ρ . It is still possible to extend the estimator (5) to the case where $\zeta < 1$:

$$\hat{\alpha}_0 = \frac{-\log\left(\hat{g}\left(\delta_*^{(\zeta)}\right)\right)}{1 - \delta_*}, \tag{6}$$

with

$$\delta_*^{(\zeta)} = \frac{\delta_* - (1 - \zeta)}{\zeta}.$$

We remark that (6) makes sense only if $|\delta_*^{(\zeta)}| \leq 1$. In particular, if $\delta = 0$, the plating efficiency ζ has to be greater than 0.5. Therefore, the

Notice that the P0 method does not directly yield an estimator of ρ . If an estimate is desired, the ML method can be used for ρ only, setting $\alpha = \hat{\alpha}_0$.

ML estimators: Since algorithms ([Embrechts and Hawkes, 1982](#); [Zheng, 2005](#); [Hamon and Ycart, 2012](#); [Ycart and Veziris, 2014](#)) enable the computation of the probabilities of the *LD* and *H* models, the ML method seems to be an obvious choice. It can be used on two kinds of samples:

1. sample of mutant counts: In that case, the likelihood is computed with the probabilities of the model *LD* or *H*. The parameter of interest is α .
2. sample of pairs of (mutant counts–final numbers): In that case, the likelihood is computed with the probabilities of the model *LDFN* or *HFN*. The parameter of interest is π .

In both cases, ρ can also be estimated.

However, when the sample maximum is large, sums of products of small terms must be computed ([Hamon and Ycart, 2012](#)). The procedure can be very long and numerically unstable. Thus, the ML estimators can fail for large α and small ρ . In practice, this instability problem is avoided using Winsorization ([Wilcox, 2012, Sec. 2.2](#)), which consists in replacing any value of the sample that exceeds a certain bound by the bound itself. The bound is 1024 by default, and it could be necessary to increase

it. All information above the bound is lost, and in an extreme case where the sample minimum is greater than the bound, irrelevant results will be returned.

In theory, it is also possible to be explicit about the probabilities of the *LD* model when $\zeta < 1$. However, these computations have not been done for the *H* model. Thus the plating process is assumed to be fully efficient when the ML method is used.

GF estimators: The GF method uses the PGF to estimate the parameter of a compound Poisson distribution (Rémillard and Theodorescu, 2000; Hamon and Ycart, 2012). Let $0 < z_1 < z_2 < 1$ and z_3 in $(0; 1)$. The estimators of α and ρ are the following:

$$\hat{\alpha}_{GF}(z_3) = \frac{\log(\hat{g}(z_3))}{h_{\hat{\rho}_{GF}(z_1, z_2)}(z_3) - 1} \quad \text{and} \quad \hat{\rho}_{GF}(z_1, z_2) = f_{z_1, z_2}^{-1}(\hat{y}),$$

where \hat{g} denotes the empirical PGF of the final number of mutants, h_x is the PGF (2) with $\rho = x$, and:

$$f_{z_1, z_2}(x) = \frac{h_x(z_1) - 1}{h_x(z_2) - 1} \quad \text{and} \quad \hat{y} = \frac{\log(\hat{g}(z_1))}{\log(\hat{g}(z_2))}. \tag{7}$$

From Rémillard and Theodorescu (2000), it can be proved that the couple of estimators $(\hat{\alpha}_{GF}, \hat{\rho}_{GF})$ is strongly consistent and asymptotically normal, with explicit asymptotic variance (Hamon and Ycart, 2012).

The GF estimators depend on the three arbitrary values of z_1, z_2, z_3 . Those tuning parameters are set to $z_1 = 0.1, z_2 = 0.9$, and $z_3 = 0.8$. For more details about the choice of those values, see Hamon and Ycart (2012).

In practice, the GF estimators are quite comparable in precision to ML estimators, with a much broader range of calculability, a better numerical stability, and a negligible computing time, even in the case where the ML method fails. For that reason, we have chosen to initialize the ML optimization by GF estimates, to improve both numerical stability and computing time.

The only practical limitation of this method is the following. A zero of the monotone function $f_{z_1, z_2}(\rho) - \hat{y}$ must be computed. An upper bound for the domain of research must be given, which can be a problem if the sample does not contain jackpots. However in that case, a mutation model is not adapted.

According to (3), the GF estimators of α and ρ can be extended to the case where $\zeta \leq 1$ as follows:

$$\hat{\alpha}_{GF}(z_3) = \frac{\log(\hat{g}(z_3))}{h_{\hat{\rho}_{GF}(z_1, z_2)}(1 - \zeta + \zeta z_3) - 1} \quad \text{and} \quad \hat{\rho}_{GF}(z_1, z_2) = f_{1 - \zeta + \zeta z_1, 1 - \zeta + \zeta z_2}^{-1}(\hat{y}),$$

where f_{z_1, z_2} and \hat{y} are still given by (7). By the same reasoning as Hamon and Ycart (2012), strong consistency and asymptotic normality of the couple $(\hat{\alpha}_{GF}, \hat{\rho}_{GF})$ can be proved.

The function `mutestim` computes estimates and their respective standard deviations for α, π and ρ according to the type of input. Moreover, the estimators mentioned here are asymptotically normal. Thus, one and two sample tests can be performed, using the function `flan.test`. The null hypothesis will be either fixed theoretical values of α, π, ρ in the one sample case, or a difference of the same in the two sample case.

Comparison of the three estimators

The Figures 1 and 2 (drawn using `ggplot2`) show “maps of usage” of the estimation methods under the *LD* and *H* models. The methods are compared in terms of the relative MSE of $(\hat{\alpha}, \hat{\rho})$ defined as:

$$\sqrt{\left(1 - \frac{\hat{\alpha}}{\alpha}\right)^2 + \left(1 - \frac{\hat{\rho}}{\rho}\right)^2}. \tag{8}$$

The RGB code is used: red for GF, green for P0, blue for ML. Twenty values of α between 0.5 and 10, and as many values of ρ from 0.2 to 5, were chosen. Thus 400 couples were considered. For each of them, the following procedure was applied:

1. draw 10^4 samples of size 100 of the $LD(\alpha, \rho, 0, 1)$;
2. for each sample, compute ML, GF and P0 estimates of (α, ρ) under *LD* model;
3. from the 10^4 estimates, compute the relative MSEs of each method;
4. assign a RGB color according to the MSEs. For each method:
 - if the MSE is less than 0.05, assign 1 to the corresponding RGB component;

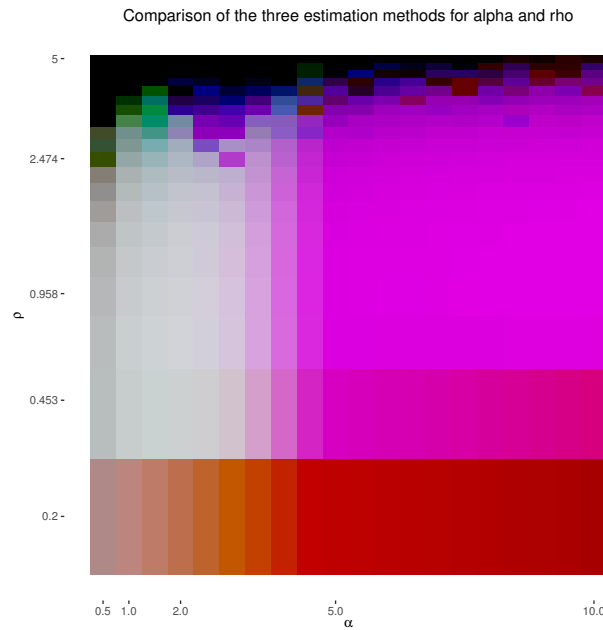


Figure 1: Map of usage of the estimation methods under LD model. The map compares the three methods according to their relative MSE (8). For each of 400 couples of parameters $\alpha = 0.5 \dots 10$ (x-axis) and $\rho = 0.2 \dots 5$ (y-axis, \log_5 -scale), 10^4 samples of size 100 of the $LD(\alpha, \rho, 0, 1)$ distribution were simulated. The estimates of (α, ρ) were calculated with the three methods. Each method is represented by a color: red for GF, green for P0, blue for ML.

- if the MSE is greater than 1, assign 0 to the corresponding RGB component;
- else, assign 1 minus the MSE to the corresponding RGB component.

The above experience is also performed considering H model instead of LD . The maps have been drawn with a \log_5 -scale for ρ (y-axis). The map can be roughly divided into four distinct parts:

- For $(\alpha, \rho) \in (0.5; 3) \times (0.2; 2.5)$, the color is essentially grey: the three methods are more or less equivalent.
- For $(\alpha, \rho) \in (3; 10) \times (0.2; 3.5)$, the color is magenta: the ML and GF methods are equivalent. The P0 method provides estimates with large MSEs or cannot be used because of the absence of null counts.
- For small values of ρ , the color is mainly red: The GF method is the only method with an acceptable MSE. Small values of ρ induce large jackpots. Moreover, the number of jackpots increases with α . Because of the Winsorization, the ML and P0 method (which uses ML to estimate ρ) provide estimates with very large MSEs.
- For ρ large, the color is darker and tends to black: the three methods provide estimates with large MSEs, specially for $\rho \in (3.5; 5)$, where jackpots are very small or absent. In those cases, estimating ρ with the GF method is not possible in practice (see previous sub-section). Consequently, the GF method will provide a biased estimate for α . The ML method, which uses the GF estimates to initialize the optimization of the log-likelihood, also provides biased estimates. The P0 method can provide good estimates of α whatever the value of ρ , which explains the presence of green areas at the top of the map. In a case where no jackpots are present in the sample it should be considered that a (heavy tailed) mutation model is not adapted.

Figure 2 is quite similar to Figure 1. There are still two remarkable differences:

- For ρ small and $m \leq 2$, the three methods seem to be more equivalent under H model than under LD model.
- For ρ large, GF method seems to provide better estimates under H model than under LD model.

The three methods should also be compared in terms of computational time. An illustration on real data will be given in section L.5. The slowest method is ML, for the reasons discussed in the previous section. It is even slower when the estimates are calculated under Haldane models H or HFN , when δ is positive, or if the initialization of ρ with the GF method fails. The GF method computes estimates of α and ρ (when possible) in negligible time. The P0 method outputs estimates of α in negligible time, but estimates of ρ are as slow as with ML.

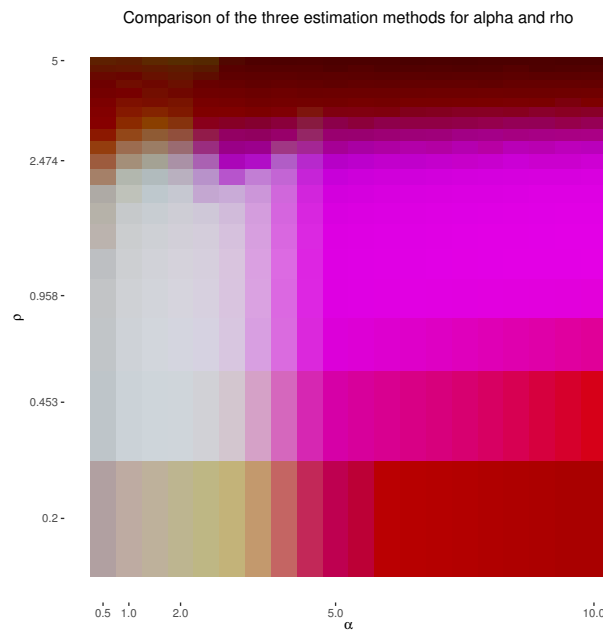


Figure 2: Map of usage of the estimation methods under H model. The map compares the three methods according to their relative MSE (8). For each of 400 couples of parameters $\alpha = 0.5 \dots 10$ (x-axis) and $\rho = 0.2 \dots 5$ (y-axis, log₅-scale), 10^4 samples of size 100 of the $H(\alpha, \rho, 0, 1)$ distribution were simulated. The estimates of (α, ρ) were calculated with the three methods. Each method is represented by a color: red for GF, green for P0, blue for ML.

Bias evidence

If the model used for the estimation does not correspond to the theoretical model, the estimates can be biased. Four different sources of bias are considered:

1. the final counts are random in the data, constant for the estimation model;
2. cell deaths occur in the data, not in the estimation model;
3. the lifetime distribution is different in the data and the estimation model;
4. the plating process is less than 100% efficient.

In each case, simulation experiments have been made along the following lines:

1. draw 10^4 samples of size 100, under one model;
2. for each sample, compute estimates using another model and the true one if available;
3. compare the empirical distributions of $\hat{\theta}/\theta$, where $\hat{\theta}$ is estimator and θ the true value.

For each figure, red lines mark unit, blue lines mark relative biases of 0.9 and 1.1. According to Figures 1 and 2, the GF method is at least equivalent in terms of the relative MSE (8) to the ML and P0 methods. Moreover, it is also the best in terms of computational time. Therefore, the bias evidences will be mainly illustrated with GF method.

Fluctuation of final counts: When N is constant, the estimate of π is derived by dividing the estimate of α by N . As mentioned in previous section, if N is a random variable, the relation between α and π can be explicit if the distribution K is known. However, this is not the case in practice. Usually, estimates of the expectation and variance of N are available at best. Assume that only the first two moments μ and σ^2 of N are known. Then a first order approximation of the Laplace transform \mathcal{L} can be used to reduce the bias. This method is explained in Ycart and Veziris (2014) for the P0 method. It has been adapted to ML and GF estimates. Figure 3 shows the influence of the coefficient of variation $C = \sigma/\mu$ on the ML estimate of π . The estimates were calculated with three different approaches:

- divide ML estimates of α by the empirical mean of N and ignore fluctuations of N (left boxplots);
- directly compute ML with the sample of pairs (mutant counts–final counts) (center boxplots);
- derive from ML estimates of α , taking into account of the empirical fluctuations of N (right boxplots).

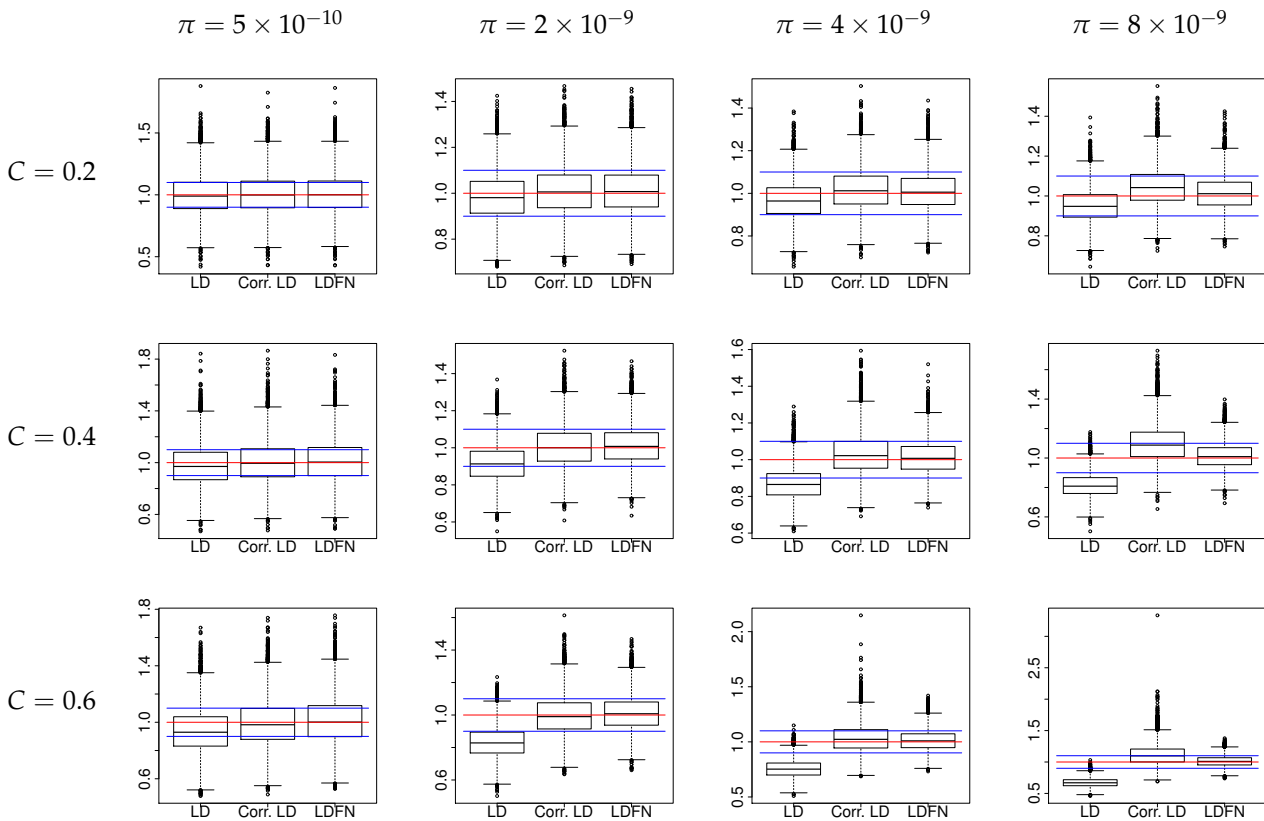


Figure 3: ML estimates of π ignoring fluctuations of final numbers or not. Red horizontal lines mark unit. Blue horizontal lines mark relative bias of 0.9 and 1.1. For each of the 12 sets of parameters $\pi = (0.5/\mu, 2/\mu, 4/\mu, 8/\mu)$ (columns), and $C = (0.2, 0.4, 0.6)$ (rows), 10^4 samples of size 100 of the $LDFN(\pi, \rho, 0, 1, K)$ distribution were simulated, with $\rho = 1$ and K being the log-normal distribution adjusted to mean $\mu = 10^9$ and coefficient of variation C . The ML estimates of π and ρ were calculated under model LD or $LDFN$. Each boxplot represents the distribution of the 10^4 ratio $\hat{\pi}/\pi$ obtained with LD model with $C = 0$ (left), LD model with bias reduction (center), $LDFN$ model (right).

According to the visual observations, the bias reduction seems to be working well when either the product $\pi\mu$ or C are small: most of the estimates have a relative bias smaller than 10%. However, the efficiency of the correction decreases as product $\pi\mu$ and C increase. In particular, for larger values of $\pi\mu$, the bias seems to be smaller without correction. It could be improved with a better approximation of \mathcal{L} , that implies knowing or estimating higher moments of N . Another solution is to improve the estimation of C . Here C was estimated by the ratio of the empirical standard deviation of the empirical mean, which is known to be a bad method in terms of MSE (Breunig, 2001).

Cell deaths: The PGFs (1), (4) and (2) depend on δ . Ignoring cell deaths involves a negative bias on the estimate of α . Assuming the exact value is known, this bias is removed. Figure 4 shows the influence of the death parameter δ on the GF estimate of α . The estimates are calculated with two different approaches:

1. computing GF estimates of α with $\delta = 0$ (left boxplots);
2. computing GF estimates of α with a theoretical value of δ (right boxplots).

The visual results show that the negative bias induced by ignoring cell deaths increases with the value of δ : the relative bias can easily exceed 10% for large values of δ . From the theory of branching processes, the growth process of a mutant clone is supercritical and δ has to be smaller than 0.5. In practice δ is smaller than 0.3. According to the boxplots, the relative bias induced by ignoring cell deaths can reach 0.80. These experiments also illustrate the difficulty in estimating δ . For example, the boxplots at the top right of the figure seems to show that the value of the likelihood for $\alpha = 4$ and $\delta = 0$ is very close to its value for $\alpha = 4$ and $\delta = 0.05$.

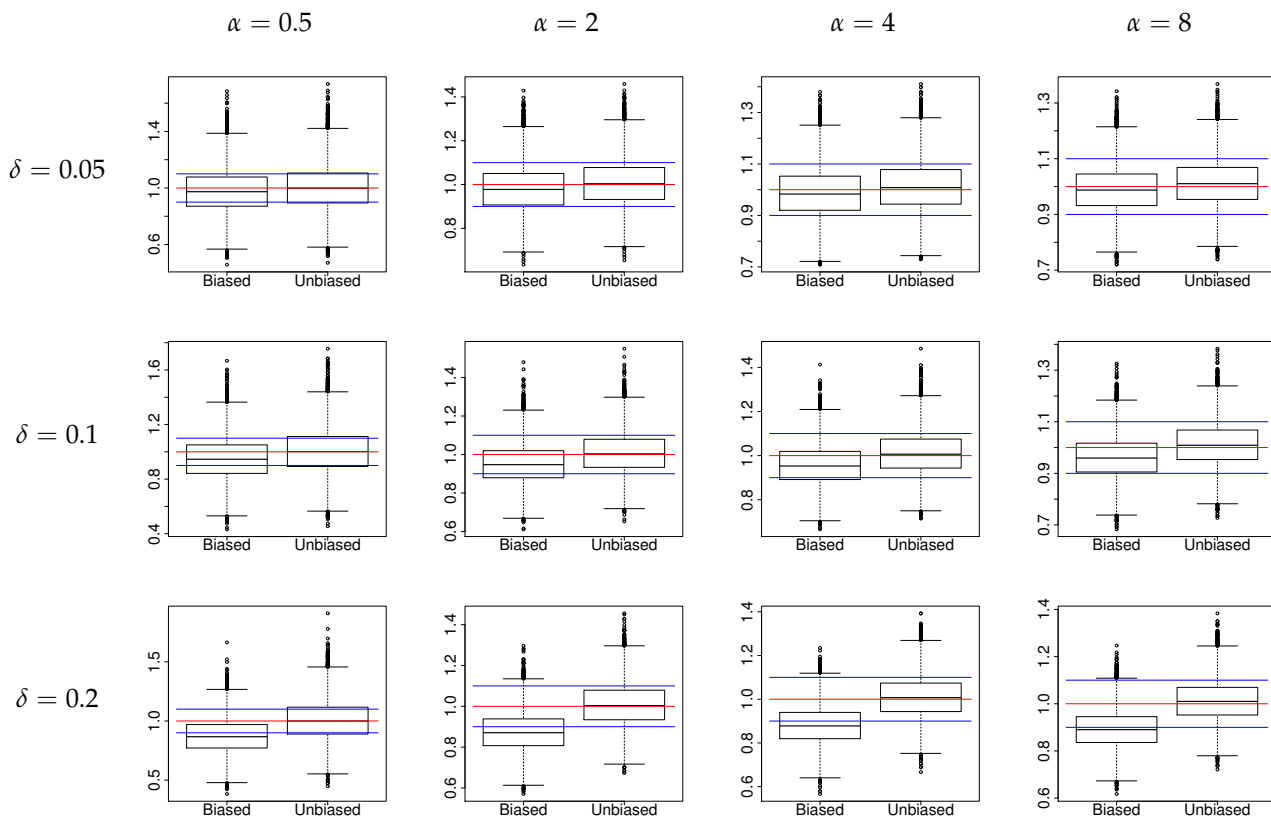


Figure 4: GF estimates of α ignoring cell deaths or not. Red horizontal lines mark unit. Blue horizontal lines mark relative bias of 0.9 and 1.1. For each of the 12 sets of parameters $\alpha = (0.5, 2, 4, 8)$ (columns), and $\delta = (0.05, 0.1, 0.2)$ (rows), 10^4 samples of size 100 of the $LD(\alpha, \rho, \delta, 1)$ distribution were simulated, with $\rho = 1$. The GF estimates of α and ρ were calculated under LD model. Each boxplot represents the distribution of the 10^4 ratio $\hat{\alpha}/\alpha$ of the estimates obtained without taking account of cells death (left) and with the theoretical value of δ (right).

Lifetime distribution: As mentioned earlier, the PGF h is explicit only for the LD and H distributions, i.e. when lifetimes are either exponential or constant. This is not the case in practice. If another lifetime distribution is used to simulate the data, and either LD or H are used to estimate the parameters, a bias will be induced on α and ρ . Figure 5 illustrates these observations. It shows the influence of the lifetime distribution on the GF estimates of α and ρ . The samples are drawn assuming the lifetimes are log-normally distributed. The estimates of α and ρ are calculated under LD (left boxplots) and H models (right boxplots).

From the visual observations, the LD and H models can be seen as extreme values for the lifetime distribution:

- both models correctly estimate α ;
- the LD model overestimates ρ and has a rather large dispersion of estimated values. The bias seems to increase as α increases;
- the H model correctly estimates ρ .

Plating efficiency: Ignoring the plating efficiency induces a negative bias on α and a positive bias on ρ . In practice, the exact value is known. Figure 6 shows the influence of ζ on the GF estimates of α and ρ . The estimates are calculated with two different approaches:

1. computing GF estimates with $\zeta = 1$ (left boxplots) ;
2. applying the correction of Stewart et al. (1990, eq. (41)) to GF estimates (center boxplots);
3. computing GF estimates with known value of ζ (right boxplots).

The visual results illustrate first the fact that the correction of Stewart et al. (1990) should not be used when $\rho \neq 1$: the induced bias on α is negative when $\rho > 1$, positive when $\rho < 1$. However, the variance

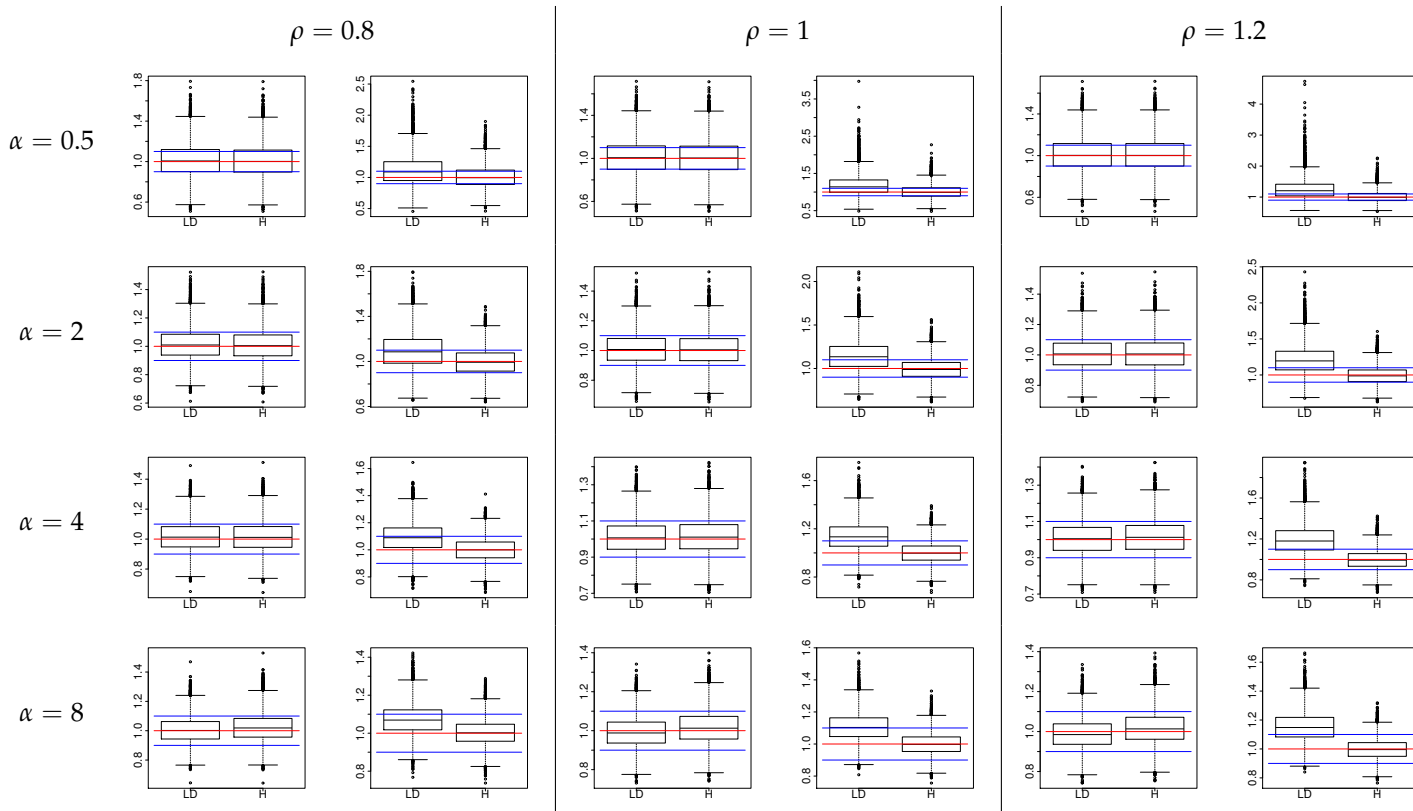


Figure 5: GF estimates of α and ρ under LD and H models on data drawn with MM model. Red horizontal lines mark unit. Blue horizontal lines mark relative bias of 0.9 and 1.1. For each of the 12 sets of parameters $\alpha = (0.5, 2, 4, 8)$ (rows), and $\rho = (0.8, 1, 1.2)$ (columns), 10^4 samples of size 100 of the $MM(\alpha, \rho, 0, 1, G)$ distribution were simulated, G being the log-normal distribution adjusted on Kelly and Rahn's data (Kelly and Rahn, 1932). The GF estimates of α and ρ were calculated with the two distributions $LD(\alpha, \rho, 0, 1)$ and $H(\alpha, \rho, 0, 1)$. For each couple (α, ρ) , the two first boxplots represent the distribution of the 10^4 ratio $\hat{\alpha}/\alpha$ obtained by the LD model (left) and the H model (right); the two last boxplots represent the distribution of the 10^4 ratio $\hat{\rho}/\rho$ obtained by the LD model (left) and the H model (right)

of the estimates obtained with these rectification is smaller than that of the estimates calculated with GF method.

Implementation details

The available functions are described here; more details are given in the manual. The behavior of inference functions for inputs which are out of practical limitations is described. Some details about the **Rcpp** implementation are also provided.

User interface

The **flan** package can be split into two distinct parts: the distribution of the final number of mutants and statistical inference. The functions `df1an`, `pf1an`, `qf1an` compute densities, probabilities and quantiles of LD and H distributions. The function `rf1an` outputs samples of pairs (mutant counts–final counts) following $LDFN$, HFN , or $MMFN$ where G has a log-normal or gamma distribution and K has a log-normal or Dirac distribution. K is adjusted to the mean and coefficient of variation provided by the user. Those functions have been designed on the principle of the classical distribution functions of R. A graphic function `draw.clone` is also provided. Using a binary tree, it represents the growth of a clone starting from a single normal cell with mutation occurrences until a finite time. The function `mutestim` computes estimates of α or π and ρ , using LD or H models. The three estimation methods are available. Fluctuations of final numbers and cells death are included. The function returns estimate(s) of the parameter(s) of interest and the standard deviations. The function `flan.test` uses asymptotic normality to perform one or two-sample hypothesis testing. It has been designed on the principle of

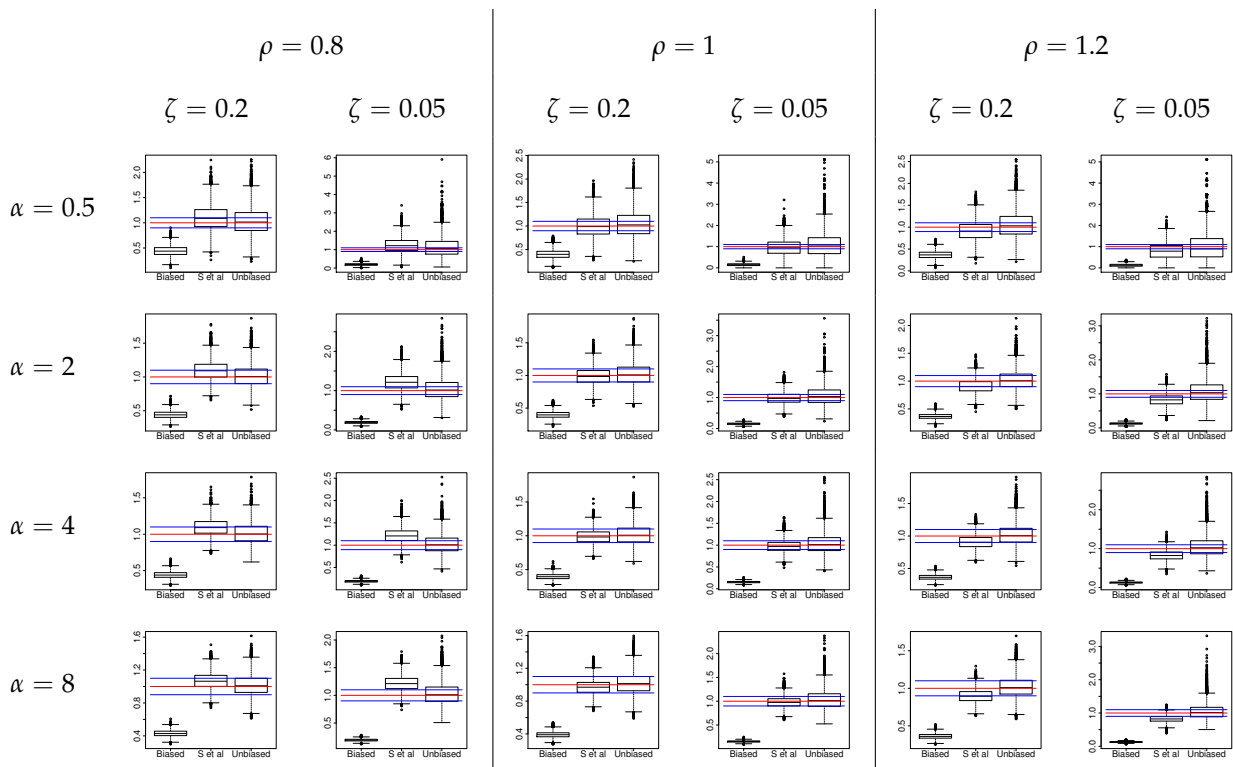


Figure 6: GF estimates of α ignoring plating efficiency or not. Red horizontal lines mark unit. Blue horizontal lines mark relative bias of 0.9 and 1.1. For each of the 12 sets of parameters $\alpha = (0.5, 2, 4, 8)$ (rows), and $\rho = (0.8, 1, 1.2)$ (columns), 10^4 samples of size 100 of the $LD(\alpha, \rho, 0, \zeta)$ distribution were simulated, with $\zeta = 0.2$ and $\zeta = 0.05$. The GF estimates of α and ρ were calculated under LD model. Each boxplot represents the distribution of the 10^4 ratio $\hat{\alpha}/\alpha$ obtained with $\zeta = 1$ (left), using the Stewart et al. (1990) correction (center), and with the true value of ζ (right).

the classical hypothesis testing functions of R, such as `t.test`. As mentioned in Section L.3, there are practical limitations for each estimation method. If the inputs of the `mutestim` function do not respect those limitations, it will output errors or warning messages:

- If $\delta = 0$, the P0 method can not be used if the sample does not contain any null counts. In that case, the `mutestim` function will throw an error with a message.
- For now, an inefficient plating (i.e. $\zeta < 1$) can be taken into account only with the GF method. If another method is specified, the `mutestim` function will return a warning message and set ζ at 1.
- Issues of the Winzorization parameter w (default is 1024) of ML method:
 1. If the minimum of the sample is larger than w , then the sample of mutant counts will be constant.
 2. If w is too large, then the optimization process can be very long.
- The GF method does not have limitations of usage, even for extreme cases where the ML estimators fail, i.e. samples with theoretical large α and small ρ . However, estimating ρ requires solving the zero equation discussed in Section L.3, which is theoretically solvable on \mathbb{R}^+ . In practice the interval of research is bounded. Thus, if the sample does not contain any jackpot, which means that ρ is very large, the zero equation may not have any solution on the interval. In that case, the function will return a warning message, and set the estimate of ρ at 1, and the estimate of the standard deviation at 0. In the `mutestim` function, the domain of research is $[0.01; 100]$.
- Moreover, the initialization of the ML method is done with GF method. Then the domain of optimization is $[0.1 \times \hat{\theta}_{GF}; 10 \times \hat{\theta}_{GF}]$, where $\hat{\theta}_{GF}$ is the GF estimate(s) of the parameter(s) of interest. Then, if the GF method does not succeed to estimate ρ , there is no chance to estimate it with ML. A warning message is returned if the initialization of the estimate of ρ with GF fails.

The function `flan.test` is a wrapper function of `mutestim`. It will output the same errors or warning messages if its inputs do not respect the practical limitations.

Implementation

Since most functions involve loops that are more expensive in R than in C, **flan** has been implemented with **Rcpp** modules. This paradigm provides an easy way to expose C++ functions and classes to R. There are four main classes in the C++ implementation:

- **FLAN_Sim**: random generation for *MM* and *MMFN* distributions. One of its members is a variable of the following type;
- **FLAN_SimClone**: random generation for clone size distribution according to the lifetimes distribution;
- **FLAN_MutationModel**: computation of the descriptive functions (probabilities, PGF,...) for *LD* and *H* distributions. One of its members is a variable of the following type;
- **FLAN_Clone**: computation of the descriptive functions for clone size distribution according to the lifetimes distribution.

The **Rcpp** interface enables also to import into the C++ code any R function. In particular, it is interesting to import the R functions which are already implemented in C. Thus no external C/C++ library is required. The installation remains basic, and the size of the installed package is reduced. For example, the computations of *LD* distributions involve numeric integrations. The C libraries **integration** and **alglib** compute integrals with an accuracy close to machine precision. We could use those libraries but the R function `integrate` is actually implemented in C. The R function can then be directly called into the C++ code. However, such imports increase the computational time and memory consumption. This is thus unsatisfactory. Another solution uses the package **RcppGSL** (Eddelbuettel, 2013, chap. 11), which creates an interface between **Rcpp** and the C library `gsl`. Several integration methods are thus available. Since it only requires that `gsl` is correctly installed, this solution is a good compromise between easy setup and computational cost.

Computing the probabilities for the *H* distribution with $\delta > 0$ involves squaring high degree polynomials. Such polynomials are easily treated by the package **polynom**. However its implementation raises memory issues, because of the degree of the polynomials involved. A more efficient way is to use the Fast Fourier Transform. It is provided by the C library `fftw3`, which can raise some installation issues. The R function `fft` could be directly called into the C++ code. For the same reasons as for the `integrate` function, it is more adequate to use the package **RcppArmadillo** (Eddelbuettel (2013, chap.10) ; Eddelbuettel and Sanderson (2014)). This package links **Rcpp** to the C++ library `armadillo`, which is dedicated to linear algebra. In particular, it includes a performant Fast Fourier Transform.

Finally, likelihood optimizations in the ML and P0 methods are done with a bounded BFGS optimizer. The package **lbfgsb3** provides the eponymous function which is implemented in Fortran. It is much faster than the basic R function `optim`.

Examples of usage

Some examples on the real data included in **flan** are provided.

Practical limitations, influence of bias sources and comparison of the estimation methods in terms of computational time are illustrated. Consider first the eleventh sample of mutant counts of the werhoff data (Werngren and Hoffner, 2003):

```
werhoff$samples$W11$mc
## [1] 0 0 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 4 4 4 4 5 5
```

This sample does not contain any jackpot, then the theoretical fitness in a mutation model, should be very large. If the GF method is used, it will output a warning message and set the fitness at $\rho = 1$, as customarily done in the literature (Foster, 2006):

```
W <- werhoff$samples$W11$mc

# Compute GF estimates of mutations number and fitness}
mutestim(mc = W, method = "GF")

## Warning in mutestim(mc = W, method = "GF"): Impossible to estimate
## 'fitness' with 'GF'-method: 'fitness' is set to default value 1 and
## only mutations number is estimated.

## $mutations
## [1] 0.8792917
```

```
##
## $sd.mutations
## [1] 0.260549
##
## $fitness
## [1] 1
##
## $sd.fitness
## [1] 0
```

Notice that Table 1 of [Werngren and Hoffner \(2003\)](#) shows mutants counts and mean final numbers of cells for 1 mL of solution. However, the total volume of the culture is 5 mL. In other words, a plating efficiency of 20% has been applied. The fact is the fitness parameter can not be estimated, even taking into account the plating efficiency:

```
# Volume of each sample: 1 mL
# Total volume of each culture: 5 mL
# i.e. plating efficiency of 0.2
pef <- 0.2
```

```
# Compute GF estimates of mutation probability and fitness
# taking into account the plating efficiency
mutestim(mc = W, plateff = pef, method = "GF")
```

```
## Warning in mutestim(mc = W, plateff = pef, method = "GF"): Impossible to estimate 'fitness' with
## 'GF'-method: 'fitness' is set to default value 1 and only mutations number is estimated.
```

```
## $mutations
## [1] 2.804244
##
## $sd.mutations
## [1] 0.74011
##
## $fitness
## [1] 1
##
## $sd.fitness
## [1] 0
```

Using the P0 method is a way to realize that setting $\rho = 1$ by default can be misleading. Since this method does not depend on the lifetime distribution, the estimate of α will not depend on the value of ρ .

```
# Compute P0 estimate of mutations number
mutestim(mc = W, fitness = 1, method = "P0")
```

```
## $mutations
## [1] 2.525729
##
## $sd.mutations
## [1] 0.678233
```

The P0 estimate of α is very different from the GF estimate (when $\zeta = 1$). Consider now the sample of [David \(1970, Tab. 2\)](#), which includes rifampin-resistant bacteria counts.

```
david$D11
```

```
## $mc
## [1] 4 0 1 0 1 0 0 0 0 0
##
## $fn
## [1] 1.3e+09 9.2e+08 1.3e+09 2.5e+09 1.3e+09 1.6e+09 1.3e+09 2.5e+09
## [9] 2.5e+09 2.0e+09
```

Since the 4 value can be seen as a jackpot, the GF method can be used to estimate α and ρ . Now let us compute the ML estimates of π and ρ taking into account or not of the final counts, under the LD model.

```

D <- david$D11
Dmfn <- mean(D$fn)

# Compute ML estimates and confidence intervals of mutation probability and
# fitness with empirical mean and null coefficient of variation
ft <- flan.test(mc = D$mc, mfn = Dmfn)
ft$estimate

## mutation probability          fitness
##          2.067641e-10          2.214676e+00

ft$conf.int

##      mutation probability  fitness
## bInf          0.000000e+00 0.000000
## bSup          4.423916e-10 8.109577
## attr(,"conf.level")
## [1] 0.95

# Compute ML estimates and confidence intervals of mutation probability and
# fitness with empirical mean and empirical coefficient of variation
ft <- flan.test(mc = D$mc, mfn = Dmfn, cvfn = sd(D$fn)/Dmfn)
ft$estimate

## mutation probability          fitness
##          2.092720e-10          2.214676e+00

ft$conf.int

##      mutation probability  fitness
## bInf          0.000000e+00 0.000000
## bSup          4.506155e-10 8.109577
## attr(,"conf.level")
## [1] 0.95

# Compute ML estimates and confidence intervals of mutation probability and
# fitness with couples (mc,fn)
ft <- flan.test(mc = D$mc, fn = D$fn)
ft$estimate

## mutation probability          fitness
##          1.977135e-10          2.048984e+00

ft$conf.int

##      mutation probability  fitness
## bInf          0.000000e+00 0.000000
## bSup          4.078591e-10 7.127426
## attr(,"conf.level")
## [1] 0.95

```

The sample of final counts is denoted by $D_{11}^{(FN)}$. The empirical mean of the final counts is denoted by $\bar{\mu}$, the empirical coefficient of variation by \bar{C} . Table 1 displays the ML estimates of π and ρ , in the same way as for Figure 3. Their 95% confidence intervals are provided. Comparing the second row to the fourth, one can see that neglecting final number fluctuations induces a bias of order 5% on π , 10% on ρ . From the third row, it turns out that the correction taking into account \bar{C} , has not improved the estimate of π . Notice also that due to the small size sample, the confidence intervals are quite large. Consider finally the data from Boe et al. (1994, Tab. 4). The author studied mutations of *Escherichia coli* from sensitivity to nalidixic acid resistance. 23 samples of resistant bacteria counts are provided. As in the original paper, the 23 samples are concatenated as one. The three estimation methods are compared in terms of computational time on the resulting sample of size 1104. The package **microbenchmark** is used, evaluating 10^4 times each method on the sample. The methods are compared when only α is estimated ($\rho = 1$), and when the couple (α, ρ) is estimated. In both cases the estimates are computed under the model LD with $\delta = 0$ and $\zeta = 1$.

	Estimates of π	Confidence intervals (95%) of π	Estimates of ρ	Confidence intervals (95%) of ρ
ML using $\mu = \bar{\mu}, C = 0$	2.07×10^{-10}	$[0; 4.22 \times 10^{-10}]$	2.21	$[0; 8.11]$
ML using $\mu = \bar{\mu}, C = \bar{C}$	2.09×10^{-10}	$[0; 4.51 \times 10^{-10}]$	2.21	$[0; 8.11]$
ML using $D_{11}^{(FN)}$	1.98×10^{-10}	$[0; 4.08 \times 10^{-10}]$	2.05	$[0; 7.13]$

Table 1: ML estimates of π and ρ of data set from (David, 1970, Tab. 2) ignoring fluctuations of final numbers or not. Each row shows the ML estimates of π and ρ and their 95% confidence intervals, deducing from LD model with $C = 0$ (first row), $C = \bar{C}$ (second row), and directly with LDFN model (third row).

```

B <- unlist(boeal)      # Concatenation of the 23 samples
require(microbenchmark)

# Comparing the methods in terms of computational performance
# (mutations number only)
microbenchmark(P0 = mutestim(mc = B, fitness = 1, method = "P0"),
               GF = mutestim(mc = B, fitness = 1, method = "GF"),
               ML = mutestim(mc = B, fitness = 1, method = "ML"),
               unit = "ms", times = 1e4)

## Unit: milliseconds
## expr      min       lq      mean     median      uq      max neval
##   P0  0.063696  0.0800720  0.09701435  0.0899955  0.0986275  2.149537 10000
##   GF  0.327989  0.3694555  0.41309310  0.3850380  0.4035050  26.774004 10000
##   ML  8.381844 10.5333670 12.23137063 10.9118690 11.3860045 43.592177 10000

# Comparing the methods in terms of computational performance}
microbenchmark(P0 = mutestim(mc = B, method = "P0"),
               GF = mutestim(mc = B, method = "GF"),
               ML = mutestim(mc = B, method = "ML"),
               unit = "ms", times = 1e4)

## Unit: milliseconds
## expr      min       lq      mean     median      uq      max neval
##   P0  8.451605 11.004419 12.832827 11.848562 13.082066 43.00397 10000
##   GF  2.274944  2.419663  2.688851  2.481529  2.549597 33.19333 10000
##   ML 73.704656 79.581238 83.841150 81.802464 84.323398 120.65768 10000

```

The results are shown on Figure 7, as boxplots of timing distributions. Times are in milliseconds and plotted on log-scale. As mentioned earlier, the ML method is the slowest. The GF method seems to be quite faster than the P0 method. However, the estimates of ρ of the P0 method is calculated maximizing the likelihood. This step is more costly in term of computational time. If only α has to be estimated, the P0 method is faster than the GF method.

Bibliography

- W. P. Angerer. An explicit representation of the Luria-Delbrück distribution. *J. Math. Biol.*, 42(2): 145–174, 2001a. URL <https://doi.org/10.1007/s002850000053>. [p335]
- W. P. Angerer. A note on the evaluation of fluctuation experiments. *Mutation Research*, 479:207–224, 2001b. URL [https://doi.org/10.1016/S0027-5107\(01\)00203-2](https://doi.org/10.1016/S0027-5107(01)00203-2). [p335]

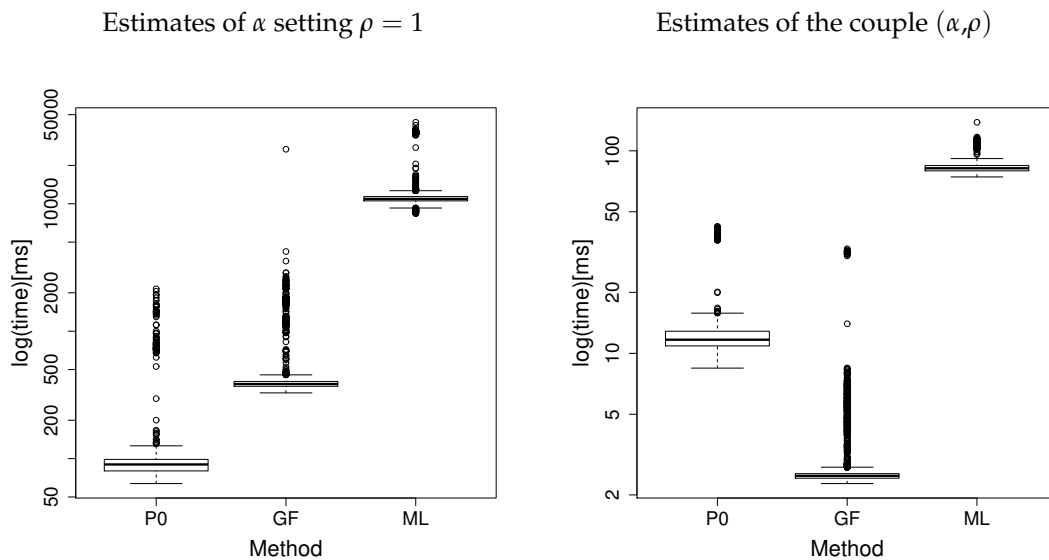


Figure 7: Computational time of the three methods on data from Boe et al. (1994, Tab. 4). Data consist in the 23 samples of boeal concatenated as one. For each method, the estimates of α setting $\rho = 1$ (left boxplots) or of α and ρ (right boxplots) have been computed under model *LD*. The timings have been returned with **microbenchmark**, evaluating 10^4 times each method. Times are in milliseconds and plotted on log-scale.

- P. Armitage. The statistical theory of bacterial populations subject to mutation. *J. R. Statist. Soc. B*, 14: 1–40, 1952. URL <http://www.jstor.org/stable/2984083>. [p334]
- K. B. Athreya and P. E. Ney. *Branching Processes*. Springer-Verlag, Berlin Heidelberg, 1972. URL <https://doi.org/10.1007/978-3-642-65371-1>. [p334, 335, 337]
- M. S. Bartlett. *An Introduction to Stochastic Processes, with Special Reference to Methods and Applications*. Cambridge University Press, 3rd edition, 1978. [p334]
- R. Bellman and T. Harris. On age-dependent binary branching processes. *Ann. Math.*, 55(2):280–295, 1952. URL <https://www.jstor.org/stable/1969779>. [p334, 336]
- L. Boe, T. Tolker-Nielsen, K. M. Eegholm, H. Spliid, and A. Vrang. Fluctuation analysis of mutations to nalidixic acid resistance in *Escherichia Coli*. *J. Bacteriol.*, 176(10):2781–2787, 1994. URL <https://doi.org/10.1128/jb.176.10.2781-2787>. [p347, 349]
- R. Breunig. An almost unbiased estimator of the coefficient of variation. *Econ. Lett.*, 70(1):15–19, 2001. URL [https://doi.org/10.1016/S0165-1765\(00\)00351-7](https://doi.org/10.1016/S0165-1765(00)00351-7). [p341]
- H. L. David. Probability distribution of drug-resistant mutants in unselected populations of *Mycobacterium Tuberculosis*. *Appl. Microbiol.*, 20(5):810–814, 1970. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC377053/>. [p346, 348]
- A. Dewanji, E. G. Luebeck, and S. H. Moolgavkar. A generalized Luria-Delbrück model. *Math. Biosci.*, 197(2):140–152, 2005. URL <https://doi.org/10.1016/j.mbs.2005.07.003>. [p335]
- D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer-Verlag, New York, 2013. URL <https://doi.org/10.1007/978-1-4614-6868-4>. [p335, 345]
- D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Comput. Stat. Data An.*, 70:1054–1063, 2014. URL <https://doi.org/10.1016/j.csda.2013.02.005>. [p345]
- P. Embrechts and J. Hawkes. A limit theorem for the tails of discrete infinitely divisible laws with applications to fluctuation theory. *J. Austral. Math. Soc. Series A*, 32:412–422, 1982. URL <https://doi.org/10.1017/S1446788700024976>. [p337]
- P. L. Foster. Methods for determining spontaneous mutation rates. *Method. Enzymol.*, 409:195–213, 2006. URL [https://doi.org/10.1016/S0076-6879\(05\)09012-9](https://doi.org/10.1016/S0076-6879(05)09012-9). [p345]

- A. Gillet-Markowska, G. Louvel, and G. Fisher. Bz-rates: a web-tool to estimate mutation rates from fluctuation analysis. *G3*, 5(11):2323–2327, 2015. URL <https://doi.org/10.1534/g3.115.019836>. [p335]
- B. M. Hall, C. X. Ma, P. Liang, and K. K. Singh. Fluctuation AnaLysis CalculatOR: a web tool for the determination of mutation rate using Luria–Delbrück fluctuation analysis. *Bioinformatics*, 25(12):1564–1565, 2009. URL <https://doi.org/10.1093/bioinformatics/btp253>. [p335]
- A. Hamon and B. Ycart. Statistics for the Luria-Delbrück distribution. *Elect. J. Statist.*, 6:1251–1272, 2012. URL <https://doi.org/10.1214/12-EJS711>. [p334, 335, 336, 337, 338]
- C. D. Kelly and O. Rahn. The growth rate of individual bacterial cells. *J. Bacteriol.*, 23(2):147–153, 1932. URL <http://europepmc.org/articles/PMC533308>. [p343]
- N. L. Komarova, L. Wu, and P. Baldi. The fixed-size Luria-Delbrück model with a nonzero death rate. *Math. Biosci.*, 210(1):253–290, 2007. URL <https://doi.org/10.1016/j.mbs.2007.04.007>. [p335]
- D. E. Lea and C. A. Coulson. The distribution of the number of mutants in bacterial populations. *Journal of Genetics*, 49(3):264–285, 1949. URL <https://doi.org/10.1007/BF02986080>. [p334]
- S. E. Luria and M. Delbrück. Mutations of bacteria from virus sensitivity to virus resistance. *Genetics*, 28(6):491–511, 1943. URL <https://www.genetics.org/content/28/6/491>. [p334, 337]
- W. T. Ma, G. v. H. Sandri, and S. Sarkar. Analysis of the Luria-Delbrück distribution using discrete convolution powers. *J. Appl. Probab.*, 29(2):255–267, 1992. URL <https://doi.org/10.1017/S0021900200043023>. [p334]
- B. Rémillard and R. Theodorescu. Inference based on the empirical probability generating function for mixtures of Poisson distributions. *Statist. Decisions*, 18:349–366, 2000. URL <https://doi.org/10.1524/strm.2000.18.4.349>. [p334, 338]
- S. Sarkar. Haldane’s solution of the Luria-Delbrück distribution. *Genetics*, 127:257–261, 1991. [p334]
- F. M. Stewart. Fluctuation analysis: The effect of plating efficiency. *Genetica*, 84(1):51–55, 1991. URL <https://doi.org/10.1007/BF00123984>. [p335]
- F. M. Stewart, D. M. Gordon, and B. R. Levin. Fluctuation analysis: The probability distribution of the number of mutants under different conditions. *Genetics*, 124(1):175–185, 1990. URL <https://www.genetics.org/content/124/1/175>. [p335, 337, 342, 344]
- J. Werngren and S. E. Hoffner. Drug susceptible *Mycobacterium Tuberculosis Beijing* genotype does not develop motation-conferred resistance to Rifampin at an elevated rate. *J. Clin. Microbiol.*, 41(4):1520–1524, 2003. URL <https://doi.org/10.1128/jcm.41.4.1520-1524.2003>. [p345, 346]
- R. Wilcox. *Introduction to Robust Estimation and Hypothesis Testing*. Elsevier, Amsterdam, 3rd edition, 2012. URL <https://doi.org/10.1016/C2010-0-67044-1>. [p334, 337]
- B. Ycart. Fluctuation analysis: Can estimates be trusted? *PLoS One*, 8(12):1–12, 2013. URL <https://doi.org/10.1371/journal.pone.0080958>. [p334]
- B. Ycart. Fluctuation analysis with cell deaths. *J. Appl. Probab. Statist*, 9(1):13–29, 2014. URL <https://arxiv.org/abs/1207.4375>. [p335, 337]
- B. Ycart and N. Veziris. Unbiased estimates of mutation rates under fluctuating final counts. *PLoS One*, 9(7):1–10, 2014. URL <https://doi.org/10.1371/journal.pone.0101434>. [p335, 337, 340]
- Q. Zheng. Statistical and algorithmic methods for fluctuation analysis with SALVADOR as an implementation. *Math. Biosci.*, 176(2):237–252, 2002. URL [https://doi.org/10.1016/S0025-5564\(02\)00087-1](https://doi.org/10.1016/S0025-5564(02)00087-1). [p335]
- Q. Zheng. New algorithms for Luria-Delbrück fluctuation analysis. *Math. Biosci.*, 196(2):198–214, 2005. URL <https://doi.org/10.1016/j.mbs.2005.03.011>. [p334, 337]

Adrien Mazoyer

Statistique pour les sciences du Vivant et de l’Homme, Laboratoire Jean Kuntzmann

Université Grenoble Alpes

Bâtiment IMAG, 700 Avenue Centrale

38400 SAINT MARTIN D’HÈRES

FRANCE

adrien.mazoyer@univ-grenoble-alpes.fr

Rémy Drouilhet

Fiabilité et Géométrie Aléatoire, Laboratoire Jean Kuntzmann

Université Grenoble Alpes

Bâtiment IMAG, 700 Avenue Centrale

38400 SAINT MARTIN D'HÈRES

FRANCE

remy.drouilhet@univ-grenoble-alpes.fr

Stéphane Després

Laboratoire Jean Kuntzmann

Université Grenoble Alpes

Bâtiment IMAG, 700 Avenue Centrale

38400 SAINT MARTIN D'HÈRES

FRANCE

stephane.despreaux@univ-grenoble-alpes.fr

Bernard Ycart

Statistique pour les sciences du Vivant et de l'Homme, Laboratoire Jean Kuntzmann

Université Grenoble Alpes

700 Avenue Centrale, 38400 SAINT MARTIN D'HÈRES

FRANCE

bernard.ycart@univ-grenoble-alpes.fr

Multilabel Classification with R Package `mlr`

by Philipp Probst, Quay Au, Giuseppe Casalicchio, Clemens Stachl and Bernd Bischl

Abstract We implemented several multilabel classification algorithms in the machine learning package `mlr`. The implemented methods are binary relevance, classifier chains, nested stacking, dependent binary relevance and stacking, which can be used with any base learner that is accessible in `mlr`. Moreover, there is access to the multilabel classification versions of `randomForestSRC` and `rFerns`. All these methods can be easily compared by different implemented multilabel performance measures and resampling methods in the standardized `mlr` framework. In a benchmark experiment with several multilabel datasets, the performance of the different methods is evaluated.

Introduction

Multilabel classification is a classification problem where multiple target labels can be assigned to each observation instead of only one, like in multiclass classification. It can be regarded as a special case of multivariate classification or multi-target prediction problems, for which the scale of each response variable can be of any kind, for example nominal, ordinal or interval.

Originally, multilabel classification was used for text classification (McCallum, 1999; Schapire and Singer, 2000) and is now used in several applications in different research fields. For example, in image classification, a photo can belong to the classes *mountain* and *sunset* simultaneously. Zhang and Zhou (2008) and others (Boutell et al., 2004) used multilabel algorithms to classify scenes on images of natural environments. Furthermore, gene functional classifications is a popular application of multilabel learning in the field of biostatistics (Elisseeff and Weston, 2002; Zhang and Zhou, 2008). Additionally, multilabel classification is useful to categorize audio files. Music genres (Sanden and Zhang, 2011), instruments (Kursa and Wieczorkowska, 2014), bird sounds (Briggs et al., 2013) or even emotions evoked by a song (Trohidis et al., 2008) can be labeled with several categories. A song could, for example, be classified both as a *rock song* and a *ballad*.

An overview of multilabel classification was given by Tsoumakas and Katakis (2007). Two different approaches exist for multilabel classification. On the one hand, there are algorithm adaptation methods that try to adapt multiclass algorithms so they can be applied directly to the problem. On the other hand, there are problem transformation methods, which try to transform the multilabel classification into binary or multiclass classification problems.

Regarding multilabel classification software, there is the `mlDR` (Charte and Charte, 2015) R package that contains some functions to get basic characteristics of specific multilabel datasets. The package is also useful for transforming multilabel datasets that are typically saved as ARFF-files (Attribute-Relation File Format) to data frames and vice versa. This is especially helpful because until now only the software packages MEKA (Read and Reutemann, 2012) and Mulan (Tsoumakas et al., 2011) were available for multilabel classification and both require multilabel datasets saved as ARFF-files to be executed. Additionally, the `mlDR` package provides a function that applies the binary relevance or label powerset transformation method which transforms a multilabel dataset into several binary datasets (one for each label) or into a multiclass dataset using the set of labels for each observation as a single target label, respectively. However, there is no R package that provides a standardized interface for executing different multilabel classification algorithms. With the extension of the `mlr` package described in this paper, it will be possible to execute several multilabel classification algorithms in R with many different base learners.

In the following section of this paper, we will describe the implemented multilabel classification methods and then give a practical instruction of how to execute these algorithms in `mlr`. Finally, we present a benchmark experiment that compares the performance of all implemented methods on several datasets.

Multilabel classification methods implemented in `mlr`

In this section, we present multilabel classification algorithms that are implemented in the `mlr` package (Bischl et al., 2016), which is a powerful and modularized toolbox for machine learning in R. The package offers a unified interface to more than a hundred learners from the areas classification, regression, cluster analysis and survival analysis. Furthermore, the package provides functions and tools that facilitate complex workflows such as hyperparameter tuning (see, e.g., Lang et al., 2015) and

feature selection that can now also be applied to the multilabel classification methods presented in this paper. In the following, we list the algorithm adaptation methods and problem transformation methods that are currently available in **mlr**.

Algorithm adaptation methods

The **rFerns** (Kursa and Wieczorkowska, 2014) package contains an extension of the random ferns algorithm for multilabel classification. In the **randomForestSRC** (Ishwaran and Kogalur, 2016) package, multivariate classification and regression random forests can be created. In the classification case, the difference to standard random forests is that a composite normalized Gini index splitting rule is used. Multilabel classification can be achieved by using binary encoding for the labels.

Problem transformation methods

Problem transformation methods try to transform the multilabel classification problem so that a simple binary classification algorithm, the so-called base learner, can be applied.

Let n be the number of observations, let p be the number of predictor variables and let $Z = \{z_1, \dots, z_m\}$ be the set of all labels. Observations follow an unknown probability distribution \mathcal{P} on $\mathcal{X} \times \mathcal{Y}$, where \mathcal{X} is a p -dimensional input space of arbitrary measurement scales and $\mathcal{Y} = \{0, 1\}^m$ is the target space. In our notation, $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)})^\top \in \mathcal{X}$ refers to the i -th observation and $x_j = (x_j^{(1)}, \dots, x_j^{(n)})^\top$ refers to the j -th predictor variable, for all $i = 1, \dots, n$ and $j = 1, \dots, p$. The observations $\mathbf{x}^{(i)}$ are associated with their multilabel outcomes $\mathbf{y}^{(i)} = (y_1^{(i)}, \dots, y_m^{(i)})^\top \in \mathcal{Y}$, for all $i = 1, \dots, n$. For all $k = 1, \dots, m$, setting $y_k^{(i)} = 1$ indicates the relevance, i.e., the occurrence, of label z_k for observation $\mathbf{x}^{(i)}$ and setting $y_k^{(i)} = 0$ indicates the irrelevance of label z_k for observation $\mathbf{x}^{(i)}$. The set of all instances thus becomes $D = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$. Furthermore, $\mathbf{y}_k = (y_k^{(1)}, \dots, y_k^{(n)})^\top$ refers to the k -th target vector, for all $k = 1, \dots, m$. Throughout this paper, we visualize multilabel classification problems in the form of tables ($n = 6, p = 3, m = 3$):

$$D \hat{=} \begin{array}{|ccc|ccc|} \hline x_1 & x_2 & x_3 & y_1 & y_2 & y_3 \\ \hline & & & 0 & 0 & 1 \\ & & & 1 & 0 & 1 \\ & & & 1 & 1 & 0 \\ & & & 1 & 1 & 1 \\ & & & 1 & 1 & 0 \\ & & & 1 & 1 & 0 \\ \hline \end{array} \tag{1}$$

The entries of x_1, x_2, x_3 can be of any (valid) kind, like continuous, binary, or categorical. The table in (1) visualizes this as an empty gray background. The target variables are indicated by a red background and can only take the binary values 0 or 1.

Binary relevance

The binary relevance method (BR) is the simplest problem transformation method. BR learns a binary classifier for each label. Each classifier C_1, \dots, C_m is responsible for predicting the *relevance* of their corresponding label by a 0/1 prediction:

$$C_k : \mathcal{X} \longrightarrow \{0, 1\}, \quad k = 1, \dots, m$$

These binary prediction are then combined to a multilabel target. An unlabeled observation $\mathbf{x}^{(l)}$ is assigned the prediction $(C_1(\mathbf{x}^{(l)}), C_2(\mathbf{x}^{(l)}), \dots, C_m(\mathbf{x}^{(l)}))^\top$. Hence, labels are predicted independently of each other and label dependencies are not taken into account. BR has linear computational complexity with respect to the number of labels and can easily be parallelized.

Modeling label dependence

In the problem transformation setting, the arguably simplest way (Montañés et al., 2014) to model label dependence is to condition classifier models not only on \mathcal{X} , but also on other label information. The idea is to augment the input space \mathcal{X} with information of the output space \mathcal{Y} , which is available in the training step. There are different ways to realize this idea of augmenting the input space. In essence, they can be distinguished in the following way:

- Should the true label information be used? (True vs. predicted label information)
- For predicting one label z_k , should all other labels augment the input space, or only a subset of labels? (Full vs. partial conditioning)

True vs. predicted label information

During the training of a classifier C_k for the label z_k , the label information of other labels are available in the training data. Consequently, these true labels can directly be used as predictors to train the classifier. Alternatively, the predictions that are produced by some classifier can be used instead of the true labels.

A classifier, which is trained on additional labels as predictors, needs those additional labels as input variables. Since these labels are not available at prediction time, they need to be predicted first. When the true label information is used to augment the feature space in the training of a classifier, the assumption that the training data and the test data should be identically distributed is violated (Senge et al., 2013). If the true label information is used in the training data and the predicted label information is used in the test data, the training data is not representative for the test data. However, experiments (Montañés et al., 2014; Senge et al., 2013) show that none of these methods should be dismissed immediately. Note that we use the superscript “true” or “pred” to emphasize that a classifier C_k^{true} or C_k^{pred} used true labels or predicted labels as additional predictors during training, respectively.

Suppose there are $n = 6$ observations with $p = 3$ predictors and $m = 3$ labels. The true label y_3 shall be used to augment the feature space of a binary classifier C_1^{true} for label y_1 . C_1^{true} is thus trained on all predictors and the true label y_3 . The binary classification task for label y_1 is therefore:

$$\text{Train } C_1^{\text{true}} \text{ on } \begin{array}{|c|c|c|c|c|} \hline x_1 & x_2 & x_3 & y_3 & y_1 \\ \hline & & & 0 & 0 \\ & & & 1 & 1 \\ & & & 0 & 1 \\ & & & 1 & 1 \\ & & & 1 & 1 \\ & & & 0 & 1 \\ & & & 0 & 1 \\ \hline \end{array} \text{ to predict } y_1 \tag{2}$$

For an unlabeled observation $\mathbf{x}^{(l)}$, only the three predictor variables $x_1^{(l)}, \dots, x_3^{(l)}$ are available at prediction time. However, the classifier C_1^{true} needs a 4-dimensional observation $(\mathbf{x}^{(l)}, y_3^{(l)})$ as input. The input $y_3^{(l)}$ therefore needs to be predicted first. A new level-1 classifier C_3^{lv1} , which is trained on the set $D' = \cup_{i=1}^6 \{(\mathbf{x}^{(i)}, y_3^{(i)})\}$, will make those predictions for $y_3^{(l)}$. The training task is:

$$\text{Train } C_3^{\text{lv1}} \text{ on } D' \hat{=} \begin{array}{|c|c|c|c|} \hline x_1 & x_2 & x_3 & y_3 \\ \hline & & & 1 \\ & & & 1 \\ & & & 1 \\ & & & 0 \\ & & & 1 \\ & & & 0 \\ & & & 0 \\ \hline \end{array} \text{ to predict } y_3 \tag{3}$$

Therefore, for a new observation $\mathbf{x}^{(l)}$, the predicted label $\hat{y}_3^{(l)}$ is obtained by using C_3^{lv1} on $\mathbf{x}^{(l)}$. The final prediction for $y_1^{(l)}$ is then obtained by using C_1^{true} on $(\mathbf{x}^{(l)}, \hat{y}_3^{(l)})$.

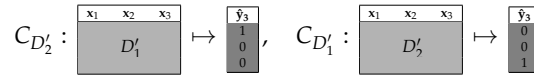
The alternative to (2) would be to use predicted labels \hat{y}_3 instead of true labels y_3 . These labels should be produced by means of an out-of-sample prediction procedure (Senge et al., 2013). This can be done by an internal leave-one-out cross-validation procedure, which can of course be computationally intensive. Because of this, coarser resampling strategies can be used. As an example, an internal 2-fold cross-validation will be shown here. Again, let $D' = \cup_{i=1}^6 \{(\mathbf{x}^{(i)}, y_3^{(i)})\}$ be the set of all predictor variables with y_3 as target variable. Using 2-fold cross-validation, the dataset D' is split into two parts $D'_1 = \cup_{i=1}^3 \{(\mathbf{x}^{(i)}, y_3^{(i)})\}$ and $D'_2 = \cup_{i=4}^6 \{(\mathbf{x}^{(i)}, y_3^{(i)})\}$:

$$\begin{array}{|c|c|c|c|} \hline x_1 & x_2 & x_3 & y_3 \\ \hline D'_1 & & & 1 \\ & & & 1 \\ & & & 0 \\ \hline D'_2 & & & 1 \\ & & & 0 \\ & & & 0 \\ \hline \end{array} \tag{4}$$

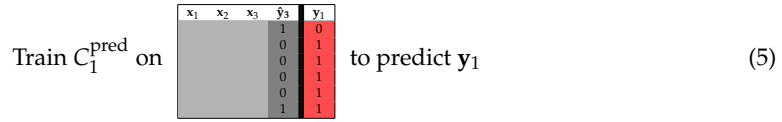
Two classifiers $C_{D'_1}$ and $C_{D'_2}$ are then trained on D'_1 and D'_2 , respectively, for the prediction of y_3 :

$$\text{Train } C_{D'_1} \text{ on } \begin{array}{|c|c|c|c|} \hline x_1 & x_2 & x_3 & y_3 \\ \hline & & & 1 \\ & & & 1 \\ & & & 0 \\ \hline \end{array} \text{ to predict } y_3, \quad \text{Train } C_{D'_2} \text{ on } \begin{array}{|c|c|c|c|} \hline x_1 & x_2 & x_3 & y_3 \\ \hline & & & 1 \\ & & & 0 \\ & & & 0 \\ \hline \end{array} \text{ to predict } y_3$$

Following the cross-validation paradigm, D'_1 is used as test set for the classifier $C_{D'_2}$, and D'_2 is used as a test set for $C_{D'_1}$:



These predictions are merged for the final predicted label \hat{y}_3 , which is used to augment the feature space. The classifier C_1^{pred} is then trained on that augmented feature space:



The prediction phase is completely analogous to (3). It is worthwhile to mention that the level-1 classifier C_3^{vl1} , which will be used to obtain predictions \hat{y}_3 at prediction time, is trained on the whole set $D' = D'_1 \cup D'_2$, following [Simon \(2007\)](#).

Full vs. partial conditioning

Recall the set of all labels $Z = \{z_1, \dots, z_m\}$. The prediction of a label z_k can either be conditioned on all remaining labels $\{z_1, \dots, z_{k-1}, z_{k+1}, \dots, z_m\}$ (*full conditioning*) or just on a subset of labels (*partial conditioning*). The only method for partial conditioning, which is examined in this paper, is the chaining method. Here, labels z_k are conditioned on all previous labels $\{z_1, \dots, z_{k-1}\}$ for all $k = 1, \dots, m$. This sequential structure is motivated by the product rule of probability ([Montañés et al., 2014](#)):

$$P(\mathbf{y}^{(i)} | \mathbf{x}^{(i)}) = \prod_{k=1}^m P(y_k^{(i)} | \mathbf{x}^{(i)}, y_1^{(i)}, \dots, y_{k-1}^{(i)}) \tag{6}$$

Methods that make use of this chaining structure are e.g., *classifier chains* or *nested stacking* (these methods will be discussed further below).

To sum up the discussions above: there are four ways in modeling label dependencies through conditioning labels z_k on other labels $z_\ell, k \neq \ell$. They can be distinguished by the subset of labels, which are used for conditioning, and by the use of predicted or real labels in the training step. In [Table 1](#) we show the four methods, which implement these ideas and describe them consequently.

	True labels	Pred. labels
Partial cond.	Classifier chains	Nested stacking
Full cond.	Dependent binary relevance	Stacking

Table 1: Distinctions in modeling label dependence and models

Classifier chains

The classifier chains (CC) method implements the idea of using partial conditioning together with the true label information. It was first introduced by [Read et al. \(2011\)](#). CC selects an order on the set of labels $\{z_1, \dots, z_m\}$, which can be formally written as a bijective function (permutation):

$$\tau : \{1, \dots, m\} \longrightarrow \{1, \dots, m\} \tag{7}$$

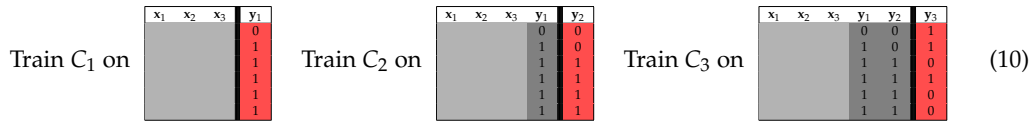
Labels will be chained along this order τ :

$$z_{\tau(1)} \rightarrow z_{\tau(2)} \rightarrow \dots \rightarrow z_{\tau(m)} \tag{8}$$

However, for this paper the permutation shall be $\tau = id$ (only for simplicity reasons). The labels therefore follow the order $z_1 \rightarrow z_2 \rightarrow \dots \rightarrow z_m$. In a similar fashion to the binary relevance (BR) method, CC trains m binary classifiers C_k , which are responsible for predicting their corresponding label $z_k, k = 1, \dots, m$. The classifiers C_k are of the form

$$C_k : \mathcal{X} \times \{0, 1\}^{k-1} \longrightarrow \{0, 1\}, \tag{9}$$

where $\{0, 1\}^0 := \emptyset$. For a classifier C_k the feature space is augmented by the true label information of all previous labels z_1, z_2, \dots, z_{k-1} . Hence, the training data of C_k consists of all observations $\left(\left(\mathbf{x}^{(i)}, y_1^{(i)}, y_2^{(i)}, \dots, y_{k-1}^{(i)} \right), y_k^{(i)} \right), i = 1, \dots, n$, with the target $y_k^{(i)}$. In the example from above, this would look like:



At prediction time, when an unlabeled observation $\mathbf{x}^{(l)}$ is labeled, a prediction $(\hat{y}_1^{(l)}, \dots, \hat{y}_m^{(l)})$ is obtained by successively predicting the labels along the chaining order:

$$\begin{aligned} \hat{y}_1^{(l)} &= C_1(\mathbf{x}^{(l)}) \\ \hat{y}_2^{(l)} &= C_2(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}) \\ &\vdots \\ \hat{y}_m^{(l)} &= C_m(\mathbf{x}^{(l)}, \hat{y}_1^{(l)}, \hat{y}_2^{(l)}, \dots, \hat{y}_{m-1}^{(l)}) \end{aligned} \tag{11}$$

The authors of [Senge et al. \(2013\)](#) summarize several factors, which have an impact on the performance of CC:

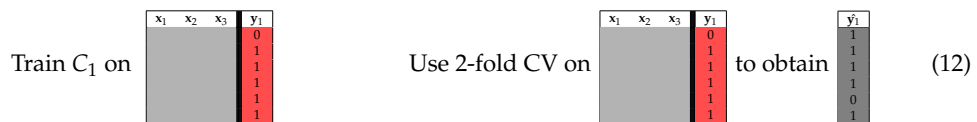
- *The length of the chain.* A high number $(k - 1)$ of preceding classifiers in the chain comes with a high potential level of feature noise for the classifier C_k . One may assume that the probability of a mistake will increase with the level of feature noise in the input space. Then the probability of a mistake will be reinforced along the chain, due to the recursive structure of CC.
- *The order of the chain.* Some labels may be more difficult to predict than others. The order of a chain can therefore be important for the performance. It can be advantageous to put simple to predict labels in the beginning and harder to predict labels more towards the end of the chain. Some heuristics for finding an optimal chain ordering have been proposed in [da Silva et al. \(2014\)](#); [Read et al. \(2013\)](#). Alternatively [Read et al. \(2011\)](#) developed an ensemble of classifier chains, which builds many randomly ordered CC-classifiers and put them on a voting scheme for a prediction. However, these methods are not subject of this article.
- *The dependency among labels.* For an improvement of performance through chaining, there should be a dependence among labels, CC cannot gain in case of label independence. However, CC is also only likely to lose if the binary classifiers C_k cannot ignore the added features $\mathbf{y}_1, \dots, \mathbf{y}_{k-1}$.

Nested stacking

The nested stacking method (NST), first proposed in [Senge et al. \(2013\)](#), implements the idea of using partial conditioning together with predicted label information. NST mimicks the chaining structure of CC, but does not use real label information during training. Like in CC the chaining order shall be $\tau = id$, again for simplicity reasons. CC uses real label information \mathbf{y}_k during training and predicted labels $\hat{\mathbf{y}}_k$ at prediction time. However, unless the binary classifiers are perfect, it is likely that \mathbf{y}_k and $\hat{\mathbf{y}}_k$ do not follow the same distribution. Hence, the key assumption of supervised learning, namely that the training data should be representative for the test data, is violated by CC. Nested stacking tries to overcome this issue by using predicted labels $\hat{\mathbf{y}}_k$ instead of true labels \mathbf{y}_k .

NST trains m binary classifiers C_k on $D_k := \cup_{i=1}^n \left\{ \left(\left(\mathbf{x}^{(i)}, \hat{y}_1^{(i)}, \dots, \hat{y}_{k-1}^{(i)} \right), y_k^{(i)} \right) \right\}$, for all $k = 1, \dots, m$. The predicted labels should be obtained by an internal out-of-sample method ([Senge et al., 2013](#)). How these predictions are obtained was already explained in the **True vs. Predicted Label Information** chapter. The prediction phase is completely analogous to (11).

The training procedure is visualized in the following with 2-fold cross-validation as an internal out-of-sample method:



Train C_2 on

x_1	x_2	x_3	\hat{y}_1	y_2
			1	0
			1	0
			1	1
			1	1
			0	1
			1	1

 Use 2-fold CV on

x_1	x_2	x_3	\hat{y}_1	y_2
			1	0
			1	0
			1	1
			1	1
			0	1
			1	1

 to obtain

\hat{y}_2
1
1
1
1
0
1
1

 (13)

Train C_3 on

x_1	x_2	x_3	\hat{y}_1	\hat{y}_2	y_3
			1	1	1
			1	1	1
			1	1	0
			1	0	1
			0	1	0
			1	0	0

 (14)

The factors which impact the performance of CC (i.e., length and order of the chain, and the dependency among labels), also impact NST, since NST mimicks the chaining method of CC.

Dependent binary relevance

The dependent binary relevance method (DBR) implements the idea of using full conditioning together with the true label information. DBR is built on two main hypotheses (Montañés et al., 2014):

- (i) Taking conditional label dependencies into account is important for performing well in multilabel classification tasks.
- (ii) Modeling and learning these label dependencies in an overcomplete way (take all other labels for modeling) may further improve model performance.

The first assumption is the main prerequisite for research in multilabel classification. It has been shown theoretically that simple binary relevance classifiers cannot achieve optimal performance for specific multilabel loss functions (Montañés et al., 2014). The second assumption, however, is harder to justify theoretically. Nonetheless, the practical usefulness of learning in an overcomplete way has been shown in many branches of (classical) single-label classification (e.g., ensemble methods (Dietterich, 2000)).

Formally, DBR trains m binary classifiers C_1, \dots, C_m (as many classifiers as labels) on the corresponding training data

$$D_k = \cup_{i=1}^n \left\{ \left(\left(\mathbf{x}^{(i)}, y_1^{(i)}, \dots, y_{k-1}^{(i)}, y_{k+1}^{(i)}, \dots, y_m^{(i)} \right), y_k^{(i)} \right) \right\}, \tag{15}$$

$k = 1, \dots, m$. Thus, each classifier C_k is of the form

$$C_k : \mathcal{X} \times \{0, 1\}^{m-1} \longrightarrow \{0, 1\}.$$

Hence, for each classifier C_k the true label information of all labels except y_k is used as augmented features. Again, here is a visualization with the example from above:

Train C_1 on

x_1	x_2	x_3	y_2	y_3	y_1
			0	1	0
			0	1	1
			1	0	1
			1	1	1
			1	0	1
			1	0	1

 Train C_2 on

x_1	x_2	x_3	y_1	y_3	y_2
			0	1	0
			1	1	0
			1	0	1
			1	1	1
			1	0	1
			1	0	1

 Train C_3 on

x_1	x_2	x_3	y_1	y_2	y_3
			0	0	1
			1	1	0
			1	1	1
			1	1	1
			1	1	0
			1	1	0

 (16)

To make these classifiers applicable, when an unlabeled instance $\mathbf{x}^{(l)}$ needs to be labeled, the help of other multilabel classifiers is needed to produce predicted labels $\hat{y}_1^{(l)}, \dots, \hat{y}_m^{(l)}$ as additional features. The classifiers, which produce predicted labels as additional features, are called *base learners* (Montañés et al., 2014). Theoretically any multilabel classifier can be used as base learner. However, in this paper, the analysis is focused on BR as base learner only. The prediction of an unlabeled instance $\mathbf{x}^{(l)}$ formally works as follows:

- (i) First level: Produce predicted labels by using the BR base learner:

$$C_{BR}(\mathbf{x}^{(l)}) = (\hat{y}_1^{(l)}, \dots, \hat{y}_m^{(l)})$$

- (ii) Second level, which is also called meta level (Montañés et al., 2014): Produce final prediction $\hat{y}_k = (\hat{y}_1^{(l)}, \dots, \hat{y}_m^{(l)})$ by applying DBR classifiers C_1, \dots, C_m :

$$\begin{aligned}
 C_1(x^{(l)}, \hat{y}_2^{(l)}, \dots, \hat{y}_m^{(l)}) &= \hat{y}_1^{(l)} \\
 C_2(x^{(l)}, \hat{y}_1^{(l)}, \hat{y}_3^{(l)}, \dots, \hat{y}_m^{(l)}) &= \hat{y}_2^{(l)} \\
 &\vdots \\
 C_m(x^{(l)}, \hat{y}_1^{(l)}, \dots, \hat{y}_{m-1}^{(l)}) &= \hat{y}_m^{(l)}
 \end{aligned}$$

Stacking

Stacking (STA) implements the last variant of Table 1, namely the use of full conditioning together with predicted label information. Stacking is short for *stacked generalization* (Wolpert, 1992) and was first proposed in the multilabel context by Godbole and Sarawagi (2004). Like in classical stacking, for each label it takes predictions of several other learners that were trained in a first step to get a new learner to make predictions for the corresponding label. Both hypotheses on which DBR is built on also apply to STA, of course.

STA trains m classifiers C_1, \dots, C_m on the corresponding training data

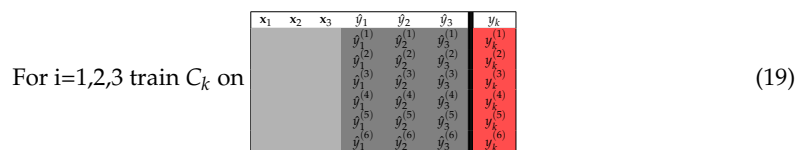
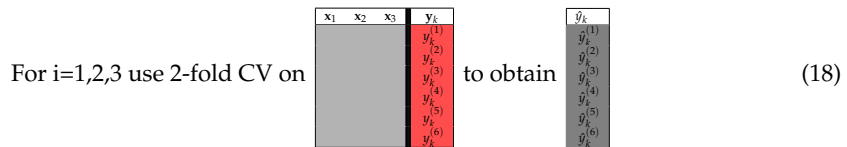
$$D_k = \cup_{i=1}^n \left\{ \left((x^{(i)}, \hat{y}_1^{(i)}, \dots, \hat{y}_m^{(i)}), y_k^{(i)} \right) \right\}, k = 1, \dots, m. \tag{17}$$

The classifiers $C_k, k = 1, \dots, m$, are therefore of the following form:

$$C_k : \mathcal{X} \times \{0,1\}^m \longrightarrow \{0,1\}$$

Like in NST, the predicted labels should be obtained by an internal out-of-sample method (Sill et al., 2009). STA can be seen as the alternative to DBR using predicted labels (like NST is for CC). However, the classifiers $C_k, k = 1, \dots, m$, are trained on all predicted labels $\hat{y}_1, \dots, \hat{y}_m$ for the STA approach (in DBR the label y_k is left out of the augmented training set).

The training procedure is outlined in the following:



Like in DBR, STA depends on a BR base learner, to produce predicted labels as additional features. Again, the use of BR as a base learner is not mandatory, but it is the proposed method in Godbole and Sarawagi (2004).

The prediction of an unlabeled instance $x^{(l)}$ works almost identically to the DBR case and is illustrated here:

- (i) First level. Produce predicted labels by using the BR base learner:

$$C_{BR}(x^{(l)}) = (\hat{y}_1^{(l)}, \dots, \hat{y}_m^{(l)})$$

- (ii) Meta level. Apply STA classifiers C_1, \dots, C_m :

$$\begin{aligned}
 C_1(x^{(l)}, \hat{y}_1^{(l)}, \dots, \hat{y}_m^{(l)}) &= \hat{y}_1^{(l)} \\
 &\vdots \\
 C_m(x^{(l)}, \hat{y}_1^{(l)}, \dots, \hat{y}_m^{(l)}) &= \hat{y}_m^{(l)}
 \end{aligned}$$

Multilabel performance measures

Analogously to multiclass classification there exist multilabel classification performance measures. Six multilabel performance measures can be evaluated in **mlr**. These are: *Subset 0/1 loss*, *hamming loss*, *accuracy*, *precision*, *recall* and *F₁-index*. Multilabel performance measures are defined on a per instance basis. The performance on a test set is the average over all instances.

Let $D_{\text{test}} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)} \right), \dots, \left(\mathbf{x}^{(n)}, \mathbf{y}^{(n)} \right) \right\}$ be a test set with $\mathbf{y}^{(i)} = \left(y_1^{(i)}, \dots, y_m^{(i)} \right) \in \{0, 1\}^m$ for all $i = 1, \dots, n$. Performance measures quantify how good a classifier C predicts the labels z_1, \dots, z_n .

- (i) The subset 0/1 loss is used to see if the predicted labels $C(\mathbf{x}^{(i)}) = \left(\hat{y}_1^{(i)}, \dots, \hat{y}_m^{(i)} \right)$ are equal to the actual labels $\left(y_1^{(i)}, \dots, y_m^{(i)} \right)$:

$$\text{subset}_{0/1} \left(C, \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right) \right) = \mathbb{1}_{\left(\mathbf{y}^{(i)} \neq C(\mathbf{x}^{(i)}) \right)} := \begin{cases} 1 & \text{if } \mathbf{y}^{(i)} \neq C \left(\mathbf{x}^{(i)} \right) \\ 0 & \text{if } \mathbf{y}^{(i)} = C \left(\mathbf{x}^{(i)} \right) \end{cases}$$

The subset 0/1 loss of a classifier C on a test set D_{test} thus becomes:

$$\text{subset}_{0/1} \left(C, D_{\text{test}} \right) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\mathbf{y}^{(i)} \neq C(\mathbf{x}^{(i)})}$$

The subset 0/1 loss can be interpreted as the analogon of the mean misclassification error in multiclass classifications. In the multilabel case it is a rather drastic measure because it treats a mistake on a single label as a complete failure (Senge et al., 2013).

- (ii) The hamming loss also takes into account observations where only some labels have been predicted correctly. It corresponds to the proportion of labels whose relevance is incorrectly predicted. For an instance $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right) = \left(\mathbf{x}^{(i)}, \left(y_1^{(i)}, \dots, y_m^{(i)} \right) \right)$ and a classifier $C \left(\mathbf{x}^{(i)} \right) = \left(\hat{y}_1^{(i)}, \dots, \hat{y}_m^{(i)} \right)$ this is defined as:

$$\text{HammingLoss} \left(C, \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right) \right) = \frac{1}{m} \sum_{k=1}^m \mathbb{1}_{\left(y_k^{(i)} \neq \hat{y}_k^{(i)} \right)}$$

If one label is predicted incorrectly, this accounts for an error of $\frac{1}{m}$. For a test set D_{test} the hamming loss becomes:

$$\text{HammingLoss} \left(C, D_{\text{test}} \right) = \frac{1}{n} \sum_{i=1}^n \frac{1}{m} \sum_{k=1}^m \mathbb{1}_{\left(y_k^{(i)} \neq \hat{y}_k^{(i)} \right)}$$

The following measures are scores instead of loss function like the two previous ones.

- (iii) The accuracy, also called Jaccard-Index, for a test set D_{test} is defined as:

$$\text{accuracy} \left(C, D_{\text{test}} \right) = \frac{1}{n} \sum_{i=1}^n \frac{\sum_{k=1}^m \mathbb{1}_{\left(y_k^{(i)} = 1 \text{ and } \hat{y}_k^{(i)} = 1 \right)}}{\sum_{k=1}^m \mathbb{1}_{\left(y_k^{(i)} = 1 \text{ or } \hat{y}_k^{(i)} = 1 \right)}}$$

- (iv) The precision for a test set D_{test} is defined as:

$$\text{precision} \left(C, D_{\text{test}} \right) = \frac{1}{n} \sum_{i=1}^n \frac{\sum_{k=1}^m \mathbb{1}_{\left(y_k^{(i)} = 1 \text{ and } \hat{y}_k^{(i)} = 1 \right)}}{\sum_{k=1}^m \mathbb{1}_{\left(\hat{y}_k^{(i)} = 1 \right)}}$$

- (v) The recall for a test set D_{test} is defined as:

$$\text{recall} \left(C, D_{\text{test}} \right) = \frac{1}{n} \sum_{i=1}^n \frac{\sum_{k=1}^m \mathbb{1}_{\left(y_k^{(i)} = 1 \text{ and } \hat{y}_k^{(i)} = 1 \right)}}{\sum_{k=1}^m \mathbb{1}_{\left(y_k^{(i)} = 1 \right)}}$$

- (vi) For a test set D_{test} the F₁-index is defined as follows:

$$F_1 \left(C, D_{\text{test}} \right) = \frac{1}{n} \sum_{i=1}^n \frac{2 \sum_{k=1}^m \mathbb{1}_{\left(y_k^{(i)} = 1 \text{ and } \hat{y}_k^{(i)} = 1 \right)}}{\sum_{k=1}^m \left(\mathbb{1}_{\left(y_k^{(i)} = 1 \right)} + \mathbb{1}_{\left(\hat{y}_k^{(i)} = 1 \right)} \right)}$$

The F_1 -index is the harmonic mean of recall and precision on a per instance basis.

All these measures lie between 0 and 1. In the case of the subset 0/1 loss and the hamming loss the values should be low, in all other cases the scores should be high. Demonstrative definitions with sets instead of vectors can be seen in [Charte and Charte \(2015\)](#).

Implementation

In this section, we briefly describe how to perform multilabel classifications in **mlr**. We provide small code examples for better illustration. A short tutorial is also available at <http://mlr-org.github.io/mlr-tutorial/release/html/multilabel/index.html>. The first step is to transform the multilabel dataset into a 'data.frame' in R. The columns must consist of vectors of features and one logical vector for each label that indicates if the label is present for the observation or not. To fit a multilabel classification algorithm in **mlr**, a multilabel task has to be created, where a vector of targets corresponding to the column names of the labels has to be specified. This task is an S3 object that contains the data, the target labels and further descriptive information. In the following example, the yeast data frame is extracted from the yeast.task, which is provided by the **mlr** package. Then the 14 label names of the targets are extracted and the multilabel task is created.

```
yeast = getTaskData(yeast.task)
labels = colnames(yeast)[1:14]
yeast.task = makeMultilabelTask(id = "multi", data = yeast, target = labels)
```

Problem transformation methods

To generate a problem transformation method learner, a binary classification base learner has to be created with 'makeLearner'. A list of available learners for classifications in **mlr** can be seen at http://mlr-org.github.io/mlr-tutorial/release/html/integrated_learners/. Specific hyperparameter settings of the base learner can be set in this step through the 'par.vals' argument in 'makeLearner'. Afterwards, a learner for any problem transformation method can be created by applying the function 'makeMultilabel[...].Wrapper', where [...] has to be substituted by the desired problem transformation method. In the following example, two multilabel variants with rpart as base learner are created. The base learner is configured to output probabilities instead of discrete labels during prediction.

```
lrn = makeLearner("classif.rpart", predict.type = "prob")
multilabel.lrn1 = makeMultilabelBinaryRelevanceWrapper(lrn)
multilabel.lrn2 = makeMultilabelNestedStackingWrapper(lrn)
```

Algorithm adaptation methods

Algorithm adaptation method learners can be created directly with 'makeLearner'. The names of the specific learner can be looked up at http://mlr-org.github.io/mlr-tutorial/release/html/integrated_learners/ in the multilabel section.

```
multilabel.lrn3 = makeLearner("multilabel.rFerns")
multilabel.lrn4 = makeLearner("multilabel.randomForestSRC")
```

Train, predict and evaluate

Training and predicting on data can be done as usual in **mlr** with the functions 'train' and 'predict'. Learner and task have to be specified in 'train'; trained model and task or new data have to be specified in 'predict'.

```
mod = train(multilabel.lrn1, yeast.task, subset = 1:1500)
pred = predict(mod, task = yeast.task, subset = 1501:1600)
```

The performance of the prediction can be assessed via the function 'performance'. Measures are represented as S3 objects and multiple objects can be passed in as a list. The default measure for multilabel classification is the hamming loss (*multilabel.hamloss*). All available measures for multilabel classification can be shown by 'listMeasures' or looked up in the appendix of the tutorial page¹ (<http://mlr-org.github.io/mlr-tutorial/release/html/measures/index.html>).

¹In the **mlr** package *precision* is named *positive predictive value* and *recall* is named *true positive rate*.


```

performance(pred, measures = list(multilabel.hamloss, timepredict))
multilabel.hamloss      timepredict
0.230                   0.174
listMeasures("multilabel")
# [1] "multilabel.ppv" "timepredict"      "multilabel.hamloss" "multilabel.f1"
# [5] "featperc"       "multilabel.subset01" "timeboth"          "timetrain"
# [9] "multilabel.tpr" "multilabel.acc"

```

Resampling

To properly evaluate the model, a resampling strategy, for example k-fold cross-validation, should be applied. This can be done in `mlr` by using the function `'resample'`. First, a description of the subsequent resampling strategy, in this case three-fold cross-validation, is defined with `'makeResampleDesc'`. The resample is executed by a call to the `'resample'` function. The hamming loss is calculated for the binary relevance method.

```

rdesc = makeResampleDesc(method = "CV", stratify = FALSE, iters = 3)
r = resample(learner = multilabel.lrn1, task = yeast.task, resampling = rdesc,
measures = list(multilabel.hamloss), show.info = FALSE)
r
# Resample Result
# Task: multi
# Learner: multilabel.classif.rpart
# multilabel.hamloss.aggr: 0.23
# multilabel.hamloss.mean: 0.23
# multilabel.hamloss.sd: 0.00
# Runtime: 6.36688

```

Binary performance

To calculate a binary performance measure like, e.g., the accuracy, the mean misclassification error (mmce) or the AUC for each individual label, the function `'getMultilabelBinaryPerformances'` can be used. This function can be applied to a single multilabel test set prediction and also on a resampled multilabel prediction. To calculate the AUC, predicted probabilities are needed. These can be obtained by setting the argument `'predict.type = "prob"'` in the `'makeLearner'` function.

```

head(getMultilabelBinaryPerformances(r$pred, measures = list(acc, mmce, auc)))
#      acc.test.mean mmce.test.mean auc.test.mean
# label1      0.7389326      0.2610674      0.6801810
# label2      0.5908151      0.4091849      0.5935160
# label3      0.6512205      0.3487795      0.6631469
# label4      0.6921804      0.3078196      0.6965552
# label5      0.7517584      0.2482416      0.6748458
# label6      0.7343815      0.2656185      0.6054968

```

Parallelization

In the case of a high number of labels and larger datasets, parallelization in the training and prediction process of the multilabel methods can reduce computation time. This can be achieved by using the package `parallelMap` in `mlr` (see also the tutorial section of parallelization: <http://mlr-org.github.io/mlr-tutorial/release/html/multilabel/index.html>). Currently, only the binary relevance method is parallelizable, the classifier for each label is trained in parallel, as they are independent of each other. The other problem transformation methods will also be parallelizable (as far as possible) soon.

```

library(parallelMap)
parallelStartSocket(2)
lrn = makeMultilabelBinaryRelevanceWrapper("classif.rpart")
mod = train(lrn, yeast.task)
pred = predict(mod, yeast.task)

```

Benchmark experiment

In a similar fashion to Wang et al. (2014), we performed a benchmark experiment on several datasets in order to compare the performances of the different multilabel algorithms.

Datasets: In Table 2 we provide an overview of the used datasets. We retrieved most datasets from the Mulan Java library for multilabel learning² as well as from other benchmark experiments of multilabel classification methods. See Table 2 for article references. We uploaded all datasets to the open data platform OpenML (Casalicchio et al., 2017; Vanschoren et al., 2013), so they now can be downloaded directly from there. In some of the used datasets, sparse labels had to be removed in order to avoid problems during cross-validation. Several binary classification methods have difficulties when labels are sparse, i.e., a strongly imbalanced binary target class can lead to constant predictions for that target. That can sometimes lead to direct problems in the base learners (when training on constant class labels is simply not allowed) or, e.g., in classifier chains, when the base learner cannot handle constant features. Furthermore, one can reasonably argue that not much is to be learned for such a label. Hence, labels that appeared in less than 2% of the observations were removed. We computed *cardinality* scores (based on the remaining labels) indicating the mean number of labels assigned to each case in the respective dataset. The following description of the datasets refers to the final versions after removal of sparse labels.

- The first dataset (*birds*) consists of 645 audio recordings of 15 different vocalizing bird species (Briggs et al., 2013). Each sound can be assigned to various bird species.
- Another audio dataset (*emotions*) consists of 593 musical files with 6 clustered emotional labels (Trohidis et al., 2008) and 72 predictors. Each song can be labeled with one or more of the labels {*amazed-surprised, happy-pleased, relaxing-calm, quiet-still, sad-lonely, angry-fearful*}.
- The *genbase* dataset contains protein sequences that can be assigned to several classes of protein families (Diplaris et al., 2005). The entire dataset contains 1186 binary predictors.
- The *langLog*³ dataset includes 998 textual predictors and was originally compiled in the doctoral thesis of Read (2010). It consists of 1460 text samples that can be assigned to one or more topics such as *language, politics, errors, humor* and *computational linguistics*.
- The UC Berkeley *enron*⁴ dataset represents a subset of the original *enron*⁵ dataset and consists of 1702 cases of emails with 24 labels and 1001 predictor variables (Klimt and Yang, 2004).
- A subset of the *reuters*⁶ dataset includes 2000 observations for text classification (Zhang and Zhou, 2008).
- The *image*⁷ benchmark dataset consists of 2000 natural scene images. Zhou and ling Zhang (2007) extracted 135 features for each image and made it publicly available as *processed* image dataset. Each observation can be associated with different label sets, where all possible labels are {*desert, mountains, sea, sunset, trees*}. About 22% of the images belong to more than one class. However, images belonging to three classes or more are very rare.
- The *scene* dataset is an image classification task where labels like *Beach, Mountain, Field, Urban* are assigned to each image (Boutell et al., 2004).
- The *yeast* dataset (Elisseeff and Weston, 2002) consists of micro-array expression data, as well as phylogenetic profiles of yeast, and includes 2417 genes and 103 predictors. In total, 14 different labels can be assigned to a gene, but only 13 labels were used due to label sparsity.
- Another dataset for text-classification is the *slashdot*⁸ dataset (Read et al., 2011). It consists of article titles and partial blurbs. Blurbs can be assigned to several categories (e.g., *Science, News, Games*) based on word predictors.

Algorithms: We used all multilabel classification methods currently implemented in **mlr**: binary relevance (BR), classifier chains (CC), nested stacking (NST), dependent binary relevance (DBR) and stacking (STA) as well as algorithm adaption methods of the **rFerns** (RFERN) and **randomForestSRC** (RFSRC) packages. For DBR and STA the first level and meta level classifiers were equal. For CC and NST we chose random chain orders for each resample iteration.

²<http://mulan.sourceforge.net/datasets-mlc.html>

³<http://languagelog ldc.upenn.edu/nll/>

⁴http://bailando.sims.berkeley.edu/enron_email.html

⁵<http://www.cs.cmu.edu/~enron/>

⁶http://lamda.nju.edu.cn/data_MIMLtext.ashx

⁷http://lamda.nju.edu.cn/data_MIMLimage.ashx

⁸<http://slashdot.org>

Dataset	Reference	# Inst.	# Pred.	# Labels	Cardinality
birds*	Briggs et al. (2013)	645	260	15	0.96
emotions	Trohidis et al. (2008)	593	72	6	1.87
genbase*	Diplaris et al. (2005)	662	112	16	1.20
langLog*	Read (2010)	1460	998	18	0.85
enron*	Klimt and Yang (2004)	1702	1001	24	3.12
reuters	Zhang and Zhou (2008)	2000	243	7	1.15
image	Zhou and ling Zhang (2007)	2000	135	5	1.24
scene	Boutell et al. (2004)	2407	294	6	1.07
yeast*	Elisseff and Weston (2002)	2417	103	13	4.22
slashdot*	Read et al. (2011)	3782	1079	14	1.13

Table 2: Used benchmark datasets including number of instances, number of predictor, number of label and label cardinality. Datasets with an asterisk differ from the original dataset as sparse labels have been removed. The genbase dataset contained many constant factor variables, which were automatically removed by mlr.

Base Learners: We employed two different binary classification base learner for each problem transformation algorithm: random forest (rf) of the **randomForest** package (Liaw and Wiener, 2002) with `ntree = 100` and adaboost (ad) from the **ada** package (Culp et al., 2012), each with standard hyperparameter settings.

Performance Measures: We used the six previously proposed performance measures. Furthermore, we calculated the reported values by means of a 10-fold cross-validation.

Code: For reproducibility, the complete code and results can be downloaded from Probst (2017). The R package **batchtools** (Bischl et al., 2015) was used for parallelization.

The results for hamming loss and F_1 -index are illustrated in Figure 1. Tables 3 and 4 contain performance values with the best performing algorithms highlighted in blue. For all remaining measures one may refer to the Appendix. We did not perform any threshold tuning that would potentially improve some of the performance of the methods.

The results of the problem transformation methods in this benchmark experiment concur with the general conclusions and results in Montañés et al. (2014). The authors ran a similar benchmark study with penalized logistic regression as base learner. They concluded that, on average, DBR performs well in F_1 and accuracy. Also, CC outperform the other methods regarding the subset 0/1 loss most of the time. For the hamming loss measure they got mixed results, with no clear winner concordant to our benchmark results. As base learner, on average, adaboost performs better than random forest in our benchmark study.

Considering the measure F_1 , the problem transformation methods DBR, CC, STA and NST outperform RFERN and RFSRC on most of the datasets and also almost always perform better than BR, which does not consider dependencies among the labels. RFSRC and RFERN only perform well on either precision or recall, but in order to be considered as good classifiers they should perform well on both. The generally poor performances of RFERN can be explained by the working mechanism of the algorithm which randomly chooses variables and split points at each split of a fern. Hence, it cannot deal with too many features that are useless for the prediction of the target labels.

Summary

In this paper, we describe the implementation of multilabel classification algorithms in the R package **mlr**. The problem transformation methods binary relevance, classifier chains, nested stacking, dependent binary relevance and stacking are implemented and can be used with any base learner that is accessible in **mlr**. Moreover, there is access to the multilabel classification versions of **randomForestSRC** and **RFerns**. We compare all of these methods in a benchmark experiment with several datasets and different implemented multilabel performance measures. The dependent binary relevance method performs well regarding the measures F_1 and *accuracy*. Classifier chains outperform the other methods in terms of the subset 0/1 loss most of the time. Parallelization is available for the binary relevance method and will be available soon for the other problem transformation methods. Algorithm adaptation methods and problem transformation methods that are currently not available can be incorporated in the current **mlr** framework easily. In our benchmark experiment we had to remove labels which occurred too sparsely, because some algorithms crashed due to one class problems, which appeared during cross-validation. A solution to this problem and an implementation into the **mlr** framework is of great interest.

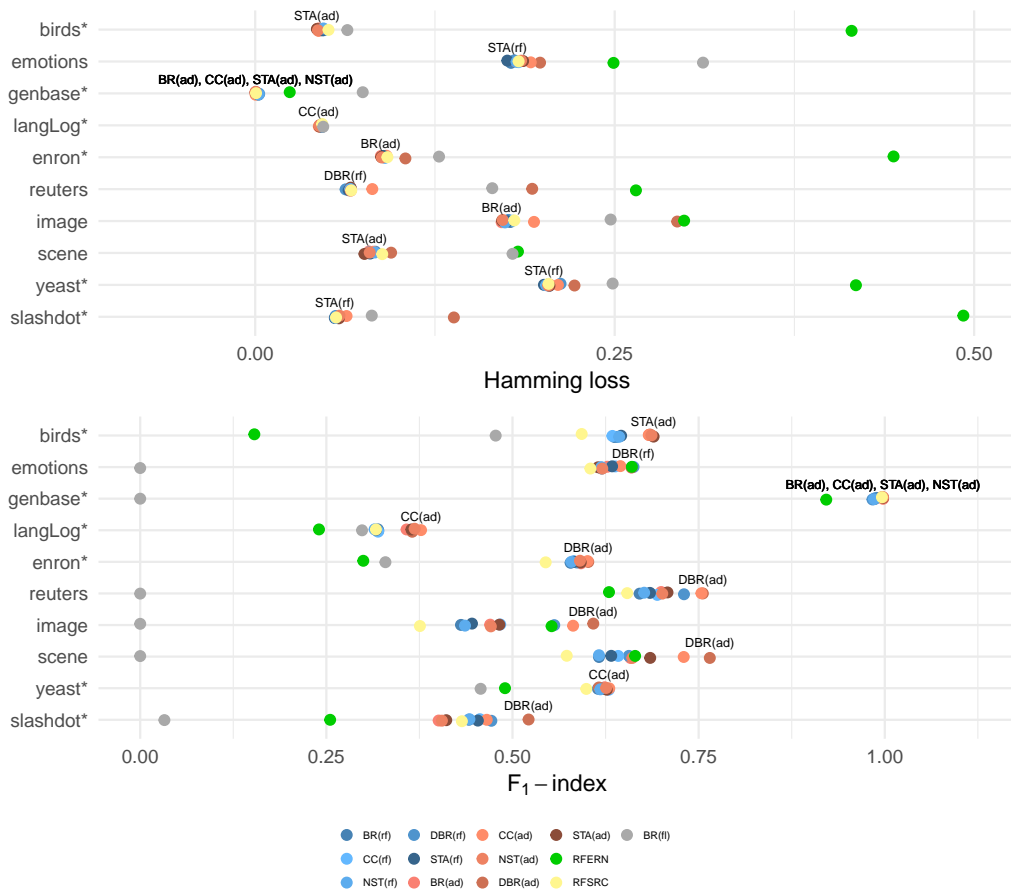


Figure 1: Results for hamming loss and F₁-index. The best performing algorithms are highlighted on the plot.

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(f)
birds*	0.0477	0.0479	0.0475	0.0472	0.0468	0.0442	0.0441	0.0436	0.0431	0.0429	0.4148	0.0510	0.0641
emotions	0.1779	0.1832	0.1818	0.1801	0.1753	0.181	0.1916	0.1849	0.1981	0.1863	0.2492	0.1832	0.3114
genbase*	0.0021	0.0023	0.0025	0.0027	0.0023	0.0003	0.0003	0.0003	0.0004	0.0003	0.0240	0.0006	0.0748
langLog*	0.0464	0.0465	0.0467	0.0464	0.0466	0.0451	0.0442	0.0446	0.0447	0.0448	0.6673	0.0466	0.0473
enron*	0.0903	0.0904	0.0902	0.0909	0.0891	0.0874	0.0913	0.0881	0.1045	0.0877	0.4440	0.0919	0.1279
reuters	0.0663	0.0654	0.0661	0.0629	0.065	0.0666	0.0814	0.0664	0.1926	0.0664	0.2648	0.0668	0.1649
image	0.1774	0.1791	0.1737	0.1761	0.1754	0.1714	0.1939	0.1721	0.2935	0.1717	0.2983	0.1802	0.2472
scene	0.0836	0.0809	0.0832	0.0796	0.0799	0.0791	0.0821	0.0796	0.0945	0.076	0.1827	0.0884	0.1790
yeast*	0.2038	0.2044	0.2023	0.2123	0.2008	0.2048	0.2105	0.2038	0.2221	0.2046	0.4178	0.2040	0.2486
slashdot*	0.0558	0.0560	0.0559	0.0559	0.0554	0.059	0.0635	0.0586	0.1382	0.0582	0.4925	0.0562	0.0811

Table 3: Hamming loss

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(f)
birds*	0.6369	0.6342	0.6433	0.64	0.6459	0.6835	0.683	0.6867	0.6846	0.6895	0.1533	0.5929	0.4774
emotions	0.6199	0.6380	0.6192	0.6625	0.6337	0.6274	0.6449	0.6206	0.6598	0.615	0.6603	0.6046	0.0000
genbase*	0.9885	0.9861	0.9855	0.9835	0.9861	0.9977	0.9977	0.9977	0.9962	0.9977	0.9214	0.9962	0.0000
langLog*	0.3192	0.3194	0.3148	0.3199	0.3167	0.3578	0.3772	0.3686	0.3653	0.3643	0.2401	0.3167	0.2979
enron*	0.5781	0.5822	0.5791	0.5866	0.5826	0.592	0.6009	0.5906	0.6017	0.5917	0.2996	0.5446	0.3293
reuters	0.6708	0.6944	0.6769	0.7303	0.6846	0.6997	0.7537	0.7012	0.7556	0.7082	0.6296	0.6541	0.0000
image	0.4308	0.4835	0.4362	0.5561	0.4456	0.47	0.5814	0.4709	0.6085	0.4824	0.5525	0.3757	0.0000
scene	0.6161	0.6420	0.6161	0.6563	0.6326	0.6585	0.73	0.661	0.765	0.685	0.6647	0.5729	0.0000
yeast*	0.6148	0.6294	0.6180	0.6195	0.6244	0.6238	0.63	0.6257	0.616	0.6266	0.4900	0.5991	0.4572
slashdot*	0.4415	0.4562	0.4422	0.4716	0.4535	0.4009	0.4654	0.4052	0.5216	0.411	0.2551	0.4320	0.0325

Table 4: F₁-index

Bibliography

- B. Bischl, M. Lang, O. Mersmann, J. Rahnenführer, and C. Weihs. BatchJobs and BatchExperiments: Abstraction mechanisms for using R in batch environments. *Journal of Statistical Software*, 64(11): 1–25, 2015. URL <https://doi.org/10.18637/jss.v064.i11>. [p363]
- B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. Mr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016. [p352]
- M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004. URL <https://doi.org/10.1016/j.patcog.2004.03.009>. [p352, 362, 363]
- F. Briggs, H. Yonghong, R. Raich, and others. New methods for acoustic classification of multiple simultaneous bird species in a noisy environment. In *IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–8, 2013. URL <https://doi.org/10.1109/mlsp.2013.6661934>. [p352, 362, 363]
- G. Casalicchio, J. Bossek, M. Lang, D. Kirchoff, P. Kerschke, B. Hofner, H. Seibold, J. Vanschoren, and B. Bischl. OpenML: An R package to connect to the networked machine learning platform OpenML. *ArXiv e-prints*, 2017. [p362]
- F. Charte and D. Charte. Working with multilabel datasets in R: The mldr package. *The R Journal*, 7(2): 149–162, 2015. [p352, 360]
- M. Culp, K. Johnson, and G. Michailidis. *ada: An R Package for Stochastic Boosting*, 2012. URL <https://cran.r-project.org/package=ada>. [p363]
- P. N. da Silva, E. C. Gonçalves, A. Plastino, and A. A. Freitas. Distinct chains for different instances: An effective strategy for multi-label classifier chains. In *European Conference, ECML PKDD 2014*, pages 453–468, 2014. URL https://doi.org/10.1007/978-3-662-44851-9_29. [p356]
- T. G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000. URL https://doi.org/10.1007/3-540-45014-9_1. [p357]
- S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas. Protein classification with multiple algorithms. In *Advances in Informatics*, pages 448–456. Springer-Verlag, 2005. URL https://doi.org/10.1007/11573036_42. [p362, 363]
- A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 681–687. MIT Press, 2002. [p352, 362, 363]
- S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *Advances in Knowledge Discovery and Data*, volume LNCS3056, pages 22–30, 2004. URL https://doi.org/10.1007/978-3-540-24775-3_5. [p358]
- H. Ishwaran and U. B. Kogalur. *Random Forests for Survival, Regression and Classification (RF-SRC)*, 2016. URL <http://cran.r-project.org/package=randomForestSRC>. [p353]
- B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. *Machine Learning: ECML 2004*, pages 217–226, 2004. URL https://doi.org/10.1007/978-3-540-30115-8_22. [p362, 363]
- M. B. Kursa and A. A. Wiecekowska. Multi-label ferns for efficient recognition of musical instruments in recordings. In *International Symposium on Methodologies for Intelligent Systems*, pages 214–223. Springer, 2014. URL https://doi.org/10.1007/978-3-319-08326-1_22. [p352, 353]
- M. Lang, H. Kotthaus, P. Marwedel, C. Weihs, J. Rahnenführer, and B. Bischl. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation*, 85(1):62–76, 2015. URL <https://doi.org/10.1080/00949655.2014.929131>. [p352]
- A. Liaw and M. Wiener. Classification and regression by randomForest. *R News: The Newsletter of the R Project*, 2(3):18–22, 2002. [p363]
- A. McCallum. Multi-label text classification with a mixture model trained by EM. *AAAI'99 Workshop on Text Learning*, pages 1–7, 1999. [p352]

- E. Montañés, R. Senge, J. Barranquero, J. R. Quevedo, J. J. del Coz, and E. Hüllermeier. Dependent binary relevance models for multi-label classification. *Pattern Recognition*, 47(3):1494–1508, 2014. URL <https://doi.org/10.1016/j.patcog.2013.09.029>. [p353, 354, 355, 357, 358, 363]
- P. Probst. Multilabel classification with R package mlr. figshare. Code may be downloaded here, 2017. URL <https://doi.org/10.6084/m9.figshare.3384802.v5>. [p363]
- J. Read. Scalable multi-label classification. *Hamilton, New Zealand: University of Waikato*, 2010. [p362, 363]
- J. Read and P. Reutemann. Meka: A multi-label extension to WEKA, 2012. URL <http://meka.sourceforge.net/>. [p352]
- J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85:333–359, 2011. URL <https://doi.org/10.1007/s10994-011-5256-5>. [p355, 356, 362, 363]
- J. Read, L. Martino, and D. Luengo. Efficient Monte Carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, (Mdc):1–36, 2013. URL <https://doi.org/10.1016/j.patcog.2013.10.006>. [p356]
- C. Sanden and J. Z. Zhang. Enhancing multi-label music genre classification through ensemble techniques. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 705–714, 2011. URL <https://doi.org/10.1145/2009916.2010011>. [p352]
- R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39:135–168, 2000. [p352]
- R. Senge, J. J. del Coz Velasco, and E. Hüllermeier. Rectifying classifier chains for multi-label classification. *Space*, 2 (8), 2013. [p354, 356, 359]
- J. Sill, G. Takacs, L. Mackey, and D. Lin. Feature-Weighted Linear Stacking. *ArXiv e-prints*, 2009. [p358]
- R. Simon. Resampling strategies for model assessment and selection. In W. Dubitzky, M. Granzow, and D. Berrar, editors, *Fundamentals of Data Mining in Genomics and Proteomics SE - 8*, pages 173–186. Springer-Verlag, 2007. URL https://doi.org/10.1007/978-0-387-47509-7_8. [p355]
- K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas. Multi-label classification of music into emotions. *ISMIR*, 8:325–330, 2008. URL <https://doi.org/10.1186/1687-4722-2011-426793>. [p352, 362, 363]
- G. Tsoumakas and I. Katakis. Multi label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007. URL <https://doi.org/10.4018/jdwm.2007070101>. [p352]
- G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. P. Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011. [p352]
- J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. URL <https://doi.org/10.1145/2641190.2641198>. [p362]
- H. Wang, X. Liu, B. Lv, F. Yang, and Y. Hong. Reliable multi-label learning via conformal predictor and random forest for syndrome differentiation of chronic fatigue in traditional chinese medicine. *PLoS ONE*, 9(6), 2014. URL <https://doi.org/10.1371/journal.pone.0099565>. [p362]
- D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. URL [https://doi.org/10.1016/s0893-6080\(05\)80023-1](https://doi.org/10.1016/s0893-6080(05)80023-1). [p358]
- M. L. Zhang and Z. H. Zhou. M3MIML: A maximum margin method for multi-instance multi-label learning. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 688–697, 2008. URL <https://doi.org/10.1109/icdm.2008.27>. [p352, 362, 363]
- Z.-H. Zhou and M. ling Zhang. Multi-instance multilabel learning with application to scene classification. *Neural Information Processing Systems*, 40(7):2038–2048, 2007. [p362, 363]

Philipp Probst
Department of Medical Informatics, Biometry and Epidemiology
LMU Munich
81377 Munich

Germany
probst@ibe.med.uni-muenchen.de

Quay Au
Department of Statistics
LMU Munich
80539 Munich
Germany
quay.au@stat.uni-muenchen.de

Giuseppe Casalicchio
Department of Statistics
LMU Munich
80539 Munich
Germany
giuseppe.casalicchio@stat.uni-muenchen.de

Clemens Stachl
Department of Psychology
LMU Munich
80802 Munich
Germany
clemens.stachl@psy.lmu.de

Bernd Bischl
Department of Statistics
LMU Munich
80539 Munich
Germany
bernd.bischl@stat.uni-muenchen.de

Appendices

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(f)
birds*	0.4481	0.4481	0.4466	0.4497	0.4451	0.4156	0.4218	0.4171	0.4233	0.4202	0.9830	0.4777	0.5226
emotions	0.6846	0.6575	0.6728	0.6457	0.6626	0.6777	0.6643	0.7031	0.6845	0.6828	0.7992	0.6829	1.0000
genbase*	0.0333	0.0363	0.0393	0.0423	0.0363	0.0045	0.0045	0.0045	0.0060	0.0045	0.2115	0.0091	1.0000
langLog*	0.6836	0.6829	0.6884	0.6842	0.6856	0.6521	0.6349	0.6418	0.6438	0.6466	0.8589	0.6856	0.7021
enron*	0.8531	0.8413	0.8560	0.8408	0.8484	0.8496	0.819	0.8484	0.8320	0.8408	1.0000	0.8619	0.9982
reuters	0.3620	0.3405	0.3575	0.311	0.3515	0.349	0.2945	0.338	0.3495	0.3385	0.5830	0.3695	1.0000
image	0.6635	0.6150	0.6505	0.575	0.6445	0.63	0.539	0.6275	0.6225	0.619	0.8365	0.6955	1.0000
scene	0.4225	0.3926	0.4217	0.3835	0.4046	0.3913	0.3095	0.3805	0.3610	0.3648	0.7540	0.4570	1.0000
yeast*	0.8316	0.7600	0.8201	0.8167	0.8155	0.8304	0.7563	0.8134	0.8217	0.806	0.9338	0.8337	0.9855
slashdot*	0.6140	0.5994	0.6116	0.5859	0.6052	0.6489	0.5923	0.6449	0.6658	0.6396	0.9966	0.6142	0.9675

Table 5: Subset 0/1 loss

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(f)
birds*	0.6153	0.6126	0.6197	0.6169	0.6232	0.6589	0.657	0.6604	0.6581	0.6621	0.0999	0.5753	0.4774
emotions	0.5453	0.5649	0.5464	0.5849	0.5609	0.5519	0.5676	0.5408	0.5727	0.5427	0.5503	0.5332	0.0000
genbase*	0.9834	0.9806	0.9796	0.9773	0.9806	0.9972	0.9972	0.9972	0.9957	0.9972	0.8884	0.9950	0.0000
langLog*	0.3185	0.3188	0.3140	0.3188	0.3161	0.3553	0.3741	0.366	0.363	0.3615	0.1953	0.3161	0.2979
enron*	0.4693	0.4757	0.4694	0.4804	0.4742	0.483	0.4987	0.4824	0.4919	0.4847	0.1859	0.4394	0.2241
reuters	0.6625	0.6856	0.6682	0.7199	0.6754	0.6873	0.7414	0.6912	0.7197	0.6964	0.5620	0.6482	0.0000
image	0.4068	0.4585	0.4142	0.5225	0.4228	0.4446	0.5508	0.4458	0.5366	0.4564	0.4467	0.3578	0.0000
scene	0.6064	0.6333	0.6067	0.6463	0.6233	0.646	0.7201	0.6505	0.7313	0.6725	0.5513	0.5654	0.0000
yeast*	0.5091	0.5320	0.5138	0.514	0.5205	0.5182	0.5345	0.522	0.5068	0.5239	0.3674	0.4945	0.3361
slashdot*	0.4274	0.4421	0.4285	0.4569	0.4385	0.3883	0.4507	0.3925	0.4613	0.3982	0.1651	0.4202	0.0325

Table 6: Accuracy

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(f)
birds*	0.2763	0.2752	0.2897	0.2859	0.2936	0.3755	0.3865	0.3772	0.3687	0.3784	0.8352	0.1949	0.0000
emotions	0.6197	0.6474	0.6187	0.6847	0.6358	0.6335	0.6708	0.6293	0.7189	0.6237	0.8276	0.6001	0.0000
genbase*	0.9846	0.9819	0.9809	0.9786	0.9819	0.9977	0.9977	0.9977	0.9962	0.9977	0.9962	0.9955	0.0000
langLog*	0.0334	0.0330	0.0270	0.0331	0.0308	0.0971	0.1191	0.1056	0.0995	0.102	0.9264	0.0301	0.0000
enron*	0.5426	0.5487	0.5421	0.5580	0.5466	0.5611	0.5902	0.5619	0.6314	0.5633	0.771	0.4959	0.2613
reuters	0.6733	0.6959	0.6801	0.7338	0.6875	0.7038	0.754	0.7046	0.9032	0.7123	0.8598	0.6559	0.0000
image	0.4192	0.4696	0.4228	0.5562	0.4335	0.4581	0.5691	0.4603	0.7787	0.4724	0.7374	0.3598	0.0000
scene	0.6148	0.6373	0.6134	0.6555	0.6306	0.6614	0.7243	0.6613	0.8174	0.6879	0.9173	0.5662	0.0000
yeast*	0.5722	0.6097	0.5788	0.6035	0.5874	0.5951	0.6229	0.5978	0.6104	0.6013	0.6296	0.5442	0.3365
slashdot*	0.4267	0.4412	0.4270	0.4574	0.4391	0.3834	0.4526	0.3868	0.6984	0.3931	0.8065	0.4094	0.0000

Table 7: Recall

	BR(rf)	CC(rf)	NST(rf)	DBR(rf)	STA(rf)	BR(ad)	CC(ad)	NST(ad)	DBR(ad)	STA(ad)	RFERN	RFSRC	BR(f)
birds*	0.8812	0.8889	0.8764	0.9056	0.8874	0.8461	0.8349	0.8401	0.8648	0.8605	0.0859	0.8996	
emotions	0.7627	0.7242	0.7499	0.7265	0.7644	0.7537	0.7014	0.739	0.6783	0.7347	0.5869	0.7577	
genbase*	0.9987	0.9987	0.9987	0.9987	0.9987	0.9995	0.9995	0.9995	0.9995	0.9995	0.8917	0.9995	
langLog*	0.7267	0.7356	0.7058	0.7207	0.6882	0.6874	0.7228	0.7133	0.7014	0.6965	0.0632	0.7233	
enron*	0.7283	0.7188	0.7305	0.7092	0.7331	0.7235	0.6807	0.7198	0.6371	0.7233	0.1973	0.7448	0.5135
reuters	0.9411	0.9168	0.9346	0.8995	0.9298	0.9014	0.7689	0.8983	0.7465	0.8931	0.5715	0.9562	
image	0.7899	0.7333	0.8029	0.7086	0.7865	0.7841	0.6281	0.7814	0.6036	0.7737	0.4813	0.83	
scene	0.9071	0.8956	0.9112	0.8917	0.9143	0.8936	0.81	0.8856	0.7879	0.8872	0.5662	0.9233	
yeast*	0.7372	0.7218	0.7351	0.7055	0.7389	0.7225	0.6947	0.7233	0.6827	0.7159	0.4361	0.7508	0.7478
slashdot*	0.8365	0.8127	0.8298	0.7927	0.8277	0.8119	0.6804	0.8161	0.5025	0.8196	0.1679	0.8366	

Table 8: Precision (For the featureless learner we have no precision results for several datasets. The reason is that the featureless learner does not predict any value in all observations in these datasets. Hence, the denominator in the precision formula is always zero. *mlr* predicts NA in this case.)

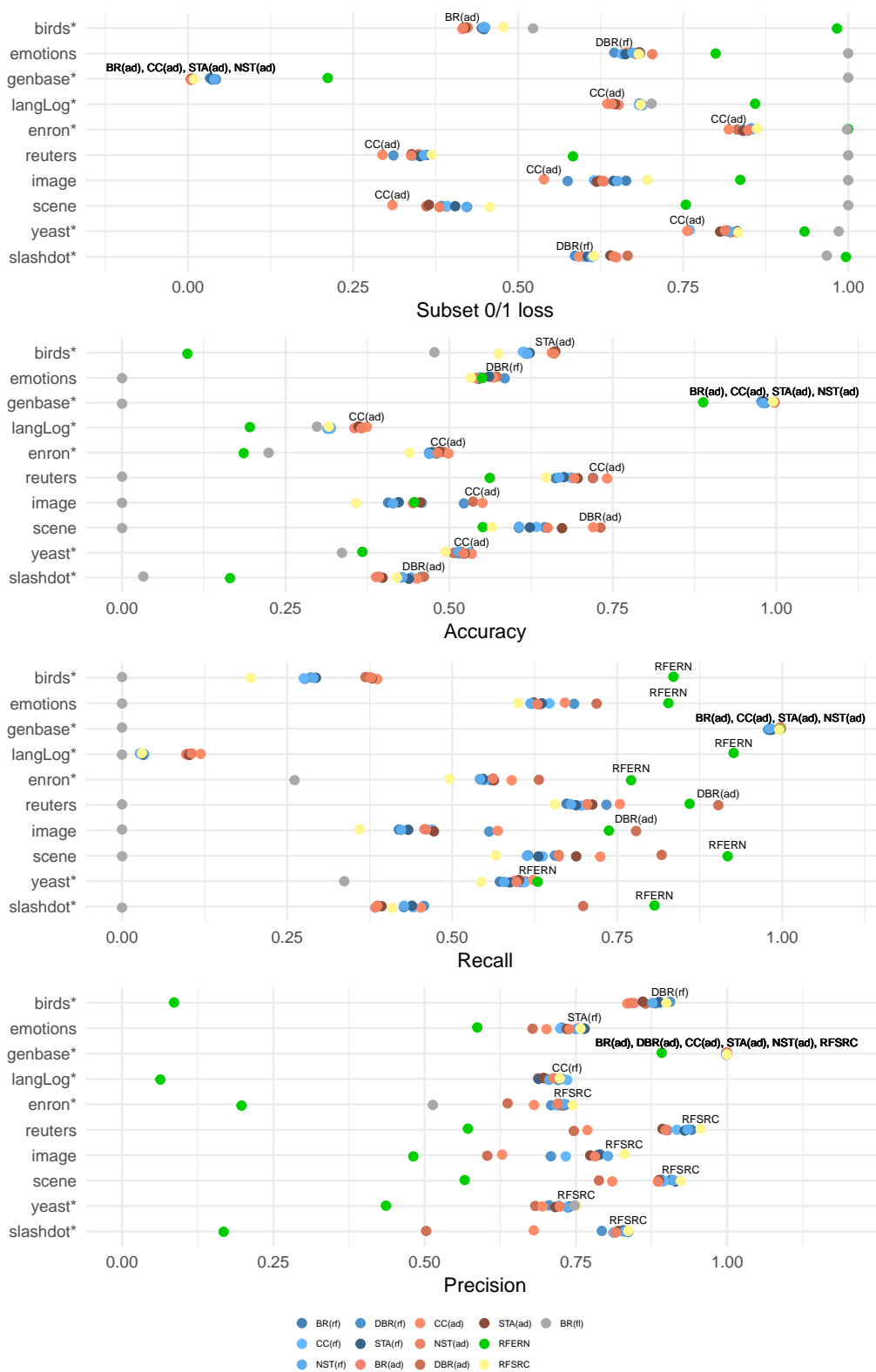


Figure 2: Results for the remaining measures.

Counterfactual: An R Package for Counterfactual Analysis

by Mingli Chen, Victor Chernozhukov, Iván Fernández-Val and Blaise Melly

Abstract The **Counterfactual** package implements the estimation and inference methods of Chernozhukov et al. (2013) for counterfactual analysis. The counterfactual distributions considered are the result of changing either the marginal distribution of covariates related to the outcome variable of interest, or the conditional distribution of the outcome given the covariates. They can be applied to estimate quantile treatment effects and wage decompositions. This paper serves as an introduction to the package and displays basic functionality of the commands contained within.

Introduction

Using econometric terminology, we can often think of a counterfactual distribution as the result of a change in either the distribution of a set of covariates X that determine the outcome variable of interest Y , or the relationship of the covariates with the outcomes, that is, a change in the conditional distribution of Y given X . Counterfactual analysis consists of evaluating the effects of such changes. The **Counterfactual** package implements the methods of (Chernozhukov et al., 2013) for counterfactual analysis. It contains commands to estimate and make inference on quantile effects constructed from counterfactual distributions. The counterfactual distributions are estimated using regression methods such as classical, duration, quantile and distribution regressions. The inference on the quantile effect function can be pointwise at a specific quantile index or uniform over a range of specified quantile indexes.

We start by giving a simple example of counterfactual analysis. Suppose we would like to analyze the wage differences between men and women. Let 0 denote the population of men and let 1 denote the population of women. The variable Y_j denotes wages and X_j denotes job market-relevant characteristics that affect wages for populations $j = 0$ and $j = 1$. The conditional distribution functions $F_{Y_0|X_0}(y|x)$ and $F_{Y_1|X_1}(y|x)$ describe the stochastic assignment of wages to workers with characteristics x , for men and women, respectively. Let $F_{Y\langle 0|0 \rangle}$ and $F_{Y\langle 1|1 \rangle}$ represent the observed distribution function of wages for men and women, and let $F_{Y\langle 0|1 \rangle}$ represent the distribution function of wages that would have prevailed for women had they faced the men's wage schedule $F_{Y_0|X_0}$:

$$F_{Y\langle 0|1 \rangle}(y) := \int_{\mathcal{X}_1} F_{Y_0|X_0}(y|x) dF_{X_1}(x).$$

The latter distribution is called counterfactual, since it does not arise as a distribution from any observable population. Rather, this distribution is constructed by integrating the conditional distribution of wages for men with respect to the distribution of characteristics for women. This quantity is well defined if \mathcal{X}_0 , the support of men's characteristics, includes \mathcal{X}_1 , the support of women's characteristics, namely $\mathcal{X}_1 \subset \mathcal{X}_0$.

Let F^{\leftarrow} denote the quantile or left-inverse function of the distribution function F . The difference in the observed wage quantile function between men and women can be decomposed in the spirit of (Oaxaca, 1973) and (Blinder, 1973) as

$$F_{Y\langle 1|1 \rangle}^{\leftarrow} - F_{Y\langle 0|0 \rangle}^{\leftarrow} = [F_{Y\langle 1|1 \rangle}^{\leftarrow} - F_{Y\langle 0|1 \rangle}^{\leftarrow}] + [F_{Y\langle 0|1 \rangle}^{\leftarrow} - F_{Y\langle 0|0 \rangle}^{\leftarrow}], \quad (1)$$

where the first term in brackets is due to differences in the wage structure and the second term is a composition effect due to differences in characteristics. These counterfactual effects are well defined econometric parameters and are widely used in empirical analysis, for example, the first term of the decomposition is a measure of gender wage discrimination. In Section P.3.2 we consider an empirical example where 0 denotes the population of nonunion workers and 1 denotes the population of union workers. In this case the wage structure effect corresponds to the treatment effect of union or union premium. It is important to note that these effects do not necessarily have a causal interpretation without additional conditions that are spelled out in (Chernozhukov et al., 2013).

The Counterfactual package

Getting started

To get started using the package **Counterfactual** for the first time, issue the command

```
> install.packages("Counterfactual")
```

into your R browser to install the package in your computer. Once the package has been installed, you can use the package **Counterfactual** during any R session by simply issuing the command

```
> library(Counterfactual)
```

Now you are ready to use the function *counterfactual* and data sets contained in **Counterfactual**. For general questions about the package you may type

```
> help(package = "Counterfactual")
```

to view the package help file, or for more questions about a specific function you can type 'help(function-name)'. For example, try:

```
> help(counterfactual)
```

or simply type

```
> ?counterfactual
```

The command *counterfactual* has the general syntax:

```
> counterfactual(formula, data, weights, na.action = na.exclude,
+               group, treatment = FALSE, decomposition = FALSE,
+               transformation = FALSE, counterfactual_var,
+               quantiles, method = "qr",
+               trimming = 0.005, nreg = 100, scale_variable,
+               counterfactual_scale_variable,
+               censoring = 0, right = FALSE, nsteps = 3,
+               firstc = 0.1, secondc = 0.05, noboot = FALSE,
+               weightedboot = FALSE, seed = 8, robust = FALSE,
+               reps = 100, alpha = 0.05, first = 0.1,
+               last = 0.9, cons_test = 0, printdeco = TRUE,
+               sepcore = FALSE, ncore=1)
```

To describe the different options of the command we need to provide some background on methods for counterfactual analysis.

Setting for counterfactual analysis

Consider a general setting with two populations labeled by $k \in \mathcal{K} = \{0, 1\}$. For each population k there is the d_x -vector X_k of covariates and the scalar outcome Y_k . The covariate vector is observable in all populations, but the outcome is only observable in populations $j \in \mathcal{J} \subseteq \mathcal{K}$. Let F_{X_k} denote the covariate distribution in population $k \in \mathcal{K}$, and $F_{Y_j|X_j}$ and $Q_{Y_j|X_j}$ denote the conditional distribution and quantile functions in population $j \in \mathcal{J}$. We denote the support of X_k by $\mathcal{X}_k \subseteq \mathbb{R}^{d_x}$, and the region of interest for Y_j by $\mathcal{Y}_j \subseteq \mathbb{R}$. We refer to j as the reference population(s) and to k as the counterfactual population(s).

The reference and counterfactual populations in the wage examples correspond to different groups such as men and women or nonunion and union workers. We can also generate counterfactual populations by artificially transforming a reference population. Formally, we can think of X_k as being created through a known transformation of X_j :

$$X_k = g_k(X_j), \quad \text{where } g_k : \mathcal{X}_j \rightarrow \mathcal{X}_k. \quad (2)$$

This case covers adding one unit to the first covariate, $X_{1,k} = X_{1,j} + 1$, holding the rest of the covariates constant. The resulting quantile effect becomes the *unconditional* quantile regression, which measures the effect of a unit change in a given covariate component on the unconditional quantiles of Y . For example, this type of counterfactual is useful for estimating the treatment effect of smoking during pregnancy on infant birth weights. Another possible transformation is a mean preserving redistribution of the first covariate implemented as $X_{1,k} = (1 - \alpha)E[X_{1,j}] + \alpha X_{1,j}$. These and more

general types of transformation defined in (2) are useful for estimating the effect of a change in taxation on the marginal distribution of food expenditure or the effect of cleaning up a local hazardous waste site on the marginal distribution of housing prices ((Stock, 1991)). We give an example of this type of transformation in Section P.3.1.

The reference and counterfactual populations can be specified to *counterfactual* in two ways that accommodate the previous two cases:

1. If the option *group* has been specified, then j is the population defined by *group* and k is the population defined by *group=1*. This means that both X and Y are observed in *group=0*, but only X needs to be observed in *group=1*. When both X and Y are observed in *group=1*, the option *treatment=TRUE* specifies that the structure or treatment effect should be computed, whereas the default option *treatment=FALSE* specifies that the composition effect should be computed; see the definition of the structure and composition effects in the decomposition (1). If in addition to *treatment=TRUE* the option *decomposition=TRUE* is selected, then the entire decomposition (1) is reported including the composition, structure and total effects. Note that we can reverse the roles of the populations defined by an indicator variable *vargroup* by setting either *group=vargroup* or *group=1-vargroup*.
2. Alternatively, the option *counterfactual_var* can be used to specify the covariates in the counterfactual population. In this case, the names on the right handside of *formula* contain the variables in X_j and *counterfactual_var* contains the variables in X_k . The option *transformation=TRUE* should be used when X_k is generated as a transformation of X_j , e.g., equation (2). The list passed to *counterfactual_var* must contain exactly the same number of variables as the list of independent variables in *formula* and the order of the variables in the list matters.

Counterfactual distribution and quantile functions are formed by combining the conditional distribution in the population j with the covariate distribution in the population k , namely:

$$F_{Y(j|k)}(y) := \int_{\mathcal{X}_k} F_{Y_j|X_j}(y|x) dF_{X_k}(x), \quad y \in \mathcal{Y}_j,$$

$$Q_{Y(j|k)}(\tau) := F_{Y(j|k)}^{\leftarrow}(\tau), \quad \tau \in (0, 1),$$

where $(j, k) \in \mathcal{JK}$, and $F_{Y(j|k)}^{\leftarrow}(\tau) = \inf\{y \in \mathcal{Y}_j : F_{Y(j|k)}(y) \geq \tau\}$ is the left-inverse function of $F_{Y(j|k)}$. The main interest lies in the quantile effect (QE) function, defined as the difference of two counterfactual quantile functions over a set of quantile indexes $\mathcal{T} \subset (0, 1)$:

$$\Delta(\tau) = Q_{Y(j|k)}(\tau) - Q_{Y(j|j)}(\tau), \quad \tau \in \mathcal{T},$$

where $j \in \mathcal{J}$ and $k \in \mathcal{K}$. In the example of Section P.1, we obtain the composition effect with $j = 0$ and $k = 1$. When Y_k is observed, then we can construct the structure effect or treatment effect on the treated

$$\Delta(\tau) = Q_{Y(k|k)}(\tau) - Q_{Y(j|k)}(\tau), \quad \tau \in \mathcal{T},$$

by specifying the option *group* and setting *treatment=TRUE*. In the example of Section P.1, we obtain the wage structure effect with $j = 0$ and $k = 1$, i.e. setting *group=1* and *treatment=TRUE*. If in addition we select the option *decomposition=TRUE*, then we obtain the entire decomposition (1) including the composition, structure and total effects. The total effect is

$$\Delta(\tau) = Q_{Y(k|k)}(\tau) - Q_{Y(j|j)}(\tau), \quad \tau \in \mathcal{T}.$$

The set \mathcal{T} is specified with the option *quantiles*, which enumerates the quantile indexes of interested and should be a vector containing numbers between 0 and 1.

To estimate the QE function we need to model and estimate the conditional distribution $F_{Y_j|X_j}$ and covariate distribution F_{X_k} . We estimate the covariate distribution using the empirical distribution, and consider several regression based methods for the conditional distribution including classical, quantile, duration, and distribution regression. Given the estimators of the conditional and covariate distributions $\hat{F}_{Y_j|X_j}$ and \hat{F}_{X_k} , the estimator of each counterfactual distribution is obtained by the plug-in rule, namely

$$\hat{F}_{Y(j|k)}(y) = \int_{\mathcal{X}_k} \hat{F}_{Y_j|X_j}(y|x) d\hat{F}_{X_k}(x), \quad y \in \mathcal{Y}_j.$$

Then, the estimator of the QE function is also obtained by the plug-in rule as

$$\hat{\Delta}(\tau) = \hat{F}_{Y(j|k)}^{\leftarrow}(\tau) - \hat{F}_{Y(j|j)}^{\leftarrow}(\tau), \quad \tau \in \mathcal{T},$$

or

$$\hat{\Delta}(\tau) = \hat{F}_{Y(k|k)}^{\leftarrow}(\tau) - \hat{F}_{Y(j|k)}^{\leftarrow}(\tau), \quad \tau \in \mathcal{T},$$

if we define the counterfactual population with *group* and set *treatment=TRUE*. If in addition to *treatment=TRUE*, we select *decomposition=TRUE*, then the plug-in estimator of the total effect is

$$\hat{\Delta}(\tau) = \hat{F}_{Y_{(k|k)}}^{\leftarrow}(\tau) - \hat{F}_{Y_{(j|j)}}^{\leftarrow}(\tau), \quad \tau \in \mathcal{T}.$$

Estimation of conditional distribution

In this section we assume that we have samples $\{(Y_{ji}, X_{ji}) : i = 1, \dots, n_j\}$ composed of independent and identically distributed copies of (Y_j, X_j) for all populations $j \in \mathcal{J}$. The conditional distribution $F_{Y_j|X_j}$ can be modeled and estimated directly, or through the conditional quantile function, $Q_{Y_j|X_j}$, using the relation

$$F_{Y_j|X_j}(y|x) \equiv \int_{(0,1)} 1\{Q_{Y_j|X_j}(u|x) \leq y\} du. \tag{3}$$

The option *formula* specifies the outcome Y as the left hand side variable and the covariates X as the right hand side variable(s). The option *method* allows to select the method to estimate the conditional distribution. The following methods are implemented:

1. *method = "qr"*, which is the default, implements the quantile regression estimator of the conditional distribution

$$\hat{F}_{Y_j|X_j}(y|x) = \varepsilon + \int_{(\varepsilon, 1-\varepsilon)} 1\{x' \hat{\beta}_j(u) \leq y\} du, \tag{4}$$

where ε is a small constant that avoids estimation of tail quantiles, and $\hat{\beta}(u)$ is the (Koenker and Bassett, 1978) quantile regression estimator

$$\hat{\beta}_j(u) = \arg \min_{b \in \mathbb{R}^{d_x}} \sum_{i=1}^{n_j} [u - 1\{Y_{ji} \leq X'_{ji}b\}] [Y_{ji} - X'_{ji}b].$$

The quantile regression estimator calls the R package **quantreg** (Koenker, 2016). The option *trimming* specifies the value of the trimming parameter ε , with default value $\varepsilon = 0.005$. The option *nreg* sets the number of quantile regressions used to approximate the integral in (4), with a default value of 100 such that $(\varepsilon, 1 - \varepsilon)$ is approximated by the grid $\{\varepsilon, \varepsilon + (1 - 2\varepsilon)/99, \varepsilon + 2(1 - 2\varepsilon)/99, \dots, 1 - \varepsilon\}$. This method should be used only with continuous dependent variables.

2. *method = "loc"* implements the estimator of the conditional distribution

$$\hat{F}_{Y_j|X_j}(y|x) = \frac{1}{n_j} \sum_{i=1}^{n_j} 1\{Y_{ji} - X'_{ji} \hat{\beta}_j \leq y - x' \hat{\beta}_j\}, \tag{5}$$

where $\hat{\beta}_j$ is the least square estimator

$$\hat{\beta}_j = \arg \min_{b \in \mathbb{R}^{d_x}} \sum_{i=1}^{n_j} (Y_{ji} - X'_{ji}b)^2. \tag{6}$$

The estimator (5) is based on a restrictive location shift model that imposes that the covariates X only affect the location of the outcome Y .

3. *method = "locsca"* implements the estimator of the conditional distribution

$$\hat{F}_{Y_j|X_j}(y|x) = \frac{1}{n_j} \sum_{i=1}^{n_j} 1 \left\{ \frac{Y_{ji} - X'_{ji} \hat{\beta}_j}{\exp(X'_{2ji} \hat{\gamma}_j / 2)} \leq \frac{y - x' \hat{\beta}_j}{\exp(x'_{2j} \hat{\gamma}_j / 2)} \right\}, \tag{7}$$

where $\hat{\beta}_j$ is the least square estimator (6), $X_{2j} \subseteq X_j$ with $\dim X_{2j} = d_{x_2}$, and

$$\hat{\gamma}_j = \arg \min_{g \in \mathbb{R}^{d_{x_2}}} \sum_{i=1}^{n_j} (\log(Y_{ji} - X'_{ji} \hat{\beta}_j)^2 - X'_{2ji}g)^2.$$

The option *scale_variable* specifies the covariates X_{2j} that affect the scale of the conditional distribution. The option *counterfactual_scale_variable* selects the counterfactual scale variables when the counterfactual population is specified using *counterfactual_var*. By default, R would use all the covariates as *scale_variable* and *counterfactual_scale_variable = counterfactual_var*. The estimator (7) is based on a restrictive location scale shift model that imposes that the covariates X only affect the location and scale of the outcome Y .

4. *method = "cqr"* implements the censored quantile regression estimator of the conditional distribution, which is the same as (4) with $\hat{\beta}(u)$ replaced by the (Chernozhukov and Hong, 2002)

censored quantile regression estimator. The options *trimming* and *nreg* apply to this method with the same functionality as for the *qr* method. Moreover, a variable containing a censoring indicator C_j must be specified with *censoring*. The censored quantile regression estimator has three-steps by default. The number of steps can be increased by the option *nsteps*. In the first step, the censoring probabilities are estimated by a logit regression of the censoring indicator C_j on all the covariates X_j . Then, for each quantile index u , the observations with sufficiently low censoring probabilities relative to u are selected. We allow for misspecification of the logit by excluding the observations that could theoretically be used but have censoring probabilities in the highest *firstc* quantiles, with a default of 0.1, i.e. 10% of the observations. In the second step, standard linear quantile regressions are estimated on the samples defined in step one. Using the estimated quantile regressions, we define a new sample of observations that can be used. This sample consists of all observations for which the estimated conditional quantile is above the censoring point. Again, we throw away observations in the lowest *secondc* quantiles of the distribution of the residuals, with a default of 0.05, i.e. 5% of the observations. Step three consists in a new linear quantile regression using the sample defined in step two. Step three is repeated if *nsteps* is above 3. This method should be used only with censored dependent variables.

5. *method = "cox"* implements the duration regression estimator of the conditional distribution function

$$\hat{F}_{Y_j|X_j}(y|x) = 1 - \exp(-\exp(\hat{t}(y) - x'\hat{\beta})), \tag{8}$$

where $\hat{\beta}$ is the Cox estimator of the regression coefficients and $\hat{t}(y)$ is the Cox estimator of the baseline integrated hazard function (Cox, 1972). The Cox estimator calls the R package *survival* (Therneau, 2015). The estimator (8) is based on a restrictive transformation location shift model that imposes that the covariates X only affect the location of a monotone transformation of the outcome $t(Y)$, i.e.

$$t(Y_j) = X_j'\beta_j + V_j,$$

where V_j has an extreme value distribution and is independent of X_j . This method should be used only with nonnegative dependent variables.

6. *method = "logit"* implements the distribution regression estimator of the conditional distribution with logistic link function

$$\hat{F}_{Y_j|X_j}(y|x) = \Lambda(x'\hat{\beta}(y)), \tag{9}$$

where Λ is the standard logistic distribution function, and $\hat{\beta}(y)$ is the distribution regression estimator

$$\hat{\beta}(y) = \arg \max_{b \in \mathbb{R}^{d_x}} \sum_{i=1}^{n_j} [1\{Y_{ji} \leq y\} \log \Lambda(X'_{ij}b) + 1\{Y_{ij} > y\} \log \Lambda(-X'_{ij}b)]. \tag{10}$$

The estimator (9) is based on a flexible model where each covariate can have a heterogenous effect at different parts of the distribution. This method can be used with continuous dependent variables and censored dependent variables with fixed censoring point.

7. *method = "probit"* implements the distribution regression estimator of the conditional distribution with normal link function, i.e. where Λ is the standard normal distribution function in (9) and (10).
8. *method = "lpm"* implements the linear probability model estimator of the conditional distribution

$$\hat{F}_{Y_j|X_j}(y|x) = x'\hat{\beta}(y),$$

where $\hat{\beta}(y)$ is the least squares estimator

$$\hat{\beta}(y) = \arg \min_{b \in \mathbb{R}^{d_x}} \sum_{i=1}^{n_j} (1\{Y_{ji} \leq y\} - X'_{ij}b)^2.$$

This method might produce estimates of the conditional distribution outside the interval $[0, 1]$.

For the methods (2)–(8), the option *nreg* sets the number of values of y to evaluate the estimator of the conditional distribution function. These values are uniformly distributed among the observed values of Y_j . If *nreg* is greater than the number of observed values of Y_j , then all the observed values are used.

Inference

The command *counterfactual* reports pointwise and uniform confidence intervals for the QEs over a prespecified set of quantile indexes. The construction of the intervals rely on functional central limit theorems and bootstrap functional central limit theorems for the empirical QEs derived in (Chernozhukov et al., 2013). In particular, the pointwise intervals are based on the normal distribution, whereas the uniform intervals are based on two resampling schemes: empirical and weighted bootstrap. Thus, the $(1 - \alpha)$ confidence interval for $\Delta(\tau)$ on \mathcal{T} has the form

$$\{\hat{\Delta}(\tau) \pm c_{1-\alpha} \hat{\Sigma}(\tau) : \tau \in \mathcal{T}\},$$

where $\hat{\Sigma}(\tau)$ is the standard error of $\hat{\Delta}(\tau)$ and $c_{1-\alpha}$ is a critical value. There are two options to obtain $\hat{\Sigma}(\tau)$. The default option *robust=FALSE* computes the bootstrap standard deviation of $\hat{\Delta}(\tau)$; whereas the option *robust=TRUE* computes the bootstrap interquartile range rescaled with the normal distribution, $\hat{\Sigma}(\tau) = (q_{0.75}(\tau) - q_{0.25}(\tau)) / (z_{0.75} - z_{0.25})$ where $q_p(\tau)$ is the p th quantile of the bootstrap draws of $\hat{\Delta}(\tau)$ and z_p is the p th quantile of the standard normal. The pointwise critical value is $c_{1-\alpha} = z_{1-\alpha}$, and the uniform critical value is $c_{1-\alpha} = \hat{t}_{1-\alpha}$, where $\hat{t}_{1-\alpha}$ is a bootstrap estimator of the $(1 - \alpha)$ th quantile of the Kolmogorov-Smirnov maximal t-statistic

$$t = \sup_{\tau \in \mathcal{T}} |\hat{\Delta}(\tau) - \Delta(\tau)| / \hat{\Sigma}(\tau).$$

In addition to the intervals, *counterfactual* reports the p-values for several functional tests based on two test-statistic: Kolmogorov-Smirnov and the Cramer-von-Misses-Smirnov. The null-hypotheses considered are

1. Correct parametric specification of the model for the conditional distribution. This test compares the empirical distribution of the outcome Y_j with the estimate of the counterfactual distribution in the reference population

$$\hat{F}_{Y(j|j)}(y) := \int_{\mathcal{X}_j} \hat{F}_{Y_j|X_j}(y|x) d\hat{F}_{X_j}(x).$$

The power of this specification test might be low because it only uses the implications of the conditional distribution on the counterfactual distribution. For example, the test is not informative for the linear probability and logit models where the counterfactual distribution in the reference population is identical to the empirical distribution by construction. If *group* is specified and *treatment=TRUE* is selected, then the test is performed in the population defined by *group=1*. If in addition the option *decomposition=TRUE* is selected, then the test is performed in the populations defined by *group=0* and *group=1*, and in the combined population including both *group=0* and *group=1*.

2. Zero QE at all the quantile indexes of interest: $\Delta(\tau) = 0$ for all $\tau \in \mathcal{T}$. This is stronger than a zero average effect. Other null hypotheses of constant quantile effect (but at a different level than 0) can be added with the option *cons_test*.
3. Constant QE at all quantile indexes of interest: $\Delta(\tau) = \Delta(0.5)$ for all $\tau \in \mathcal{T}$.
4. First-order stochastic dominance: $\Delta(\tau) \geq 0$ for all $\tau \in \mathcal{T}$.
5. Negative first-order stochastic dominance: $\Delta(\tau) \leq 0$ for all $\tau \in \mathcal{T}$.

The options of *counterfactual* related to inference are:

1. *noboot = TRUE* suppresses the bootstrap. The bootstrap can be very demanding in terms of computation time. Therefore, it is recommended to switch it off for the exploratory analysis of the data.
2. *weightedboot = TRUE* selects weighted bootstrap with standard exponential weights. The default *weightedboot = FALSE* selects empirical bootstrap with multinomial weights. We recommend weighted bootstrap when the covariates include categorical variables with small cell sizes to avoid singular designs in the bootstrap draws.
3. *reps* specifies the number of bootstrap replications. This number will matter only if the bootstrap has not be suppressed. The default is 100.
4. *alpha* specifies the significance level of the tests and confidence intervals. Note that the confidence level of the confidence interval is $1 - \alpha$. Thus, the default value of 0.05 produces 95% confidence intervals.

5. *first* and *last* select the subset of quantile indexes of interest for inference. The tails of the distribution should not be used because standard asymptotic does not apply to these parts. The needed amount of tail trimming depends on the sample size and on the distribution of the dependent variable. *first* sets the lowest quantile index used and *last* sets the highest quantile index used. The default values are 0.1 and 0.9 so that $\mathcal{T} = [0.1, 0.9]$.
6. *const_test* add tests of the null hypothesis that $\Delta(\tau) = \text{const_test}$ for all τ between *first* and *last*. The null hypothesis that $\Delta(\tau) = 0$ for all τ between *first* and *last* is tested by default. The null hypothesis that the quantile effects are constant is also tested by default.

Parallel computing

The command *counterfactual* provides functionality for parallel computing, which is specially useful to speed up the execution of the bootstrap. There are two options related to parallel computing:

1. *setcore* specifies whether multiple cores should be used. The default value *setcore* = *FALSE* turns off the parallel computing.
2. *ncore* selects the number of cores to use for parallel computing. The information of this option is only used when parallel computing is switched on with *setcore* = *TRUE*.

Empirical examples

We consider two empirical examples to illustrate the functionality of the command *counterfactual*. The first example is an estimation of Engel curves that includes a counterfactual analysis where the counterfactual population is an artificial transformation of a reference population. The second example is wage decomposition with respect to union status where the reference and counterfactual populations correspond to two different groups.

Engel curves

We use the classical Engel 1857 dataset to estimate the relationship between food expenditure (*foodexp*) and annual household income (*income*), and then report the estimates of the QE of a change in the distribution of the annual household income that might be induced for example by a variation in income taxation.¹ We estimate the conditional distribution with the quantile regression method, i.e., *method* = "qr".

First, we generate the variable *counterfactual_income* with the counterfactual values of income and plot the reference and counterfactual income distributions. The counterfactual distribution corresponds to a mean preserving spread of the distribution in the reference population that reduces standard deviation by 25%.

```
> library(quantreg)
> data(engel)
> attach(engel)
> counter_income <- mean(income)+0.75*(income-mean(income))
> cdfx <- c(1:length(income))/length(income)
> plot(c(0,4000),range(c(0,1)), xlim =c(0, 4000), type="n", xlab = "Income",
+      ylab="Probability")
> lines(sort(income), cdfx)
> lines(sort(counter_income), cdfx, lwd = 2, col = 'grey70')
> legend(1500, .2, c("Original", "Counterfactual"), lwd=c(1,2),bty="n",
+       col=c(1, 'grey70'))
```

To estimate the QEs of this counterfactual change we turn on the option *transformation* of *counterfactual* by setting *transformation* = *TRUE*, and specify that the counterfactual values of the covariate *income* are in the generated variable *counter_income* by setting *counterfactual_var* = *counter_income*. This yields:

```
> qrres <- counterfactual(foodexp~income, counterfactual_var
+                       = counter_income, transformation = TRUE)
```

¹This is the same data set as in the quantile regression package **quantreg**, see (Koenker, 2016).

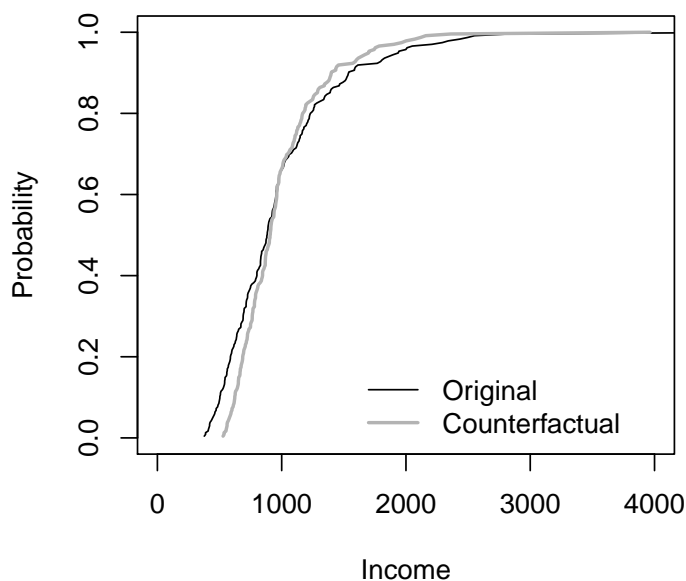


Figure 1: Observed and counterfactual distributions of income

Conditional Model: linear quantile regression
 Number of regressions estimated: 100

The variance has been estimated by bootstrapping the results 100 times.

No. of obs. in the reference group: 235
 No. of obs. in the counterfactual group: 235

Quantile Effects -- Composition

Quantile	Est.	Pointwise		Functional	
		Std.Err	95% Conf.Interval	95% Conf.Interval	
0.1	68.049	4.214	59.789 76.308	55.939	80.159
0.2	57.897	4.332	49.406 66.388	45.448	70.346
0.3	43.851	5.246	33.568 54.133	28.774	58.927
0.4	29.248	5.091	19.270 39.227	14.618	43.878
0.5	16.716	4.602	7.696 25.735	3.491	29.940
0.6	5.744	4.308	-2.698 14.187	-6.634	18.123
0.7	-8.866	7.132	-22.845 5.113	-29.361	11.630
0.8	-40.099	8.191	-56.153 -24.045	-63.637	-16.561
0.9	-88.56	13.83	-115.67 -61.44	-128.31	-48.80

Bootstrap inference on the counterfactual quantile process

NULL-Hypothesis	P-values	
	KS-statistic	CMS-statistic
Correct specification of the parametric model	0	0
No effect: $QE(\tau)=0$ for all τ s	0	0
Constant effect: $QE(\tau)=QE(0.5)$ for all τ s	0	0
Stochastic dominance: $QE(\tau)>0$ for all τ s	0	0
Stochastic dominance: $QE(\tau)<0$ for all τ s	0	0

We reject the simultaneous hypotheses of zero, constant, positive and negative effect of the income redistribution at all the deciles. The QR model for the conditional distribution cannot be rejected at conventional significance levels.

Finally, we reestimate the QE function on the larger set of quantiles $\{0.01, 0.02, \dots, 0.99\}$, and plot a uniform confidence band over the subset $\{0.10, 0.11, \dots, 0.90\}$ constructed by empirical bootstrap with 100 replications. In Figure 2 we can visually reject the functional hypotheses of zero, constant, positive and negative effect at the percentiles considered. We use the option `printdeco = FALSE` to suppress the display of the table of results.

```
> taus <- c(1:99)/100
> first <- sum(as.double(taus <= .10))
> last <- sum(as.double(taus <= .90))
> rang <- c(first:last)
> rqres <- counterfactual(foodexp~income, counterfactual_var=counter_income,
+                         nreg=100, quantiles=taus, transformation = TRUE,
+                         printdeco = FALSE, sepcore = TRUE, ncore=2)

cores used= 2

> duqf <- (rqres$resCE)[,1]
> l.duqf <- (rqres$resCE)[,3]
> u.duqf <- (rqres$resCE)[,4]

> plot(c(0,1), range(c(min(l.duqf[rang]), max(u.duqf[rang]))), xlim = c(0,1),
+      type = "n", xlab = expression(tau), ylab = "Difference in Food Expenditure",
+      cex.lab=0.75)
> polygon(c(taus[rang], rev(taus[rang])), c(u.duqf[rang], rev(l.duqf[rang])),
+        density = -100, border = F, col = "grey70", lty = 1, lwd = 1)
> lines(taus[rang], duqf[rang])
> abline(h = 0, lty = 2)
> legend(0, -90, "QE", cex = 0.75, lwd = 4, bty = "n", col = "grey70")
> legend(0, -90, "QE", cex = 0.75, lty = 1, bty = "n", lwd = 1)
```

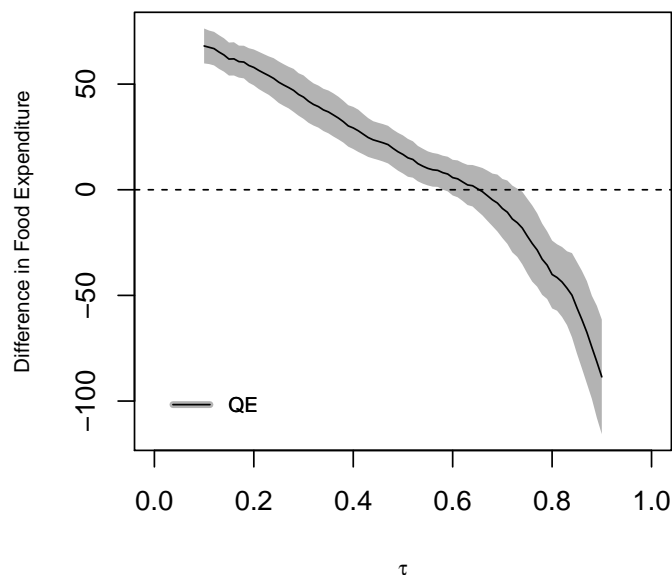


Figure 2: Quantile effects of income redistribution on food consumption

Union premium

We use an extract of the U.S. National Longitudinal Survey of Young Women (NLSW) for employed women in 1988 to estimate a wage decomposition with respect to union status.² The outcome variable Y is the log hourly wage ($lwage$), the covariates X include job tenure in years ($tenure$), years of schooling ($grade$), and total experience (tth_exp), and the union indicator $union$ defines the reference and counterfactual populations. We estimate the conditional distributions by distribution regression with logistic link and duration regression, i.e., $method = "logit"$ and $method = "cox"$. We use weighted bootstrap for the construction of uniform confidence intervals and hypothesis tests and run parallel computing with 2 nodes.

We start by estimating the wage decomposition by logistic distribution regression, where the counterfactual population is specified with $group=union$ with the options $treatment=TRUE$ and $decomposition=TRUE$ to estimate the composition, structure and total effects. The structure effect in this case correspond to the treatment effect of union on the treated or union premium. The tables show that the union workers earn higher wages than the nonunion workers throughout the distribution although the union wage gap is decreasing in the quantile index. This gap can be mostly explained by differences in tenure, education and experience between union and nonunion workers in the upper tail of the distribution and by the union premium in the rest of the distribution.

```
> data(nlsw88)
> attach(nlsw88)
> lwage <- log(wage)
> logitres <- counterfactual(lwage~tenure+tth_exp+grade,
+                           group = union, treatment=TRUE,
+                           decomposition=TRUE, method = "logit",
+                           weightedboot = TRUE, sepcore = TRUE, ncore=2)
cores used= 2
```

```
Conditional Model:          logit
Number of regressions estimated: 96
```

The variance has been estimated by bootstrapping the results 100 times.

```
No. of obs. in the reference group: 1407
No. of obs. in the counterfactual group: 459
```

Quantile Effects -- Structure

Quantile	Est.	Pointwise	Pointwise		Functional	
		Std.Err	95% Conf.Interval	95% Conf.Interval	95% Conf.Interval	95% Conf.Interval
0.1	0.24047	0.06005	0.12278	0.35817	0.07757	0.40338
0.2	0.21903	0.05630	0.10869	0.32937	0.06631	0.37176
0.3	0.23437	0.04628	0.14366	0.32508	0.10881	0.35992
0.4	0.18524	0.04252	0.10190	0.26857	0.06989	0.30059
0.5	0.16041	0.04404	0.07410	0.24671	0.04095	0.27987
0.6	0.13897	0.04618	0.04845	0.22949	0.01369	0.26425
0.7	0.05701	0.04407	-0.02937	0.14339	-0.06255	0.17657
0.8	0.01945	0.04179	-0.06245	0.10135	-0.09391	0.13281
0.9	0.006434	0.078547	-0.147514	0.160382	-0.206650	0.219518

Bootstrap inference on the counterfactual quantile process

NULL-Hypothesis	P-values	
	KS-statistic	CMS-statistic
Correct specification of the parametric model	1	1
No effect: $QE(\tau)=0$ for all τ s	0	0
Constant effect: $QE(\tau)=QE(0.5)$ for all τ s	0	0.02
Stochastic dominance: $QE(\tau)>0$ for all τ s	0.95	0.95

²This dataset is available from the Stata's sample datasets at <http://www.stata-press.com/data/r9/nlsw88.dta>.


```

> first <- sum(as.double(taus <= .10))
> last  <- sum(as.double(taus <= .90))
> rang  <- c(first:last)
> logitres <- counterfactual(lwage~tenure+ttl_exp+grade,
+   group = union, treatment=TRUE, quantiles=taus,
+   method="logit", nreg=100, weightedboot = TRUE,
+   printdeco=FALSE, decomposition = TRUE,
+   sepcore = TRUE, ncore=2)
cores used= 2
> duqf_SE <- (logitres$resSE)[,1]
> l.duqf_SE <- (logitres$resSE)[,3]
> u.duqf_SE <- (logitres$resSE)[,4]
> duqf_CE <- (logitres$resCE)[,1]
> l.duqf_CE <- (logitres$resCE)[,3]
> u.duqf_CE <- (logitres$resCE)[,4]
> duqf_TE <- (logitres$resTE)[,1]
> l.duqf_TE <- (logitres$resTE)[,3]
> u.duqf_TE <- (logitres$resTE)[,4]
> range_x <- min(c(min(l.duqf_SE[rang]), min(l.duqf_CE[rang]),
+   min(l.duqf_TE[rang])))
> range_y <- max(c(max(u.duqf_SE[rang]), max(u.duqf_CE[rang]),
+   max(u.duqf_TE[rang])))

> par(mfrow=c(1,3))
> plot(c(0,1), range(c(range_x, range_y)), xlim = c(0,1), type = "n",
+   xlab = expression(tau), ylab = "Difference in Wages", cex.lab=0.75,
+   main = "Total")
> polygon(c(taus[rang], rev(taus[rang])),
+   c(u.duqf_TE[rang], rev(l.duqf_TE[rang])), density = -100, border = F,
+   col = "grey70", lty = 1, lwd = 1)
> lines(taus[rang], duqf_TE[rang])
> abline(h = 0, lty = 2)
> plot(c(0,1), range(c(range_x, range_y)), xlim = c(0,1), type = "n",
+   xlab = expression(tau), ylab = "", cex.lab=0.75, main = "Structure")
> polygon(c(taus[rang], rev(taus[rang])),
+   c(u.duqf_SE[rang], rev(l.duqf_SE[rang])), density = -100, border = F,
+   col = "grey70", lty = 1, lwd = 1)
> lines(taus[rang], duqf_SE[rang])
> abline(h = 0, lty = 2)
> plot(c(0,1), range(c(range_x, range_y)), xlim = c(0,1), type = "n",
+   xlab = expression(tau), ylab = "", cex.lab=0.75, main = "Composition")
> polygon(c(taus[rang], rev(taus[rang])),
+   c(u.duqf_CE[rang], rev(l.duqf_CE[rang])), density = -100, border = F,
+   col = "grey70", lty = 1, lwd = 1)
> lines(taus[rang], duqf_CE[rang])
> abline(h = 0, lty = 2)

```

Finally, we repeat the point and interval estimation using the duration regression method with the option *method = "cox"*. Despite of relying on a more restrictive model for the conditional distribution, the duration regression estimates in Figure 4 are similar to the logit regression estimates in Figure 3.

```

> coxres <- counterfactual(lwage~tenure+ttl_exp+grade,
+   group = union, treatment=TRUE, quantiles=taus,
+   method="cox", nreg=100, weightedboot = TRUE,
+   printdeco = FALSE, decomposition = TRUE, sepcore = TRUE, ncore=2)
cores used= 2
> duqf_SE <- (coxres$resSE)[,1]
> l.duqf_SE <- (coxres$resSE)[,3]
> u.duqf_SE <- (coxres$resSE)[,4]
> duqf_CE <- (coxres$resCE)[,1]
> l.duqf_CE <- (coxres$resCE)[,3]
> u.duqf_CE <- (coxres$resCE)[,4]
> duqf_TE <- (coxres$resTE)[,1]
> l.duqf_TE <- (coxres$resTE)[,3]

```

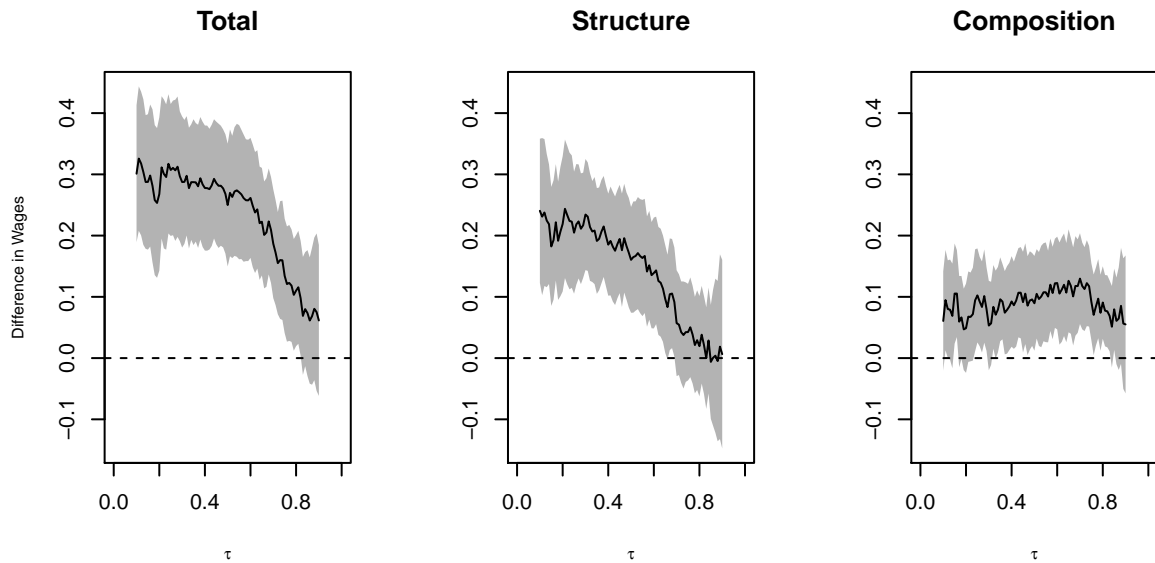


Figure 3: Wage decomposition with respect to union: logit regression estimates

```

> u.duqf_TE <- (coxres$resTE)[,4]
> range_x = min(c(min(l.duqf_SE[rang]), min(l.duqf_CE[rang]),
+               min(l.duqf_TE[rang])))
> range_y = max(c(max(u.duqf_SE[rang]), max(u.duqf_CE[rang]),
+               max(u.duqf_TE[rang])))

> par(mfrow=c(1,3))
> plot(c(0,1), range(c(range_x, range_y)), xlim = c(0,1), type = "n",
+      xlab = expression(tau), ylab = "Difference in Wages", cex.lab=0.75,
+      main = "Total")
> polygon(c(taus[rang],rev(taus[rang])),
+        c(u.duqf_TE[rang], rev(l.duqf_TE[rang])), density = -100, border = F,
+        col = "grey70", lty = 1, lwd = 1)
> lines(taus[rang], duqf_TE[rang])
> abline(h = 0, lty = 2)
> plot(c(0,1), range(c(range_x, range_y)), xlim = c(0,1), type = "n",
+      xlab = expression(tau), ylab = "", cex.lab=0.75, main = "Structure")
> polygon(c(taus[rang],rev(taus[rang])),
+        c(u.duqf_SE[rang], rev(l.duqf_SE[rang])), density = -100, border = F,
+        col = "grey70", lty = 1, lwd = 1)
> lines(taus[rang], duqf_SE[rang])
> abline(h = 0, lty = 2)
> plot(c(0,1), range(c(range_x, range_y)), xlim = c(0,1), type = "n",
+      xlab = expression(tau), ylab = "", cex.lab=0.75, main = "Composition")
> polygon(c(taus[rang],rev(taus[rang])),
+        c(u.duqf_CE[rang], rev(l.duqf_CE[rang])), density = -100, border = F,
+        col = "grey70", lty = 1, lwd = 1)
> lines(taus[rang], duqf_CE[rang])
> abline(h = 0, lty = 2)

```

Bibliography

- A. Blinder. Wage discrimination: Reduced form and structural estimates. *Journal of Human resources*, pages 436–455, 1973. [p370]
- V. Chernozhukov and H. Hong. Three-step censored quantile regression and extramarital affairs. *Journal of the American Statistical Association*, 97(459):872–882, 2002. [p373]

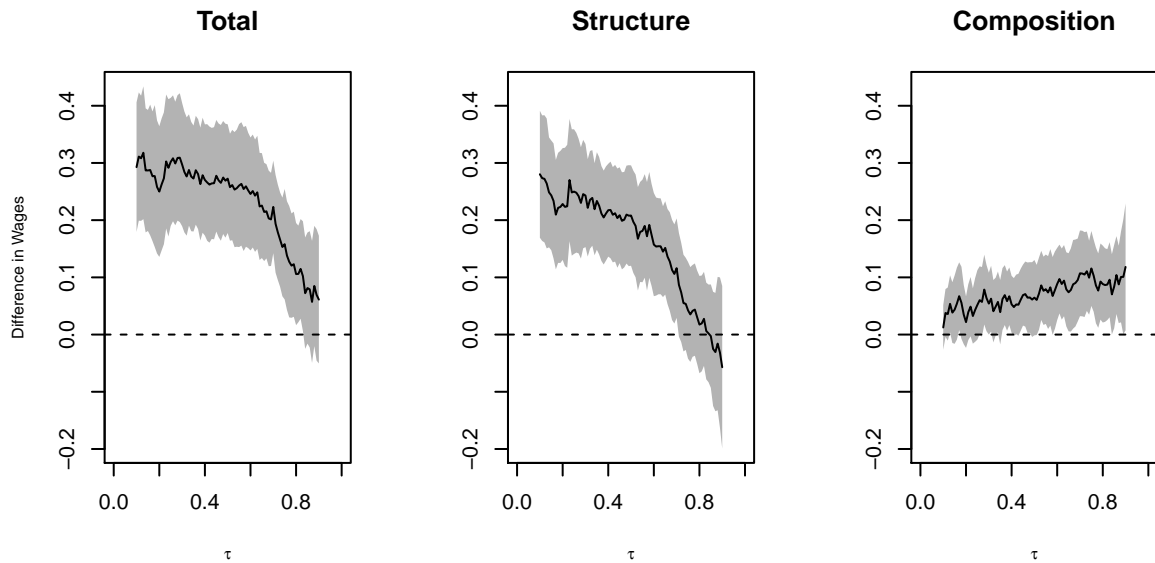


Figure 4: Wage decomposition with respect to union: duration regression estimates

- V. Chernozhukov, I. Fernández-Val, and B. Melly. Inference on counterfactual distributions. *Econometrica*, 81(6):2205–2268, 2013. [p370, 375]
- D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society B*, 34(2):187–220, 1972. [p374]
- R. Koenker. *Quantreg: Quantile Regression*, 2016. URL <https://CRAN.R-project.org/package=quantreg>. R package version 5.21. [p373, 376]
- R. Koenker and G. Bassett. Regression quantiles. *Econometrica: journal of the Econometric Society*, pages 33–50, 1978. [p373]
- R. Oaxaca. Male-female wage differentials in urban labor markets. *International economic review*, pages 693–709, 1973. [p370]
- J. Stock. Nonparametric policy analysis: An application to estimating hazardous waste cleanup benefits. *Nonparametric and Semiparametric Methods in Econometrics and Statistics*, pages 77–98, 1991. [p372]
- T. M. Therneau. *A Package for Survival Analysis in S*, 2015. URL <http://CRAN.R-project.org/package=survival>. version 2.38. [p374]

Mingli Chen
 Department of Economics
 University of Warwick
 UK
m.chen.3@warwick.ac.uk

Victor Chernozhukov
 Department of Economics
 Massachusetts Institute of Technology
 USA
vchern@mit.edu

Iván Fernández-Val
 Department of Economics
 Boston University
 USA
ivanf@bu.edu

Blaise Melly
Department of Economics
University of Bern
Germany
mellyblaise@gmail.com

Retrieval and Analysis of Eurostat Open Data with the `eurostat` Package

by Leo Lahti, Janne Huovari, Markus Kainu, and Przemysław Biecek

Abstract The increasing availability of open statistical data resources is providing novel opportunities for research and citizen science. Efficient algorithmic tools are needed to realize the full potential of the new information resources. We introduce the `eurostat` R package that provides a collection of custom tools for the Eurostat open data service, including functions to query, download, manipulate, and visualize these data sets in a smooth, automated and reproducible manner. The online documentation provides detailed examples on the analysis of these spatio-temporal data collections. This work provides substantial improvements over the previously available tools, and has been extensively tested by an active user community. The `eurostat` R package contributes to the growing open source ecosystem dedicated to reproducible research in computational social science and digital humanities.

Introduction

Eurostat, the statistical office of the European Union, provides a rich collection of data through its open data service¹, including thousands of data sets on European demography, economics, health, infrastructure, traffic and other topics. The statistics are often available with fine geographical resolution and include time series spanning over several years or decades.

Availability of algorithmic tools to access and analyse open data collections can greatly benefit reproducible research (Gandrud, 2013; Boettiger et al., 2015), as complete analytical workflows spanning from raw data to final publications can be made fully replicable and transparent. Dedicated software packages help to simplify, standardize, and automate analysis workflows, greatly facilitating reproducibility, code sharing, and efficient data analytics. The code for data retrieval need to be customized to specific data sources to accommodate variations in raw data formats, access details, and typical use cases so that the end users can avoid repetitive programming tasks and save time. A number of packages for governmental and other sources have been designed to meet these demands, including packages for the Food and Agricultural Organization (FAO) of the United Nations (`FAOSTAT`; Kao et al. 2015), World Bank (`WDI`; Arel-Bundock 2013), national statistics authorities (`pxweb`; Magnusson et al. 2014), Open Street Map (`osmar`; Eugster and Schlesinger 2012) and many other sources.

A dedicated R package for the Eurostat open data has been missing. The `eurostat` package fills this gap. It expands the capabilities of our earlier `statfi` (Lahti et al., 2013a) and `smarterpoland` (Biecek, 2015) packages. Since its first CRAN release in 2014, the `eurostat` package has been developed by several active contributors based on frequent feedback from the user community. We are now reporting mature version that has been improved and tested by multiple users, and applied in several case studies by us and others (Kenett and Shmueli, 2016). The Eurostat database has three services for programmatic data access: a bulk download, json/unicode, and SDMX web service; we provide targeted methods for the first two in the `eurostat` package; generic tools for the SDMX format are available via the `rsdmx` package (Blondel, 2017). The bulk download provides single files, which is convenient and fast for retrieving major parts of data. More light-weight json methods allow data subsetting before download and may be preferred in more specific retrieval tasks but the query size is limited to 50 categories. Finally, the package can be used to download custom administrative boundaries by EuroGeographics© that allow seamless visualization of the data on the European map.

Specific versions of the Eurostat data can be accessed with the `datamart` (Weinert, 2014), `quandl` (McTaggart et al., 2015), `pdfetch` (Reinhart, 2015), and `rsdmx` packages. Unlike these generic database packages, `eurostat` is particularly tailored for the Eurostat open data service. It depends on further R packages including `classInt` (Bivand, 2015), `httr` (Wickham, 2016), `jsonlite` (Ooms, 2014), `readr` (Wickham and Francois, 2015), `sp` (Bivand et al., 2013), and `stringi` (Gagolewski and Tartanus, 2015). The following CRAN task views are particularly relevant *ReproducibleResearch*, *SocialSciences*, *Spatial*, *SpatioTemporal*, *TimeSeries*, *WebTechnologies*. The package is part of rOpenGov (Lahti et al., 2013b) reproducible research initiative for computational social science and digital humanities.

In summary, the `eurostat` package provides custom tools for Eurostat open data. Key features such as cache, date formatting, tidy data principles (Wickham, 2014), and `tibble` (Wickham et al., 2016) data format support seamless integration with other tools for data manipulation and analysis. This article provides an overview of the core functionality in the current CRAN release version (3.1.1). A comprehensive documentation and source code are available via the package homepage in Github².

¹<http://ec.europa.eu/eurostat/data/database>

²<http://ropengov.github.io/eurostat>

Search and download commands

To install and load the CRAN release version, just type the standard installation command in R.

```
install.packages("eurostat")
library("eurostat")
```

The database table of contents is available on-line³, or can be downloaded in R with `get_eurostat_toc()`. A more focused search is provided by the `search_eurostat()` function.

```
query <- search_eurostat("road accidents", type = "table")
```

This seeks data on road accidents. The `type` argument limits the search on a selected data set type, one of three hierarchical levels including *table*, which resides in *dataset*, which is in turn stored in a *folder*. Values in the `code` column of the `search_eurostat()` function output provide data identifiers for subsequent download commands. Alternatively, these identifiers can be browsed at the Eurostat open data service; check the codes in the Data Navigation Tree listed after each dataset in parentheses. Let us look at the data identifier and title for the first entry of the query data.

```
query$code[[1]]
[1] "tsdtr420"
```

```
query$title[[1]]
[1] "People killed in road accidents"
```

Let us next retrieve the data set with this identifier.

```
dat <- get_eurostat(id = "tsdtr420", time_format = "num")
```

Here we used the numeric time format as it is more convenient for annual time series than the default date format. The transport statistics returned by this function call (Table 1) could be filtered before download with the `filters` argument, where the list names and values refer to Eurostat variable and observation codes, respectively. To retrieve transport statistics for specific countries, for instance, use the `get_eurostat` function.

```
countries <- c("UK", "SK", "FR", "PL", "ES", "PT")
t1 <- get_eurostat("tsdtr420", filters = list(geo = countries))
```

	unit	sex	geo	time	values
1	NR	T	AT	1999.00	1079.00
2	NR	T	BE	1999.00	1397.00
3	NR	T	CZ	1999.00	1455.00
4	NR	T	DE	1999.00	7772.00
5	NR	T	DK	1999.00	514.00
6	NR	T	EL	1999.00	2116.00

Table 1: First entries of the road accident data set retrieved with `get_eurostat(id = "tsdtr420", time_format = "num")`.

	unit	sex	geo	time	values
1	Number	Total	Austria	1999.00	1079.00
2	Number	Total	Belgium	1999.00	1397.00
3	Number	Total	Czech Republic	1999.00	1455.00
4	Number	Total	Germany (until 1990 former territory of the FRG)	1999.00	7772.00
5	Number	Total	Denmark	1999.00	514.00
6	Number	Total	Greece	1999.00	2116.00

Table 2: The `get_eurostat()` output (Table 1) converted into human-readable labels with `label_eurostat()`.

A subsequent visualization reveals a decreasing trend of road accidents over time in Figure 1.

³<http://ec.europa.eu/eurostat/data/database>

```
ggplot(t1, aes(x = time, y = values, color = geo, group = geo, shape = geo)) +
  geom_point(size = 4) + geom_line() + theme_bw() +
  ggtitle("Road accidents") + xlab("Year") + ylab("Victims (n)") +
  theme(legend.position = "none") +
  ggrepel::geom_label_repel(data = t1 %>% group_by(geo) %>% na.omit() %>%
  filter(time %in% c(min(time), max(time))), aes(fill = geo, label = geo), color = "white")
```

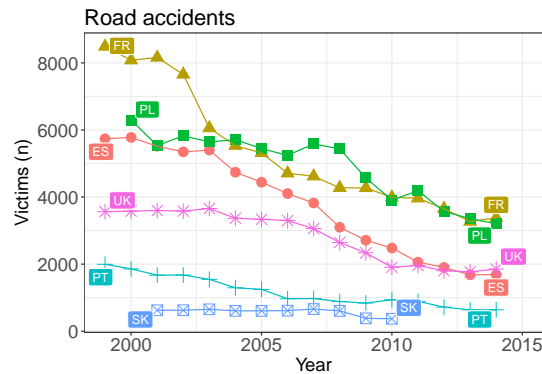


Figure 1: Timeline indicating the number of people killed in road accidents in various countries based on Eurostat open data retrieved with the **eurostat** R package.

Utilities

Many entries in Table 1 are not readily interpretable, but a simple call `label_eurostat(dat)` can be used to convert the original identifier codes into human-readable labels (Table 2) based on translations in the Eurostat database. Labels are available in English, French and German languages.

The Eurostat database includes a variety of demographic and health indicators. We see, for instance, that overweight varies remarkably across different age groups (Figure 2A). Sometimes the data sets require more complicated pre-processing. Let's consider, for instance, the distribution of renewable energy sources in different European countries. In order to summarise such data one needs to first aggregate a multitude of possible energy sources into a smaller number of coherent groups. Then one can use standard R tools to process the data, chop country names, filter countries depending on production levels, normalize the within country production. After a series of transformations (see Appendix for the source code) we can finally plot the data to discover that countries vary a lot in terms of renewable energy sources (Figure 2B). Three-dimensional data sets such as this can be conveniently visualized as triangular maps by using the `plotrix` (Lemon, 2006) package.

The data sets are stored in cache by default to avoid repeated downloads of identical data and to speed up the analysis. Storing an exact copy of the retrieved raw data on the hard disk will also support reproducibility when the source database is constantly updated.

Geospatial information

Map visualizations

The indicators in the Eurostat open data service are typically available as annual time series grouped by country, and sometimes at more refined temporal or geographic levels. Eurostat provides complementary geospatial data on the corresponding administrative statistical units to support visualizations at the appropriate geographic resolution. The geospatial data sets are available as standard shapefiles⁴. Let us look at disposable income of private households (data identifier `tgs00026`⁵). This is provided at the geographic NUTS2 regions, the intermediate territorial units in the Eurostat regional classifications, roughly corresponding to provinces or states in each country⁶ (Figure 3). The map can be generated with the following code chunk.

⁴<http://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-units>

⁵<http://ec.europa.eu/eurostat/en/web/products-datasets/-/TGS00026>

⁶<http://ec.europa.eu/eurostat/web/nuts/overview>

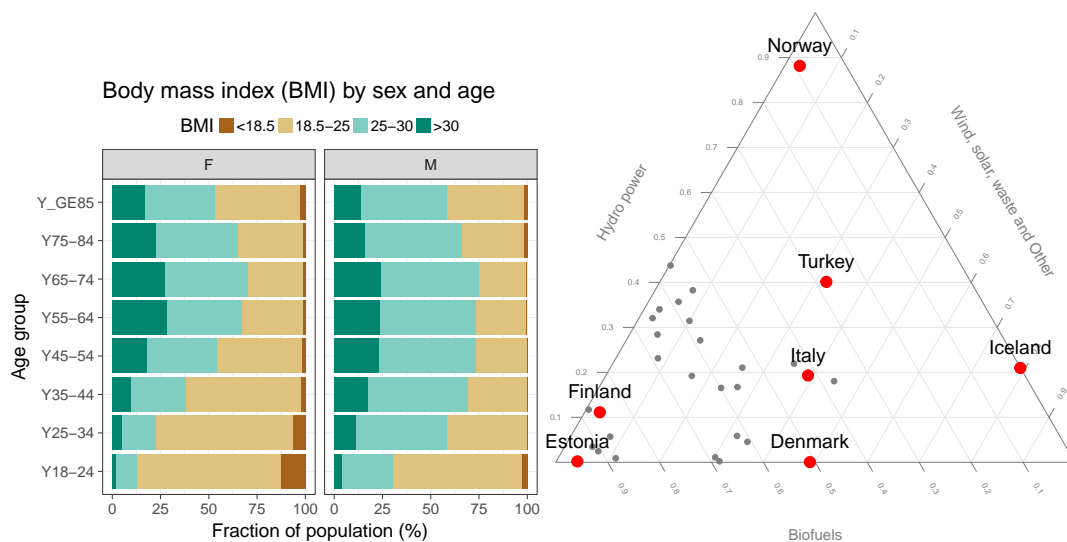


Figure 2: A The body-mass index (BMI) in different age groups in Poland (Eurostat table hlth_ehis_de1). B Production of renewable energy in various countries in 2013 (Eurostat table ten00081). See the Appendix for the source code.

```
# Load the required libraries
library(eurostat)
library(dplyr)
library(ggplot2)

# Download and manipulate tabular data
get_eurostat("tgs00026", time_format = "raw") %>%
  # Subset to year 2005 and NUTS-3 level
  dplyr::filter(time == 2005, nchar(as.character(geo)) == 4) %>%

# Classify the values the variable
dplyr::mutate(cat = cut_to_classes(values)) %>%

# Merge Eurostat data with geodata from Cisco
merge_eurostat_geodata(data = ., geocolumn = "geo", resolution = "60",
  output_class = "df", all_regions = TRUE) %>%

# Plot the map
ggplot(data = ., aes(long, lat, group = group)) +
  geom_polygon(aes(fill = cat), colour = alpha("white", 1/2), size = .2) +
  scale_fill_manual(values = RColorBrewer::brewer.pal(n = 5, name = "Oranges")) +
  labs(title = "Disposable household income") +
  coord_map(project = "orthographic", xlim = c(-22, 34), ylim = c(35, 70)) +
  theme_minimal() +
  guides(fill = guide_legend(title = "EUR per Year",
    title.position = "top", title.hjust = 0))
```

This demonstrates how the Eurostat statistics and geospatial data, retrieved with the `eurostat` package, can be combined with other utilities, in this case the `maptools` (Bivand and Lewin-Koh, 2015), `rgdal` (Bivand et al., 2015), `rgeos` (Bivand and Rundel, 2015), `scales` (Wickham, 2015a), and `stringr` (Wickham, 2015b) R packages.

Standard country groupings

To facilitate the analysis and visualization of standard European country groups, the `eurostat` package includes ready-made country code lists. The list of EFTA countries (Table 3), for instance, is retrieved with the `data` command.

```
data(efta_countries)
```

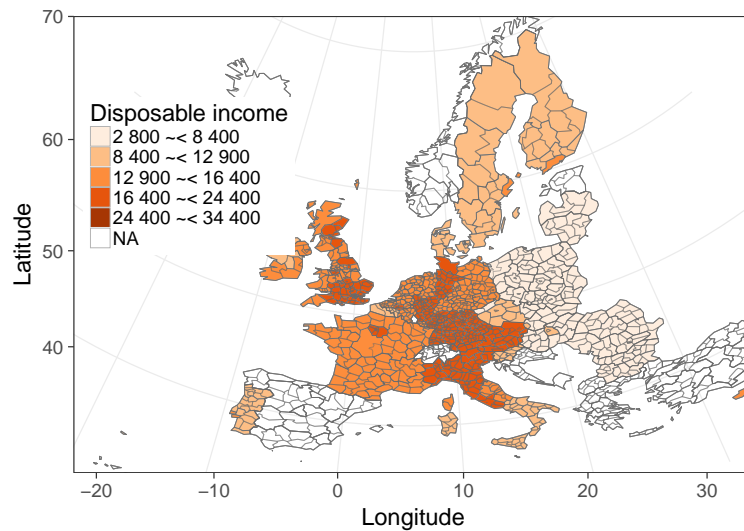


Figure 3: Disposable income of private households across NUTS2-level national regions in European countries. The household income statistics provided by Eurostat and the administrative boundaries by EuroGeographics© were obtained via the Eurostat open data service with the `eurostat` R package.

	code	name
1	IS	Iceland
2	LI	Liechtenstein
3	NO	Norway
4	CH	Switzerland

Table 3: The EFTA country listing from the `eurostat` R package.

Similar lists are available for Euro area (`ea_countries`), EU (`eu_countries`) and the EU candidate countries (`eu_candidate_countries`). These auxiliary data sets facilitate smooth selection of specific country groups for a closer analysis. The full name and a two-letter identifier are provided for each country according to the Eurostat database. The country codes follow the ISO 3166-1 alpha-2 standard, except that GB and GR are replaced by UK (United Kingdom) and EL (Greece) in the Eurostat database, respectively. Linking these country codes with external data sets can be facilitated by conversions between different country coding standards with the `countrycode` package (Arel-Bundock, 2014).

Discussion

By combining programmatic access to data with custom analysis and visualization tools it is possible to facilitate a seamless automation of the complete analytical workflow from raw data to statistical summaries and final publication. The package supports automated and transparent data retrieval from institutional data repositories, featuring options such as search, subsetting and cache. Moreover, it provides several custom functions to facilitate the Eurostat data analysis and visualization. These tools can be used by researchers and statisticians in academia, government, and industry, and their applicability has been demonstrated in recent, independent publications (Kenett and Shmueli, 2016).

The `eurostat` R package provides a convenient set of tools to access open data from Eurostat, together with a comprehensive documentation and open source code via the package homepage. The documentation includes simple examples for individual functions, a generic package tutorial, and more advanced case studies on data processing and visualization. The package follows best practices in open source software development, taking advantage of version control, automated unit tests, continuous integration, and collaborative development (Perez-Riverol et al., 2016).

The source code can be freely used, modified and distributed under a modified BSD-2-clause license⁷. We value feedback from the user community, and the package has already benefited greatly from the user bug reports and feature requests, which can be systematically provided through the

⁷<https://opensource.org/licenses/BSD-2-Clause>

Github issue tracker⁸; advanced users can also implement and contribute new features by making pull requests. Indeed, these collaborative features have been actively used during the package development. We are committed to active maintenance and development of the package, and hope that this will encourage further feedback and contributions from the user community.

Acknowledgements

We are grateful to all package contributors, including François Briatte, Joonas Lehtomäki, Oliver Reiter, and Wietse Dol, and to Eurostat for maintaining the open data service. This work is in no way officially related to or endorsed by Eurostat. The work has been partially funded by Academy of Finland (decisions 295741, 307127 to LL), and is part of rOpenGov⁹.

Bibliography

- V. Arel-Bundock. *WDI: World Development Indicators (World Bank)*, 2013. URL <http://CRAN.R-project.org/package=WDI>. R package version 2.4. [p385]
- V. Arel-Bundock. *countrycode: Convert Country Names and Country Codes*, 2014. URL <http://CRAN.R-project.org/package=countrycode>. R package version 0.18. [p389]
- P. Biecek. *SmarterPoland: Tools for Accessing Various Datasets Developed by the Foundation SmarterPoland.pl*, 2015. URL <http://CRAN.R-project.org/package=SmarterPoland>. R package version 1.5. [p385]
- R. Bivand. *classInt: Choose Univariate Class Intervals*, 2015. URL <https://CRAN.R-project.org/package=classInt>. R package version 0.1-23. [p385]
- R. Bivand and N. Lewin-Koh. *maptools: Tools for Reading and Handling Spatial Objects*, 2015. URL <http://CRAN.R-project.org/package=maptools>. R package version 0.8-37. [p388]
- R. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine - Open Source (GEOS)*, 2015. URL <http://CRAN.R-project.org/package=rgeos>. R package version 0.3-14. [p388]
- R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2015. URL <http://CRAN.R-project.org/package=rgdal>. R package version 1.0-7. [p388]
- R. S. Bivand, E. Pebesma, and V. Gómez-Rubio. *Classes for Spatial Data in R*, volume 10. Springer, second edition, 2013. URL <https://doi.org/10.1007/978-1-4614-7618-4>. [p385]
- E. Blondel. *r sdmx: Tools for Reading SDMX Data and Metadata*, 2017. URL <https://CRAN.R-project.org/package=r sdmx>. R package version 0.5-8. [p385]
- C. Boettiger, S. Chamberlain, E. Hart, and K. Ram. Building software, building community: Lessons from the ropensci project. *Journal of Open Research Software*, 3(1), 2015. URL <https://doi.org/10.5334/jors.bu>. [p385]
- M. J. A. Eugster and T. Schlesinger. OpenStreetMap and R. *R Journal*, 5(1):53–63, 2012. [p385]
- M. Gagolewski and B. Tartanus. *R Package stringi: Character String Processing Facilities*, 2015. URL <https://doi.org/10.5281/zenodo.19071>. [p385]
- C. Gandrud. *Reproducible Research with R and R Studio*. Chapman & Hall/CRC, 2013. [p385]
- M. C. J. Kao, M. Gesmann, and F. Gheri. *FAOSTAT: Download Data from the FAOSTAT Database of the Food and Agricultural Organization (FAO) of the United Nations*, 2015. URL <http://CRAN.R-project.org/package=FAOSTAT>. R package version 2.0. [p385]
- D. R. S. Kenett and G. Shmueli. *Information Quality: The Potential of Data and Analytics to Generate Knowledge*. John Wiley & Sons, 2016. URL <https://doi.org/10.1002/9781118890622>. [p385, 389]
- L. Lahti, J. Parkkinen, and J. Lehtomäki. *statfi* R Package, 2013a. URL <https://cran.r-project.org/src/contrib/Archive/statfi/>. [p385]
- L. Lahti, J. Parkkinen, J. Lehtomäki, and M. Kainu. rOpenGov: Open Source Ecosystem for Computational Social Sciences and Digital Humanities. Int'l Conf. on Machine Learning - ICML/MLOSS Open Source Software workshop, 2013b. URL <http://ropengov.github.io>. [p385]

⁸<https://github.com/ropengov/eurostat/issues>

⁹<https://github.com/ropengov.io>

- J. Lemon. Plotrix: a Package in the Red Light District of R. *R News*, 6(4):8–12, 2006. [p387]
- M. Magnusson, L. Lahti, and L. Hansson. pxweb: R Tools for the Px-Web API, 2014. URL <http://CRAN.R-project.org/package=pxweb>. R package version 0.5.57. [p385]
- R. McTaggart, G. Daroczi, and C. Leung. Quandl: API Wrapper for Quandl.com, 2015. URL <http://CRAN.R-project.org/package=Quandl>. R package version 2.7.0. [p385]
- J. Ooms. The Jsonlite Package: a Practical and Consistent Mapping between JSON Data and R Objects. *arXiv:1403.2805 [stat.CO]*, 2014. [p385]
- Y. Perez-Riverol, L. Gatto, R. Wang, T. Sachsenberg, J. Uszkoreit, F. da Veiga Leprevost, C. Fufezan, T. Ternent, S. J. Eglén, D. S. Katz, T. J. Pollard, A. Konovalov, R. M. Flight, K. Blin, and J. A. Vizcaíno. Ten Simple Rules for Taking Advantage of Git and GitHub. *PLoS Computational Biology*, 12(7): e1004947, 2016. URL <https://doi.org/10.1371/journal.pcbi.1004947>. [p389]
- A. Reinhart. *pdfetch: Fetch Economic and Financial Time Series Data from Public Sources*, 2015. URL <http://CRAN.R-project.org/package=pdfetch>. R package version 0.1.7. [p385]
- K. Weinert. *datamart: Unified Access to your Data Sources*, 2014. URL <http://CRAN.R-project.org/package=datamart>. R package version 0.5.2. [p385]
- H. Wickham. Tidy Data. *Journal of Statistical Software*, 59(10), 2014. URL <https://doi.org/10.18637/jss.v059.i10>. [p385]
- H. Wickham. *scales: Scale Functions for Visualization*, 2015a. URL <http://CRAN.R-project.org/package=scales>. R package version 0.3.0. [p388]
- H. Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2015b. URL <http://CRAN.R-project.org/package=stringr>. R package version 1.0.0. [p388]
- H. Wickham. *httr: Tools for Working with URLs and HTTP*, 2016. URL <https://CRAN.R-project.org/package=httr>. R package version 1.2.1. [p385]
- H. Wickham and R. Francois. *readr: Read Tabular Data*, 2015. URL <https://CRAN.R-project.org/package=readr>. R package version 0.2.2. [p385]
- H. Wickham, R. Francois, and K. Müller. *tibble: Simple Data Frames*, 2016. URL <https://CRAN.R-project.org/package=tibble>. R package version 1.1. [p385]

Leo Lahti
Department of Mathematics and Statistics
PO Box 20014 University of Turku
Finland
ORCID: 0000-0001-5537-637X
leo.lahti@iki.fi

Janne Huovari
Pellervo Economic Research PTT
Eerikinkatu 28 A, 00180 Helsinki
Finland
janne.huovari@ptt.fi

Markus Kainu
Research Department, The Social Insurance Institution of Finland
PO Box 450, 00101 Helsinki
Finland
markus.kainu@kela.fi

Przemysław Biecek
Faculty of Mathematics and Information Science
Warsaw University of Technology
Koszykowa 75, 00-662 Warsaw
Poland
P.Biecek@mimuw.edu.pl

Appendix

The full source code for this manuscript is available at the package homepage¹⁰. Source code for the obesity example (Figure 2A) is as follows.

```
library(dplyr)
tmp1 <- get_eurostat("hlth_ehis_de1", time_format = "raw")
tmp1 %>%
  dplyr::filter(isced97 == "TOTAL" ,
                sex != "T", age != "TOTAL", geo == "PL") %>%
  mutate(BMI = factor(bmi,
                     levels=c("LT18P5","18P5-25","25-30","GE30"),
                     labels=c("<18.5", "18.5-25", "25-30",">30"))) %>%
  arrange(BMI) %>%

  ggplot(aes(y = values, x = age, fill = BMI)) + geom_bar(stat = "identity") +
  facet_wrap(~sex) + coord_flip() +
  theme(legend.position = "top") +
  ggtitle("Body mass index (BMI) by sex and age") +
  xlab("% of population") + scale_fill_brewer(type = "div")
```

Source code for the renewable energy example (Figure 2B).

```
# All sources of renewable energy are to be grouped into three sets
dict <- c("Solid biofuels (excluding charcoal)" = "Biofuels",
         "Biogasoline" = "Biofuels",
         "Other liquid biofuels" = "Biofuels",
         "Biodiesels" = "Biofuels",
         "Biogas" = "Biofuels",
         "Hydro power" = "Hydro power",
         "Tide, Wave and Ocean" = "Hydro power",
         "Solar thermal" = "Wind, solar, waste and Other",
         "Geothermal Energy" = "Wind, solar, waste and Other",
         "Solar photovoltaic" = "Wind, solar, waste and Other",
         "Municipal waste (renewable)" = "Wind, solar, waste and Other",
         "Wind power" = "Wind, solar, waste and Other",
         "Bio jet kerosene" = "Wind, solar, waste and Other")

# Some cleaning of the data is required
energy3 <- get_eurostat("ten00081") %>%
  label_eurostat(dat) %>%
  filter(time == "2013-01-01",
         product != "Renewable energies") %>%
  mutate(nproduct = dict[as.character(product)], # just three categories
         geo = gsub(geo, pattern=" \\(.*", replacement="")) %>%
  select(nproduct, geo, values) %>%
  group_by(nproduct, geo) %>%
  summarise(svalue = sum(values)) %>%
  group_by(geo) %>%
  mutate(tvalue = sum(svalue), svalue = svalue/sum(svalue)) %>%
  filter(tvalue > 1000) %>%
  spread(nproduct, svalue)

# Triangle plot
positions <- plotrix::triax.plot(as.matrix(energy3[, c(3,5,4)]),
  show.grid = TRUE, label.points = FALSE, point.labels = energy3$geo,
  col.axis = "gray50", col.grid = "gray90",
  pch = 19, cex.axis = 1.1, cex.ticks = 0.7, col = "grey50")

ind <- which(energy3$geo %in% c("Norway", "Iceland","Denmark","Estonia", "Turkey", "Italy", "Finland"))
df <- data.frame(positions$xypos, geo = energy3$geo)
points(df$x[ind], df$y[ind], cex = 2, col = "red", pch = 19)
text(df$x[ind], df$y[ind], df$geo[ind], adj = c(0.5,-1), cex = 1.5)
```

¹⁰<http://ropengov.github.io/eurostat>

PGEE: An R Package for Analysis of Longitudinal Data with High-Dimensional Covariates

by Gul Inan and Lan Wang

Abstract We introduce an R package **PGEE** that implements the penalized generalized estimating equations (GEE) procedure proposed by Wang et al. (2012) to analyze longitudinal data with a large number of covariates. The **PGEE** package includes three main functions: `CVfit`, `PGEE`, and `MGEE`. The `CVfit` function computes the cross-validated tuning parameter for penalized generalized estimating equations. The function `PGEE` performs simultaneous estimation and variable selection for longitudinal data with high-dimensional covariates; whereas the function `MGEE` fits unpenalized GEE to the data for comparison. The R package **PGEE** is illustrated using a yeast cell-cycle gene expression data set.

Introduction

Longitudinal data arises from repeated measurements on the same subjects over time. A popular approach to analyzing longitudinal data is generalized estimating equations (GEE), which were proposed by Liang and Zeger (1986) and Zeger and Liang (1986). The GEE approach fits a marginal regression model to the longitudinal data. Instead of specifying the full joint likelihood, it only requires to specify the first two marginal moments. This is particularly attractive when the responses are discrete as specifying a joint distribution for multivariate discrete distribution is known to be challenging. Furthermore, although the GEE procedure relies on a working correlation model, it produces a consistent and asymptotically normal estimator even if the working correlation structure is misspecified. If the specified working correlation structure is close to the true correlation structure, further efficiency gain can be expected. Some commonly used working correlation structures include the exchangeable (Exch), first-order autoregressive (Ar(1)), stationary-1-dependent (MV_1) and so on. The generalized estimating equations are now implemented in two nice R packages: the `gee` package (Carey, 2015) and the `geepack` package (Halekoh et al., 2006).

With the advent of technology in data-collection, longitudinal data with a large number of covariates, in other words, high-dimensional longitudinal data, have now been commonly observed in fields such as health and genomic studies, economics and behavioral sciences. Including redundant covariates in model results in loss of accuracy in both estimation and inference. In the modern “large n , diverging p ” framework, Wang (2011) studied the consistency and asymptotic normality of GEE regression estimators and verified the validity of the sandwich variance formula of GEE estimators and the large-sample Wald test under regularity conditions. Wang et al. (2012) further proposed penalized GEE for simultaneous variable selection and estimation for the cases where the number of covariates in the model is large. Similarly as GEE, the penalized GEE procedure only requires to specify the first two marginal moments and a working correlation matrix and assumes that missing data is valid only under missing completely at random, which means that missingness is independent of both observed and unobserved data. It leads to consistent variable selection performance even if the working correlation structure is misspecified, that is, with probably approaching one, the true model is selected if it is one of the candidate models. Recently, there has been growing interest in high-dimensional longitudinal data analysis, see for example Lian et al. (2014) and Wang et al. (2014).

In this paper, we present the R package **PGEE** (Inan et al., 2017) which implements the penalized generalized estimating equations procedure in Wang et al. (2012) to analyze the longitudinal data with high-dimensional covariates. The package **PGEE** is available on CRAN at <https://cran.r-project.org/web/packages/PGEE>. The rest of the paper is organized as follows. Section T.2 provides a brief overview for both GEE and PGEE. Section T.3 describes the main features of the functions in the **PGEE** package. Section T.4 illustrates the use of **PGEE** via a yeast cell-cycle gene expression data set. Section T.5 concludes the paper.

An overview for penalized generalized estimating equations

Data structure

Consider a longitudinal study where for the i th ($i = 1, 2, \dots, N$) subject at time t ($t = 1, 2, \dots, n_i$) we observe a response variable Y_{it} and a $p \times 1$ vector of covariates X_{it} . Let $\mathbf{Y}_i = (Y_{i1}, \dots, Y_{in_i})^T$ and

$\mathbf{X}_i = (\mathbf{X}_{i1}, \dots, \dots, \mathbf{X}_{in_i})^T$ denote $n_i \times 1$ vector of responses and $n_i \times p$ matrix of covariates for the i th subject, respectively. The observations obtained from the same subject are correlated whereas those obtained from different subjects are assumed to be independent.

Background on generalized estimating equations

Liang and Zeger (1986) assume that the first two marginal moments of Y_{ij} are $\mu_{it}(\boldsymbol{\beta}) := \mathbb{E}(Y_{it}|\mathbf{X}_{it}) = \mu(\theta_{it})$ and $\sigma_{it}^2(\boldsymbol{\beta}) := \text{Var}(Y_{it}|\mathbf{X}_{it}) = \dot{\mu}(\theta_{it})$, where $\theta_{it} = \mathbf{X}_{it}^T \boldsymbol{\beta}$, and $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^T$ is $p \times 1$ vector of unknown regression coefficients of interest, $i = 1, 2, \dots, N, t = 1, 2, \dots, n_i$. These moment assumptions would follow when the marginal response variable has a canonical exponential family distribution with scaling parameter one.

Let $\mathbf{V}_i = \text{Var}(\mathbf{Y}_i|\mathbf{X}_{it})$ be the $n_i \times n_i$ covariance matrix of the i th subject, $i = 1, \dots, N$. In practice, Liang and Zeger (1986) suggest to estimate \mathbf{V}_i via a working correlation structure. Specifically, let

$$\mathbf{V}_i = \mathbf{A}_i^{1/2} R_{n_i}(\boldsymbol{\alpha}) \mathbf{A}_i^{1/2}, \tag{1}$$

where \mathbf{A}_i is an $n_i \times n_i$ diagonal matrix with the marginal variance of responses on the diagonals, and $R_{n_i}(\boldsymbol{\alpha})$ represents an $n_i \times n_i$ working correlation matrix indexed by a vector of parameters $\boldsymbol{\alpha}$. From now on, we will use $R(\boldsymbol{\alpha})$ rather than $R_{n_i}(\boldsymbol{\alpha})$ for simplicity. The popular choices for $R(\boldsymbol{\alpha})$ may be independence (Indep), exchangeable (Exch), first-order autoregressive (Ar(1)), stationary-m-dependent (MV_m), and non-stationary-m-dependent (NMV_m) (m denotes the lag order), and unstructured (UN) working correlation structure. A good review of the commonly used working correlation matrices is given in Horton and Lipsitz (1999).

Let $\mathbf{A}_i(\boldsymbol{\beta}) = \text{diag}(\sigma_{i1}^2(\boldsymbol{\beta}), \dots, \sigma_{in_i}^2(\boldsymbol{\beta}))$ and $\boldsymbol{\mu}_i(\boldsymbol{\beta}) = (\mu_{i1}(\boldsymbol{\beta}), \dots, \mu_{in_i}(\boldsymbol{\beta}))^T$. Liang and Zeger (1986) proposed to estimate the regression parameters $\boldsymbol{\beta}$ by solving the following set of estimating equations

$$\mathbf{S}(\boldsymbol{\beta}) = \sum_{i=1}^N \mathbf{X}_i^T \mathbf{A}_i^{1/2} \hat{R}^{-1}(\boldsymbol{\alpha}) \mathbf{A}_i^{-1/2} (\mathbf{Y}_i - \boldsymbol{\mu}_i) = 0, \tag{2}$$

where $\hat{R}^{-1}(\boldsymbol{\alpha})$ denotes the estimated working correlation matrix. The estimating equations can be solved using a modified Fisher scoring algorithm. Within the iterative Fisher scoring algorithm, the parameter $\boldsymbol{\alpha}$ in $R(\boldsymbol{\alpha})$ can be estimated by residual-based moment method, see Hardin and Hilbe (2003). Liang and Zeger (1986) showed that the resulted estimator is consistent even if R is misspecified.

Penalized generalized estimating equations for longitudinal data with high-dimensional covariates

With high-dimensional covariates, it is often reasonable to assume that many of these covariates are not relevant for modeling the marginal mean of the response variable, in other words, the regression coefficients vector $\boldsymbol{\beta}$ can be assumed to be sparse in the sense that most of its components are exactly zero. Wang et al. (2012) introduced penalized generalized estimating equations (PGEE) for simultaneous estimation and variable selection in this setting. More specifically, they propose to estimate $\boldsymbol{\beta}$ by $\hat{\boldsymbol{\beta}}$, which solves the following set of penalized estimating equations

$$\mathbf{U}(\boldsymbol{\beta}) = \mathbf{S}(\boldsymbol{\beta}) - \mathbf{q}_\lambda(|\boldsymbol{\beta}|) \circ \text{sign}(\boldsymbol{\beta}) \tag{3}$$

where $\mathbf{q}_\lambda(|\boldsymbol{\beta}|) = (q_\lambda(|\beta_1|), \dots, q_\lambda(|\beta_p|))^T$, $\text{sign}(\boldsymbol{\beta}) = (\text{sign}(\beta_1), \dots, \text{sign}(\beta_p))^T$, and $\mathbf{q}_\lambda(|\boldsymbol{\beta}|) \circ \text{sign}(\boldsymbol{\beta})$ denotes the Hadamard product (element-wise product) of these two vectors. The penalty function $q_\lambda(|\beta_j|)$, the j th component of $\mathbf{q}_\lambda(|\boldsymbol{\beta}|)$, takes a zero value for a large value of $|\beta_j|$ and takes a large value for a small value of $|\beta_j|$. Consequently, the generalized estimating equation $S(\beta_j)$, the j th component of $\mathbf{S}(\boldsymbol{\beta})$, is not penalized if $|\beta_j|$ is large in magnitude, whereas $S(\beta_j)$ is penalized if $|\beta_j|$ is smaller than a cut-off value (greater than zero). Hence, the role of the penalty function $q_\lambda(|\beta_j|)$ is to shrink the estimates of small coefficients toward zero. The coefficients whose estimates are shrunken to zero are excluded from the final model. The cut-off value is chosen as 10^{-3} as in Cho and Qu (2013), Wang et al. (2012), and Wang et al. (2007). The penalty has a tuning parameter λ that controls the model complexity. Wang et al. (2012) studied the SCAD penalty function (Fan and Li, 2001) which is defined on $[0, +\infty]$ as

$$q_\lambda(t) = \lambda \left\{ I(t < \lambda) + \frac{(a\lambda - t)_+}{(a - 1)\lambda} I(t \geq \lambda) \right\}, \tag{4}$$

where $\lambda \geq 0$, $a > 2$ and $b_+ = bI(b > 0)$ for a real number b . Following the recommendation of Fan and Li (2001), we use $a = 3.7$ and find it usually works well. The nonconvex SCAD penalty function alleviates the main drawback of L_1 penalty function in that it avoids over-penalizing large coefficients and hence leads to consistent variable selection under more relaxed conditions. Under regularity conditions, the estimated coefficients for redundant covariates are shrunken to exactly zero.

Motivated by Johnson et al. (2008), Wang et al. (2012) proposed to solve the penalized estimated equations in Equation (3) by combining the minorization-maximization (MM) algorithm with a Newton-Raphson (NR) algorithm. At the j th iteration,

$$\hat{\beta}^j = \hat{\beta}^{j-1} + [\mathbf{H}(\hat{\beta}^{j-1}) + \mathbf{NE}(\hat{\beta}^{j-1})]^{-1} [\mathbf{S}(\hat{\beta}^{j-1}) - \mathbf{NE}(\hat{\beta}^{j-1})\hat{\beta}^{j-1}], \tag{5}$$

where

$$\begin{aligned} \mathbf{H}(\hat{\beta}^{j-1}) &= \sum_{i=1}^N \mathbf{X}_i^T \mathbf{A}_i^{1/2}(\hat{\beta}^{j-1}) \hat{R}^{-1}(\alpha) \mathbf{A}_i^{1/2}(\hat{\beta}^{j-1}) \mathbf{X}_i \\ \mathbf{E}(\hat{\beta}^{j-1}) &= \text{diag} \left\{ \frac{q_\lambda(|\hat{\beta}_1|+)}{\epsilon + |\hat{\beta}_1|}, \dots, \frac{q_\lambda(|\hat{\beta}_p|+)}{\epsilon + |\hat{\beta}_p|} \right\}, \end{aligned} \tag{6}$$

where ϵ is a small value (e.g., 10^{-6}). Wang et al. (2012) derived the asymptotic theory of PGEE in a high-dimensional framework where the number of covariates p increases as N increases, and p can reach the same order as N . An important feature of PGEE is that even if the working correlation structure is misspecified, the consistency of model selection holds, that is, with probability approaching one, it correctly identifies the zero coefficients to be zero and the nonzero coefficients to be nonzero. They also suggested a sandwich formula to estimate the asymptotic covariance matrix of the penalized GEE estimator as follows:

$$\text{Cov}(\hat{\beta}) \approx [\mathbf{H}(\hat{\beta}) + \mathbf{NE}(\hat{\beta})]^{-1} \mathbf{M}(\hat{\beta}) [\mathbf{H}(\hat{\beta}) + \mathbf{NE}(\hat{\beta})]^{-1}, \tag{7}$$

where

$$\mathbf{M}(\hat{\beta}) = \sum_{i=1}^N \mathbf{X}_i^T \mathbf{A}_i^{1/2}(\hat{\beta}) \hat{R}^{-1} \epsilon_i \epsilon_i^T \hat{R}^{-1} \mathbf{A}_i^{1/2}(\hat{\beta}) \mathbf{X}_i, \tag{8}$$

where $\epsilon_i = \mathbf{A}_i^{-1/2}(\hat{\beta})(\mathbf{Y}_i - \mu_i)$.

The tuning parameter λ in Equation (4) can be selected using K -fold cross-validation, where K is a positive integer. The data is divided into non-overlapping K sub-samples of equal sizes. The k th sub-sample being left out as the testing data set, and the remaining data are used as the training data set, $k = 1, \dots, K$. We use the set N_{-k} to denote the indice set of the subjects in the training data set and use $|N_{-k}|$ to denote the cardinality of N_{-k} . We fit the PGEE under working independence assumption using the training data and then evaluate the prediction error using the test data by $PE_{-k}(\lambda)$, which is defined as

$$PE_{-k}(\lambda) = \frac{1}{|N_{-k}|} \sum_{i \in N_{-k}} \frac{1}{n_i} \sum_{t=1}^{n_i} (Y_{it} - g(\mathbf{X}_{it}^T \hat{\beta}))^2,$$

We repeat the above computation for each of the K subsamples, and the overall cross-validated prediction error is given by

$$CV(\lambda) = \frac{1}{K} \sum_{k=1}^K PE_{-k}(\lambda) \tag{9}$$

Given a set of λ values over a grid, we choose the value of λ that yields the smallest $CV(\lambda)$.

PGEE estimation algorithm

The algorithm for solving penalized generalized estimating equations is summarized as follows:

1. Determine a reasonable grid of values for λ .
2. Given a value of λ :
 - Assign an initial value for β .
 - Compute $\mathbf{U}(\tilde{\beta})$, $\mathbf{H}(\tilde{\beta})$, and $\mathbf{E}(\tilde{\beta})$ for current value of $\tilde{\beta}$.

- Update the current estimate of β through formula in Equation (5).
 - Stop the iteration sequence if a convergence criterion is satisfied such as when the L_1 norm of the difference of estimated β between iterations is below a threshold (e.g., 10^{-3}) and denote the estimated value at convergence as $\hat{\beta}$.
 - Compute the cross-validation value of λ via Equation (9).
3. Repeat step 2 for each value of λ over the grid and find the value of λ that gives the smallest cross-validation prediction error.
 4. Find the estimated parameter $\hat{\beta}$ corresponding to the selected λ , and compute the sandwich covariance matrix by Equation (7).

The PGEE package

The R package **PGEE** consists of three core functions: `CVfit`, `PGEE`, and `MGEE`, respectively. The main function `PGEE` fits PGEEs to the longitudinal data with high-dimensional covariates. Prior to model fitting with `PGEE`, the cross-validated tuning parameter should be computed via the function `CVfit`. The package also includes the function `MGEE` which fits the unpenalized GEEs to the data. The `PGEE` and `MGEE` functions are written by the authors. The R package **PGEE** depends on the R packages **MASS** (Ripley, 2015) and **mvtnorm** (Genz et al., 2015) only.

In this section, we introduce the input arguments of the functions `CVfit` and `PGEE`, whereas the function `MGEE` shares the same arguments with the function `PGEE` except the arguments `lambda`, `pindex`, and `eps`.

The usage and input arguments of `CVfit` function are summarized as follows:

```
CVfit(formula, id, data, family, scale.fix = TRUE, scale.value = 1, fold, lambda.vec,
pindex, eps, maxiter, tol)
```

The function `CVfit` applies the step 3 in the estimation algorithm in Section T.2.4 via Equation (9). It uses the function `PGEE` inside such that both functions share common input arguments. The input argument `formula` is a formula expression in the form of `response predictors` as in `lm` and `glm` functions. The argument `id` is a vector for identifying subjects/clusters and the argument `data` is a data frame which stores the response and covariates in `formula` with `id` variable as in `gee` function in R package **gee**. Please note that the function `PGEE` requires the covariates to be numeric variables and does not work with factor covariates. The argument `family` is a list of functions and expressions for defining link and variance functions. While families supported in the R package **PGEE** are `binomial`, `gaussian`, `gamma`, and `poisson`, link functions supported are `identity`, `log`, `logit`, `inverse`, `probit`, and `cloglog`. The argument `scale.fix` is a logical variable. The default value is `TRUE`. On the other hand, if `scale.fix = TRUE`, `scale.value` assigns a numeric value to which the scale parameter should be fixed at. Otherwise, the default value is 1. The arguments `fold`, `pindex`, and `eps` are the main buildings of the function `CVfit` for cross-validation. The argument `fold` is the number of folds used in cross-validation. The argument `lambda.vec` is a vector of tuning parameters used in cross-validation. The argument `pindex` is an index vector showing the parameters which are not subject to penalization. The default value is `NULL`. However, in case of a model with intercept, the intercept parameter should be never penalized. The argument `eps` is a numerical value for the ϵ used in Equation (6). The default value is 10^{-6} . The argument `maxiter` is the number of iterations that is used in the estimation algorithm. The default value is 30. The argument `tol` is the tolerance level that is used in the estimation algorithm to evaluate algorithm convergence. The default value is 10^{-3} . The function `CVfit` returns an object class of `CVfit`. Applying the function `print` to the object returned by function `CVfit` provides detailed information related to cross-validation and gives the value of λ that minimizes the cross-validated value of prediction error.

PGEEs (Wang et al., 2012) and, in turn, the function `CVfit` in R package **PGEE** accommodate the SCAD penalty. Fan and Li (2001) demonstrated that the SCAD penalty function is a popular nonconvex penalty function that satisfies three desirable properties of variable selection (e.g., sparsity, unbiasedness, and continuity) simultaneously. Recently, a number of R packages have been developed for estimation and variable selection problems in linear regression models, logistic regression models, quantile regression models, and Cox proportional hazards models for cross-sectional data with high-dimensional covariates; see the R package **ncvreg** (Breheny and Huang, 2011) for linear and logistic regression models with SCAD and MCP penalization functions, the R package **penalized** (Goeman, 2010) for generalized linear regression models and Cox proportional hazards models with L_1 and L_2 penalty functions, the R package **glmnet** (Friedman et al., 2010) for generalized linear regression models and Cox proportional hazards models with LASSO and elastic-net penalty functions, and the R package **rqPen** (Sherwood and Maidman, 2016) for quantile regression penalized quantile regression

with LASSO, SCAD, and MCP functions. However, these packages do not apply to the clustered data setting which we consider in this paper.

The usage and input arguments of PGEE function are summarized as follows:

```
PGEE(formula, id, data, na.action = NULL, family = gaussian(link = "identity"),
corstr = "independence", Mv = NULL, beta_int = NULL, R = NULL, scale.fix = TRUE,
scale.value = 1, lambda, pindex = NULL, eps = 10^-6, maxiter = 30, tol = 10^-3,
silent = TRUE)
```

The input arguments `formula`, `data`, `id`, and `family` are the same as those in the `CVfit` function. Here, the default value for `family` is `gaussian`. The argument `na.action` is a function to remove missing values from the data, where only `na.omit` is allowed. The argument `corstr` is a character string, which specifies the type of correlation structure within the repeated measurements of a subject. The correlation structures supported in the R package `PGEE` are "AR-1", "exchangeable", "fixed", "independence", "stat_M_dep", "non_stat_M_dep", and "unstructured". The default `corstr` type is "independence". If either "stat_M_dep" or "non_stat_M_dep" is specified in `corstr`, then the argument `Mv` assigns a numeric value for `Mv`, which is one minus less than the largest number of repeated measurements of a subject has in the data. Otherwise, the default value is `NULL`. When the longitudinal data is unbalanced, the use of "non_stat_M_dep" and "unstructured" is not allowed in the argument `corstr`. If `corstr = "fixed"` is specified, then the argument `R` is a user specified square correlation matrix of dimension maximum of the number of repeated measurements of a subject has in the data. Otherwise, the default value is `NULL`. The argument `beta_int` is a user specified vector of initial values for regression parameters including the intercept. The default value is `NULL` which gets initial values through fitting a `glm` to the whole longitudinal data. The argument `lambda` is a numerical value returned by `CVfit` function. The input arguments `scale.fix`, `scale.value`, `pindex`, `eps`, `maxiter`, and `tol` are the same as those in the `CVfit` function. The argument `silent` is a logical variable; if false, the regression parameter estimates at each iteration are printed. The default value is `TRUE`.

The function `PGEE` returns an object class of `PGEE`. Applying the functions `print` and `summary` to an object returned by function `PGEE` provides detailed information related to the model fitting and summarizes the results as illustrated in the next section.

The function `MGEE` closely follows the syntax of the function `gee` in the R package `gee` except that the argument `subset` for data sub-setting and the argument `contrasts` for coding factor variables in terms of dummy variables are not used in the function `MGEE`. Furthermore, while any lag order can be assumed in the argument `corstr = "AR-M"` of the function `gee`, only first-order lag is allowed in the function `MGEE` (e.g., `corstr = "AR-1"`). On the other hand, there is much discrepancy between the arguments of the function `MGEE` and the function `geeglm` in the R package `geepack` since the latter inherits its arguments mostly from the function `glm`. As the result, arguments in `geepack` such as `weights`, `subset`, `etastart`, `mustart`, `offset`, and `waves` are not available in our function `MGEE`. Its `corstr` menu consists of "AR-1", "exchangeable", "fixed", "independence", and "unstructured" structures, which is less comprehensive compared to the `corstr` menu of the function `MGEE`. Lastly, in addition to the sandwich variance estimator, the function `geeglm` offers jackknife variance estimator for data sets with small number of clusters via the argument `std.err`.

Example

In this section, we demonstrate the use of the R package `PGEE` using a yeast cell-cycle gene expression data set collected in the CDC15 experiment of [Spellman et al. \(1998\)](#). The experiment measured genome-wide mRNA levels of 6178 yeast open reading frames (ORFs) (translates DNA sequences into its corresponding amino acid sequences which will appear in the final protein). This experiment covered two cell-cycle periods, where measurements were taken at 7-minute intervals over a 119-minutes period yielding a total of 18 time points.

As discussed in [Wang et al. \(2012\)](#) and [Wang et al. \(2007\)](#), the cell-cycle process is a regulated life process where the cell grows, replicates its DNA and prepares itself for cell-division. This process is generally divided into M/G1-G1-S-G2-M stages, where M refers to "mitosis", G1 refers "GAP 1", S refers to DNA "synthesis", and G2 refers to "GAP 2", respectively. [Spellman et al. \(1998\)](#) identified a total of 800 genes that showed periodic variation during the cell-cycle process. However, to better understand the phenomenon underlying cell-cycle process, it is important to identify transcription factors (TFs) that regulate the gene expression levels of cell cycle-regulated genes. Specifically, TFs are proteins that control gene regulation by determining the rate of transcription of genetic information from DNA to mRNA.

As [Wang et al. \(2012\)](#) and [Wang et al. \(2007\)](#), we used a subset of 297 cell-cycled-regularized genes and the binding probabilities of a total of 96 TFs obtained from a mixture model approach of [Wang](#)

et al. (2007) based on the ChIP data of Lee et al. (2002). We applied PGEE to identify the TFs that are significantly associated with gene expression level at M/G1, G1, S, G2, and M stages (each stage has different number of time points from the cycle). We fitted the following PGEE to each stage with three different working correlation matrices (independence, exchangeable, and Ar(1)):

$$Y_{it} = \gamma_0 + \gamma_1 time_{it} + \sum_{p=1}^{96} \beta_p x_{ip}, \quad (10)$$

where $time_{it}$ is the time variable and x_{ip} 's are standardized transcription factor values ($p = 1, 2, \dots, 96$). When fitting PGEE, only β_p 's were subject to penalization. Table 1 summarizes the number of TFs which are identified as statistically significant for each stage under three different working correlation structures.

Correlation	M/G1	G1	S	G2	M
Independence	16	12	14	8	9
Exchangeable	15	13	12	8	7
Ar(1)	15	13	13	8	8

Table 1: Number of TFs selected at each stage in the yeast cell-cycle process with PGEE

For illustration, we used the yeast data from "G1" stage. Specifically, like the R package `gee`, the package `PGEE` requires the response and covariate columns to be ordered by `id` column and within `id` column according to `time` column. In our example, the yeast data was saved under the name of `yeastG1` object. The first column was `id` column, which identifies the genes. Then, while the continuous responses were placed under the column `y`, the `time` variable and the standardized values of 96 TFs were placed subsequently. Due to space limitation, we illustrated a portion of the `yeastG1` data as follows:

```
> # load data
> data(yeastG1)
> data = yeastG1
> # get the column names
> colnames(data)[1:9]
[1] "id" "y" "time" "ABF1" "ACE2" "ADR1" "ARG80" "ARG81" "AR080"
> # see some portion of yeast G1 data
> head(data,5)[1:9]
  id  y time      ABF1      ACE2      ADR1      ARG80      ARG81      AR080
1  1 0.88   3 -0.09702788  8.3839614 -0.1435702 -0.1482691 -0.09690053 -0.1073776
2  1 0.32   4 -0.09702788  8.3839614 -0.1435702 -0.1482691 -0.09690053 -0.1073776
3  1 1.09  12 -0.09702788  8.3839614 -0.1435702 -0.1482691 -0.09690053 -0.1073776
4  1 0.73  13 -0.09702788  8.3839614 -0.1435702 -0.1482691 -0.09690053 -0.1073776
5  2 0.66   3 -0.34618104 -0.1418099 -0.1397801 -0.1476834 -0.08486203 -0.1073536
```

Prior to model fitting with the function `PGEE`, we needed to compute the cross-validated value of tuning parameter over a grid via the function `CVfit`. This process requires a trial-error period, where one can start with a wide grid interval and then narrow it down. As described in Section T.3, we determined the main input arguments of the function `CVfit` as follows:

```
> library(PGEE)
> # define the input arguments
> formula <- "y ~.-id"
> family <- gaussian(link = "identity")
> lambda.vec <- seq(0.01,0.2,0.01)
> # find the optimum lambda
> cv <- CVfit(formula = formula, id = id, data = data, family = family, scale.fix = TRUE,
+ scale.value = 1, fold = 4, lambda.vec = lambda.vec, pindex = c(1,2), eps = 10^-6,
+ maxiter = 30, tol = 10^-6)
> # print the results
> print(cv)
Call:
CVfit(formula = formula, id = id, data = data, family = family,
      scale.fix = TRUE, scale.value = 1, fold = 4, lambda.vec = lambda.vec,
      pindex = c(1, 2), eps = 10^-6, maxiter = 30, tol = 10^-6)
```

4-fold CV results:

```

      lambda      Cv
[1,]  0.01 720.6857
[2,]  0.02 482.1046
[3,]  0.03 382.6932
[4,]  0.04 358.1970
[5,]  0.05 344.1034
[6,]  0.06 338.4713
[7,]  0.07 335.7137
[8,]  0.08 333.5432
[9,]  0.09 330.7405
[10,] 0.10 327.8395
[11,] 0.11 326.3243
[12,] 0.12 325.3068
[13,] 0.13 324.6835
[14,] 0.14 324.5388
[15,] 0.15 324.8667
[16,] 0.16 325.7849
[17,] 0.17 327.1245
[18,] 0.18 328.4365
[19,] 0.19 329.5468
[20,] 0.20 330.6265

```

Optimal tuning parameter:

```

Best lambda
      0.14
> # see the returned values by CVfit
> names(cv)
[1] "fold"      "lam.vect"  "cv.vect"  "lam.opt"  "cv.min"   "call"
> # get the optimum lambda
> cv$lam.opt
[1] 0.14

```

After selecting the tuning parameter λ via the function `CVfit`, we apply the PGEE function with the working correlation matrix type `corstr = "independence"` as follows:

```

> # fit the PGEE model
> myfit1 <- PGEE(formula = formula, id = id, data = data, na.action = NULL,
+ family = family, corstr = "independence", Mv = NULL,
+ beta_int = c(rep(0,dim(data)[2]-1)), R = NULL, scale.fix = TRUE,
+ scale.value = 1, lambda = cv$lam.opt, pindex = c(1,2), eps = 10^-6,
+ maxiter = 30, tol = 10^-6, silent = TRUE)

```

For comparison, we also fit the same model with `corstr = "exchangeable"` and `corstr = "AR-1"` (see Table 1). Here, we use 0 initial values for the regression coefficients for γ_0 , γ_1 , and β_p 's ($p = 1, 2, \dots, 96$) by assigning 0's for `beta_int`. Alternatively, the initial estimates could be obtained via fitting a `glm` while setting `beta_int = NULL`. We specify the vector of index for unpenalized parameters as `pindex = c(1,2)` since the first two regression coefficients γ_0 and γ_1 in Equation (10) are not subject to penalization. Furthermore, the returned values of `myfit1` object (in a similar way `summary(myfit1)` object) can be found out by the `names` function:

```

> # get the values returned by myfit object
> names(myfit1)
[1] "title"          "version"       "model"
[4] "call"           "terms"         "nobs"
[7] "iterations"     "coefficients"  "nas"
[10] "linear.predictors" "fitted.values" "residuals"
[13] "family"         "y"             "id"
[16] "max.id"         "working.correlation" "scale"
[19] "epsilon"        "lambda.value"  "robust.variance"
[22] "naive.variance" "xnames"        "error"

```

The returned objects by PGEE function are in similar format as those returned by `gee` function in R package. For example, while we could obtain whole model fitting results by `summary(myfit1)` object,

we could also obtain the summary of the model fitting results, i.e., estimate of regression coefficients, their corresponding naive and robust standard errors as well as z-values through coefficients method for `summary(myfit1)` object as follows:

```
> # see a portion of the results returned by coef(summary(myfit1))
> head(coef(summary(myfit1)),7)
              Estimate Naive S.E.      Naive z Robust S.E.  Robust z
(Intercept) 9.835775e-02 0.0603431824 1.629972904 4.334557e-02 2.2691533
time        9.774627e-03 0.0065644728 1.489019375 3.274155e-03 2.9853891
ABF1       -4.032513e-03 0.0095653833 -0.421573608 2.054339e-03 -1.9629245
ACE2       1.824265e-06 0.0002669801 0.006832963 1.634333e-06 1.1162135
ADR1       1.911308e-07 0.0001733864 0.001102340 7.611678e-07 0.2511020
ARG80      2.017436e-07 0.0001741572 0.001158399 8.684975e-07 0.2322903
ARG81      2.374483e-05 0.0007900111 0.030056320 2.296590e-05 1.0339165
```

Any regression estimate less than 10^{-3} in magnitude can be considered as equal to 0 (and thus not selected) in PGEE. In this sense, we obtained the variables whose regression estimates greater than 10^{-3} and their summary statistics as follows:

```
> # see the variables which have non-zero coefficients
> index1 <- which(abs(coef(summary(myfit1))["Estimate"]) > 10^-3)
> names(abs(coef(summary(myfit1))[index1,"Estimate"]))
[1] "(Intercept)" "time"      "ABF1"      "FKH1"      "FKH2"      "GAT3"
[7] "GCR2"        "MBP1"      "MSN4"      "NDD1"      "PHD1"      "RGM1"
[13] "RLM1"       "SMP1"      "SRD1"      "STB1"      "SWI4"      "SWI6"
```

> # see the PGEE summary statistics of these non-zero variables

```
> coef(summary(myfit1))[index1,]
              Estimate Naive S.E.      Naive z Robust S.E.  Robust z
(Intercept) 0.098357752 0.060343182 1.6299729 0.043345573 2.269153
time        0.009774627 0.006564473 1.4890194 0.003274155 2.985389
ABF1       -0.004032513 0.009565383 -0.4215736 0.002054339 -1.962925
FKH1       -0.009152898 0.013746477 -0.6658359 0.004173178 -2.193268
FKH2       -0.091503036 0.029629441 -3.0882471 0.017178993 -5.326449
GAT3       0.009780852 0.014721289 0.6644019 0.002192983 4.460067
GCR2       -0.005837966 0.011396288 -0.5122690 0.003227041 -1.809077
MBP1       0.102623543 0.028474614 3.6040363 0.017389748 5.901382
MSN4       0.011652400 0.015301265 0.7615318 0.004533629 2.570215
NDD1       -0.068098866 0.027962789 -2.4353389 0.017078278 -3.987455
PHD1       0.018224333 0.017586392 1.0362747 0.006676215 2.729740
RGM1       0.031474714 0.022152842 1.4207980 0.006025010 5.224010
RLM1       0.004245315 0.009823147 0.4321746 0.003155203 1.345497
SMP1       0.018181353 0.017691495 1.0276889 0.007614400 2.387759
SRD1       -0.009422532 0.013882871 -0.6787164 0.005117179 -1.841353
STB1       0.038198667 0.022075228 1.7303860 0.017485954 2.184534
SWI4       0.007370389 0.012622711 0.5838990 0.004184668 1.761284
SWI6       0.033957904 0.022673644 1.4976818 0.013225660 2.567577
```

For comparison, we fitted the unpenalized GEEs via MGEE function under the same settings defined above as follows:

```
> # fit the GEE model
> myfit2 <- MGEE(formula = formula, id = id, data = data, na.action = NULL,
+ family = family, corstr = "independence", Mv = NULL,
+ beta_int = c(rep(0,dim(data)[2]-1)), R = NULL, scale.fix = TRUE,
+ scale.value = 1, maxiter = 30, tol = 10^-6, silent = TRUE)
```

Finally, we obtain the TFs which were significantly associated with the gene expression levels at G1 stage via PGEE and GEE analyses, respectively.

```
> # see the significantly associated TFs in PGEE analysis
> names(which(abs(coef(summary(myfit1))[index1,"Robust z"]) > 1.96))
[1] "(Intercept)" "time"      "ABF1"      "FKH1"      "FKH2"      "GAT3"
[7] "MBP1"        "MSN4"      "NDD1"      "PHD1"      "RGM1"      "SMP1"
[13] "STB1"       "SWI6"
> # see the significantly associated TFs in GEE analysis
```



```
> names(which(abs(coef(summary(myfit2))[, "Robust z"]) > 1.96))
[1] "(Intercept)" "time" "ABF1" "ARG81" "ASH1" "CAD1"
[7] "GAT3" "GCN4" "GCR1" "GRF10.Pho2." "MBP1" "MET31"
[13] "MET4" "MTH1" "NDD1" "PDR1" "ROX1" "STB1"
[19] "STP1" "YAP5" "ZAP1"
```

When the results of PGEE and GEE analysis are compared, it is observed that while TFs such that *FKH1*, *FKH2*, *MSN4*, *PHD1*, *RGM1*, and *SMP1* are determined as significantly associated by PGEE analysis, these TFs are not detected by GEE analysis. The direct use of classical unpenalized GEE in high-dimensional longitudinal data analysis may lead to misleading results as the efficiency of PGEE over GEE has been showed in the simulation studies presented in Wang et al. (2012).

We repeat the steps above for each M/G1, G1, S, G2, and M stages under three different working correlation matrices (independence, exchangeable, and Ar(1)) and identify the number of TFs selected at each stage where the results are presented in Table 1. The results in Table 1 suggest that PGEE is robust to working correlation matrix specification and the selected TFs do not change significantly across working correlation matrix types within a stage. Furthermore, the analysis also reveals that different TFs may play different roles on gene expression levels in each cell-cycle process and there may be a small overlap in the selected TFs at different stages (e.g., only *FKH1*, *FKH2*, and *SMP1* are related to all stages of the yeast cell-cycle process).

Conclusion

In this paper, we present the R package **PGEE** which implements the PGEEs approach in Wang et al. (2012). The PGEE procedure performs simultaneous estimation and variable selection for longitudinal data analysis with high-dimensional covariates. We believe that this package is useful to practitioners in diverse fields where high-dimensional longitudinal data commonly arises such as genetics and large-scale health studies.

Acknowledgment

Lan Wang's research is supported by NSF grant NSF DMS-1512267. Gul Inan's visit to University of Minnesota is supported through a 2219-Fellowship by the Scientific and Technological Research Council of Turkey (TUBITAK). Authors would also like to thank the editor and the reviewer for their constructive comments which improved the quality of the paper.

Bibliography

- P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Annals of Applied Statistics*, 5(1):232–253, 2011. URL <https://doi.org/10.1214/10-AOAS388>. [p396]
- V. J. Carey. *gee: Generalized Estimation Equation Solver*, 2015. URL <https://CRAN.R-project.org/package=gee>. R package version 4.13-19. [p393]
- H. Cho and A. Qu. Model selection for correlated data with diverging number of parameters. *Statistica Sinica*, 23:901–927, 2013. URL <https://doi.org/10.5705/ss.2011.058>. [p394]
- J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001. URL <https://doi.org/10.1198/016214501753382273>. [p394, 395, 396]
- J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010. URL <https://doi.org/10.18637/jss.v033.i01>. [p396]
- A. Genz, F. Bretz, T. Miwa, X. Mi, and T. Hothorn. *mvtnorm: Multivariate Normal and t Distributions*, 2015. URL <https://CRAN.R-project.org/package=mvtnorm>. R package version 1.0-3. [p396]
- J. Goeman. L1 penalized estimation in the Cox proportional hazards model. *Biometrical Journal*, 52(1): 70–84, 2010. URL <https://doi.org/10.1002/bimj.200900028>. [p396]
- U. Halekoh, S. Højsgaard, and J. Yan. The R package *geepack* for generalized estimating equations. *Journal of Statistical Software*, 15(2):1–11, 2006. URL <https://doi.org/10.18637/jss.v015.i02>. [p393]

- J. W. Hardin and J. M. Hilbe. *Generalized Estimating Equations*. Wiley Online Library, 2003. [p394]
- N. J. Horton and S. R. Lipsitz. Review of software to fit generalized estimating equation regression models. *The American Statistician*, 53(2):160–169, 1999. URL <https://doi.org/10.1080/00031305.1999.10474451>. [p394]
- G. Inan, J. Zhou, and L. Wang. *PGEE: Penalized Generalized Estimating Equations in High-Dimension*, 2017. URL <https://CRAN.R-project.org/package=PGEE>. R package version 1.5. [p393]
- B. A. Johnson, D. Lin, and D. Zeng. Penalized estimating functions and variable selection in semiparametric regression models. *Journal of the American Statistical Association*, 103(482):672–680, 2008. URL <https://doi.org/10.1198/016214508000000184>. [p395]
- T. I. Lee, N. J. Rinaldi, F. Robert, D. T. Odom, Z. Bar-Joseph, G. K. Gerber, N. M. Hannett, C. T. Harbison, C. M. Thompson, I. Simon, et al. Transcriptional regulatory networks in *saccharomyces cerevisiae*. *Science*, 298(5594):799–804, 2002. URL <https://doi.org/10.1126/science.1075090>. [p398]
- H. Lian, H. Liang, and L. Wang. Generalized additive partial linear models for clustered data with diverging number of covariates using GEE. *Statistica Sinica*, 24:173–196, 2014. [p393]
- K.-Y. Liang and S. L. Zeger. Longitudinal data analysis using generalized linear models. *Biometrika*, 73(1):13–22, 1986. URL <https://doi.org/10.2307/2336267>. [p393, 394]
- B. Ripley. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2015. URL <https://CRAN.R-project.org/package=MASS>. R package version 7.3-45. [p396]
- B. Sherwood and A. Maidman. *rqPen: Penalized Quantile Regression*, 2016. URL <https://cran.r-project.org/web/packages/rqPen/>. R package version 1-4. [p396]
- P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of Cell*, 9(12):3273–3297, 1998. URL <https://doi.org/10.1091/mbc.9.12.3273>. [p397]
- L. Wang. GEE analysis of clustered binary data with diverging number of covariates. *The Annals of Statistics*, 39(1):389–417, 2011. URL <https://doi.org/10.1214/10-AOS846>. [p393]
- L. Wang, G. Chen, and H. Li. Group SCAD regression analysis for microarray time course gene expression data. *Bioinformatics*, 23(12):1486–1494, 2007. URL <https://doi.org/10.1093/bioinformatics/btm125>. [p394, 397]
- L. Wang, J. Zhou, and A. Qu. Penalized generalized estimating equations for high-dimensional longitudinal data analysis. *Biometrics*, 68(2):353–360, 2012. URL <https://doi.org/10.1111/j.1541-0420.2011.01678.x>. [p393, 394, 395, 396, 397, 401]
- L. Wang, L. Xue, A. Qu, and H. Liang. Estimation and model selection in generalized additive partial linear models for correlated data with diverging number of covariates. *The Annals of Statistics*, 42(2):592–624, 2014. URL <https://doi.org/10.1214/13-AOS1194>. [p393]
- S. L. Zeger and K.-Y. Liang. Longitudinal data analysis for discrete and continuous outcomes. *Biometrics*, 4(1):121–130, 1986. [p393]

Gul Inan
Department of Statistics
Middle East Technical University
Ankara, 06800
Turkey
ginan@metu.edu.tr

Lan Wang
School of Statistics
University of Minnesota
Minneapolis, MN 55455
USA
wangx346@umn.edu

BayesBinMix: an R Package for Model Based Clustering of Multivariate Binary Data

by Panagiotis Papastamoulis and Magnus Rattray

Abstract The **BayesBinMix** package offers a Bayesian framework for clustering binary data with or without missing values by fitting mixtures of multivariate Bernoulli distributions with an unknown number of components. It allows the joint estimation of the number of clusters and model parameters using Markov chain Monte Carlo sampling. Heated chains are run in parallel and accelerate the convergence to the target posterior distribution. Identifiability issues are addressed by implementing label switching algorithms. The package is demonstrated and benchmarked against the Expectation-Maximization algorithm using a simulation study as well as a real dataset.

Introduction

Clustering data is a fundamental task in a wide range of applications and finite mixture models are widely used for this purpose (McLachlan and Peel, 2000; Marin et al., 2005; Frühwirth-Schnatter, 2006). In this paper our attention is focused on clustering binary datasets. A variety of studies aims at identifying patterns in binary data including, but not limited to, voting data (Ilin, 2012), text classification (Juan and Vidal, 2002), handwritten digit recognition (Al-Ohali et al., 2003), medical research (Sun et al., 2007), animal classification Li (2005) and genetics (Abel et al., 1993).

Throughout this paper the term *cluster* is used as a synonym of *mixture component*. Finite mixture models can be estimated under a frequentist approach using the Expectation-Maximization (EM) algorithm (Dempster et al., 1977). However, the likelihood surface of a mixture model can exhibit many local maxima and it is well known that the EM algorithm may fail to converge to the main mode if it is initialized from a point close to a minor mode. Moreover, under a frequentist approach, the selection of the number of clusters is not straightforward: a mixture model for each possible value of number of clusters is fitted and then the optimal one is selected according to penalized likelihood criteria such as the Bayesian information criterion (Schwarz, 1978) or the integrated complete likelihood criterion (Biernacki et al., 2000). The reader is also referred to Lindsay (1995); Böhning (2000) for non-parametric likelihood estimation of a mixture model.

On the other hand, the Bayesian framework allows to put a prior distribution on both the number of clusters as well as the model parameters and then (approximately) sample from the joint posterior distribution using Markov chain Monte Carlo (MCMC) algorithms (Richardson and Green, 1997; Stephens, 2000a; Nobile and Fearnside, 2007; White et al., 2016). However this does not mean that the Bayesian approach is not problematic. In general, vanilla MCMC algorithms may require a very large number of iterations to discover the high posterior density areas and/or sufficiently explore the posterior surface due to the existence of minor modes. Second, identifiability issues arise due to the label switching phenomenon (Redner and Walker, 1984) which complicate the inference procedure.

The **BayesBinMix** package explicitly takes care of the previously mentioned problems for the problem of clustering multivariate binary data:

1. Allows missing values in the observed data,
2. Performs MCMC sampling for estimating the posterior distribution of the number of clusters and model parameters,
3. Produces a rapidly mixing MCMC sample by running parallel heated chains which can switch states,
4. Post-processes the generated MCMC sample and produces meaningful posterior mean estimates using state of the art algorithms to deal with label switching.

The rest of the paper is organised as follows. The mixture model is presented in Section V.2. Its prior assumptions and the corresponding hierarchical model is introduced in Section V.3. The basic MCMC scheme is detailed in Section V.4.1. Section V.4.2 deals with post-processing the generated MCMC sample in order to overcome identifiability issues due to the label switching problem. Finally, the basic sampler is embedded in a Metropolis-coupled MCMC algorithm as described in Section V.4.3. The main function of the package is described in Section V.5. Simulated and real datasets are analyzed in Sections V.6.1 and V.6.2, respectively.

Model

Let $\mathbf{x} = (x_1, \dots, x_n)$ denote a random sample of multivariate binary data, where $x_i = (x_{i1}, \dots, x_{id})$; $d > 1$, for $i = 1, \dots, n$. Assume that the observed data has been generated from a mixture of independent Bernoulli distributions, that is,

$$\begin{aligned} x_i &\sim \sum_{k=1}^K p_k \prod_{j=1}^d f(x_{ij}; \theta_{kj}) \\ &= \sum_{k=1}^K p_k \prod_{j=1}^d \theta_{kj}^{x_{ij}} (1 - \theta_{kj})^{1-x_{ij}} \mathbb{I}_{\{0,1\}}(x_{ij}), \end{aligned} \tag{1}$$

independently for $i = 1, \dots, n$, where $\theta_{kj} \in \Theta = (0, 1)$ denotes the probability of success for the k -th cluster and j -th response for $k = 1, \dots, K$; $j = 1, \dots, d$, $\mathbf{p} = (p_1, \dots, p_K) \in \mathcal{P}_{K-1} = \{p_k; k = 1, \dots, K - 1 : 0 \leq p_k \leq 1; 0 \leq p_K = 1 - \sum_{k=1}^{K-1} p_k\}$ corresponds to the vector of mixture weights and $\mathbb{I}_A(\cdot)$ denotes the indicator function of a (measurable) subset A .

It is straightforward to prove that the variance-covariance matrix of a mixture of independent Bernoulli distributions is not diagonal (see e.g. Bishop (2006)), which is the case for a collection of independent Bernoulli distributions. Therefore, the mixture model exhibits richer covariance structure thus it can prove useful to discover correlations in heterogeneous multivariate binary data.

The observed likelihood of the model is written as

$$L_K(\mathbf{p}, \boldsymbol{\theta}; \mathbf{x}) = \prod_{i=1}^n \sum_{k=1}^K p_k \prod_{j=1}^d \theta_{kj}^{x_{ij}} (1 - \theta_{kj})^{1-x_{ij}}, \quad (\mathbf{p}, \boldsymbol{\theta}) \in \mathcal{P}_{K-1} \times \Theta^{Kd} \tag{2}$$

where $\mathbf{x} \in \mathcal{X}^n = \{0, 1\}^{nd}$. For any fixed value of K , Equation (2) can be further decomposed by considering that observation i has been generated from the z_i -th mixture component, that is,

$$x_i | z_i = k \sim \prod_{j=1}^d f(x_{ij}; \theta_{kj}), \quad \text{independent for } i = 1, \dots, n. \tag{3}$$

Note that the allocation variables $z_i \in \mathcal{Z}_K = \{1, \dots, K\}$; $i = 1, \dots, n$ are unobserved, so they are treated as missing data. Assume that

$$P(z_i = k | \mathbf{p}, K) = p_k, \quad k = 1, \dots, K \tag{4}$$

and furthermore that (x_i, z_i) are independent for $i = 1, \dots, n$. Data augmentation (Tanner and Wong, 1987) considers jointly the complete data $\{(x_i, z_i); i = 1, \dots, n\}$ and it is a standard technique exploited both by the Expectation-Maximization algorithm (Dempster et al., 1977) as well as the Gibbs sampler (Gelfand and Smith, 1990). The complete likelihood is defined as

$$\begin{aligned} L_K^c(\mathbf{p}, \boldsymbol{\theta}; \mathbf{x}, \mathbf{z}) &= \prod_{i=1}^n p_{z_i} \prod_{j=1}^d \theta_{z_i j}^{x_{ij}} (1 - \theta_{z_i j})^{1-x_{ij}} \\ &= \prod_{k=1}^K p_k^{n_k} \prod_{j=1}^d \theta_{kj}^{s_{kj}} (1 - \theta_{kj})^{n_k - s_{kj}}, \quad (\mathbf{p}, \boldsymbol{\theta}) \in \mathcal{P}_{K-1} \times \Theta^{Kd} \end{aligned} \tag{5}$$

where $n_k = \sum_{i=1}^n \mathbb{I}(z_i = k)$ and $s_{kj} = \sum_{i=1}^n \mathbb{I}(z_i = k) x_{ij}$, $k = 1, \dots, K$; $j = 1, \dots, d$, for a given $(\mathbf{x}, \mathbf{z}) \in \mathcal{X}^n \times \mathcal{Z}_K^n$.

Prior assumptions

Note that the quantities $\mathbf{p}, \boldsymbol{\theta}, \mathbf{z}$ are defined conditionally on K . For convenience we will assume that $K \in \mathcal{K} = \{1, \dots, K_{\max}\}$, where K_{\max} denotes an upper bound on the number of clusters. Hence, under a model-based clustering point of view, the vector

$$(K, \mathbf{p}, \boldsymbol{\theta}, \mathbf{z}) \in \mathcal{A} := \mathcal{K} \times \mathcal{P}_{K-1} \times \Theta^{Kd} \times \mathcal{Z}_K^n$$

summarizes all unknown parameters that we wish to infer.

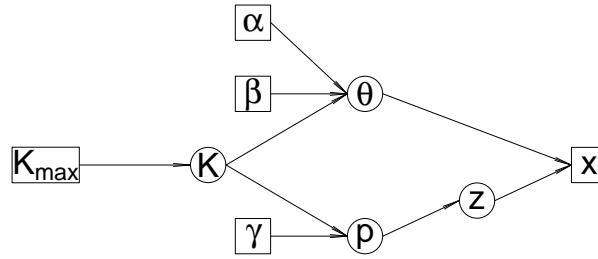


Figure 1: Representation of the hierarchical model (9) as a directed acyclic graph. Squares and circles denote observed/fixed and unknown variables, respectively.

The following prior assumptions are imposed

$$\begin{aligned}
 K &\sim \text{Discrete}\{1, \dots, K_{\max}\} & (6) \\
 \mathbf{p}|K &\sim \text{Dirichlet}(\gamma_1, \dots, \gamma_K) & (7) \\
 \theta_{kj}|K &\sim \text{Beta}(\alpha, \beta), & (8)
 \end{aligned}$$

independent for $k = 1, \dots, K; j = 1, \dots, d$. The discrete distribution in Equation (6) can be either a Uniform or a Poisson distribution with mean $\lambda = 1$ truncated on the set $\{1, \dots, K_{\max}\}$. Equations (7) and (8) correspond to typical prior distributions for the mixture weights and success probabilities, that furthermore enjoy conjugacy properties. Typically, we set $\gamma_1 = \dots = \gamma_K = \gamma > 0$ so that the prior assumptions do not impose any particular information that separates the mixture components between them, which is also a recommended practice in mixture modelling.

According to Equations (4), (5), (6), (7) and (8), the joint probability density function of the model is

$$f(x, K, z, \mathbf{p}, \boldsymbol{\theta}) = f(x|K, z, \mathbf{p}, \boldsymbol{\theta}) f(z|K, \mathbf{p}) f(\mathbf{p}|K) f(\boldsymbol{\theta}|K) f(K), \tag{9}$$

and its graphical representation is shown in Figure 1.

Inference

Allocation sampler

Let $C_K = \frac{\Gamma(\sum_{k=1}^K \gamma_k)}{\prod_{k=1}^K \Gamma(\gamma_k)} \left\{ \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \right\}^{Kd}$. From Equation (9) the joint posterior distribution of $(K, \mathbf{p}, \boldsymbol{\theta}, z)$ can be expressed as:

$$f(\boldsymbol{\theta}, z, \mathbf{p}, K|x) \propto C_K f(K) \prod_{k=1}^K \left\{ p_k^{n_k + \gamma_k - 1} \prod_{j=1}^d \theta_{kj}^{\alpha + s_{kj} - 1} (1 - \theta_{kj})^{\beta + n_k - s_{kj} - 1} \right\} \mathbb{I}_{\mathcal{A}}(K, \mathbf{p}, \boldsymbol{\theta}, z). \tag{10}$$

From the last expression it is straightforward to derive the full conditional distributions of the component specific parameters and latent allocation variables as follows:

$$\mathbf{p}|K, z \sim \text{Dirichlet}(\gamma_1 + n_1, \dots, \gamma_K + n_K) \tag{11}$$

$$\theta_{kj}|K, z, \mathbf{x} \sim \text{Beta}(\alpha + s_{kj}, \beta + n_k - s_{kj}) \tag{12}$$

$$P(z_i = k|K, \mathbf{x}_i, \mathbf{p}, \boldsymbol{\theta}) \propto p_k \prod_{j=1}^d \theta_{kj}^{x_{ij}} (1 - \theta_{kj})^{1 - x_{ij}}, \quad k = 1, \dots, K,$$

independent for $i = 1, \dots, n; k = 1, \dots, K; j = 1, \dots, d$.

A general framework for updating the number of mixture components (K) is given by trans-dimensional MCMC approaches, such as the reversible jump MCMC (Green, 1995; Richardson and Green, 1997; Papastamoulis and Iliopoulos, 2009) or the Birth-Death MCMC (Stephens, 2000a) methodologies. However, the conjugate prior assumptions used for the component specific parameters $(\mathbf{p}, \boldsymbol{\theta})$ allow us to use simpler techniques by integrating those parameters out from the model and perform collapsed sampling (Liu, 1994) on the space of (K, z) . We use the allocation sampler (Nobile and Fearnside, 2007) which introduced this sampling scheme for parametric families such that conjugate prior distributions exist and also applied it in the specific context of mixtures of normal distributions. This approach was recently followed by White et al. (2016) which also allowed for variable selection.

Let $\mathcal{A}_0 = \Theta^{Kd} \times \mathcal{P}_{K-1}$ and $\mathcal{A}_1 = \mathcal{K} \times \{1, \dots, K\}^n$. Integrating out (θ, \mathbf{p}) from (10) we obtain

$$\begin{aligned}
 f(K, \mathbf{z} | \mathbf{x}) &= \int_{\mathcal{A}_0} f(\mathbf{z}, \theta, \mathbf{p}, K | \mathbf{x}) d\theta d\mathbf{p} \\
 &\propto C_K f(K) \mathbb{I}_{\mathcal{A}_1}(K, \mathbf{z}) \int_{\mathcal{A}_0} \prod_{k=1}^K p_k^{n_k + \gamma_k - 1} \prod_{j=1}^d \theta_{kj}^{\alpha + s_{kj} - 1} (1 - \theta_{kj})^{\beta + n_k - s_{kj} - 1} d\theta d\mathbf{p} \\
 &\propto C_K f(K) \frac{\prod_{k=1}^K \Gamma(n_k + \gamma_k)}{\Gamma(n + \sum_{k=1}^K \gamma_k)} \prod_{k=1}^K \prod_{j=1}^d \frac{\Gamma(\alpha + s_{kj}) \Gamma(\beta + n_k - s_{kj})}{\Gamma(\alpha + \beta + n_k)} \mathbb{I}_{\mathcal{A}_1}(K, \mathbf{z}). \tag{13}
 \end{aligned}$$

Let now $\mathbf{z}_{[-i]} = \{z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_n\}$ and also define the following quantities, for $i = 1, \dots, n$:

$$\begin{aligned}
 n_k^{[i]} &= \sum_{h \neq i} \mathbb{I}(z_h = k), k = 1, \dots, K \\
 s_{kj}^{[i]} &= \sum_{h \neq i} \mathbb{I}(z_h = k) x_{hj}, k = 1, \dots, K; j = 1, \dots, d \\
 A_1^{[i]} &= \{j = 1, \dots, d : x_{ij} = 1\} \\
 A_0^{[i]} &= \{j = 1, \dots, d : x_{ij} = 0\}.
 \end{aligned}$$

From Equation (13), the (collapsed) conditional posterior distribution of z_i is

$$P(z_i = k | \mathbf{z}_{[-i]}, K, \mathbf{x}) \propto \frac{n_k^{[i]} + \gamma_k}{(\alpha + \beta + n_k^{[i]})^d} \prod_{j \in A_1^{[i]}} (\alpha + s_{kj}^{[i]}) \prod_{j \in A_0^{[i]}} (\beta + n_k - s_{kj}^{[i]}), \tag{14}$$

$k = 1, \dots, K; i = 1, \dots, n$.

It is well known that draws from the conditional distributions in Equation (14) exhibit strong serial correlation, slowing down the convergence of the MCMC sampler. The mixing can be improved by proposing simultaneous updates of blocks of $\mathbf{z} | K$, by incorporating proper Metropolis-Hastings moves on $\mathbf{z} | K$. Following Nobile and Fearnside (2007), we also propose jumps to configurations that massively update the allocation vector as follows:

1. **Move 1:** select two mixture components and propose a random reallocation of the assigned observations.
2. **Move 2:** select two mixture components and propose to move a randomly selected subset of observations from the 1st to the 2nd one.
3. **Move 3:** select two mixture components and propose a reallocation of the assigned observations according to the full conditional probabilities given the already processed ones.

Each move is accepted according to the corresponding Metropolis-Hastings acceptance probability, see Nobile and Fearnside (2007) for details.

The final step of the allocation sampler is to update the number of clusters (K). According to Nobile and Fearnside (2007), this is achieved by performing a Metropolis-Hastings type move, namely a pair of absorption/ejection moves which decrease/increase K , respectively. Assume that the current state of chain is $\{K, \mathbf{z}\}$. The following pseudocode describes the Absorption/Ejection step:

1. Attempt ejection with probability p_K^e , where $p_K^e = 1/2, K = 2, \dots, K_{\max} - 1, p_1^e = 1$ and $p_{K_{\max}}^e = 0$. Otherwise, an absorption move is attempted.
2. Suppose that an ejection is attempted. The candidate state is $\{K', \mathbf{z}'\}$ with $K' = K + 1$.
 - (a) Propose reallocation of observations assigned to the ejecting component between itself and the ejected component according to the Beta($\tilde{\alpha}, \tilde{\alpha}$) distribution.
 - (b) Accept the candidate state with probability $\min\{1, R\}$ where

$$R = R(\tilde{\alpha}) = \frac{f(K', \mathbf{z}' | \mathbf{x}) P(\{K', \mathbf{z}'\} \rightarrow \{K, \mathbf{z}\})}{f(K, \mathbf{z} | \mathbf{x}) P(\{K, \mathbf{z}\} \rightarrow \{K', \mathbf{z}'\})} \tag{15}$$

3. If an absorption is attempted:
 - (a) all observations allocated to the absorbed component are reallocated to the absorbing component.

- (b) the candidate state is accepted with probability $\min\{1, 1/R(\tilde{\alpha})\}$.

The parameter $\tilde{\alpha}$ is chosen in a way that ensures that the probability of ejecting an empty component is sufficiently large. For full details the reader is referred to [Nobile and Fearnside \(2007\)](#).

The allocation sampler for mixtures of multivariate Bernoulli distributions is summarized in the following algorithm.

Algorithm 1 (Allocation sampler for Bernoulli mixtures) Given an initial state $\{K^{(0)}, \mathbf{z}^{(0)}\} \in \mathcal{A}_1$ iterate the following steps for $t = 1, 2, \dots$

1. For $i = 1, \dots, n$

(a) Compute $n_k^{[i]} = \sum_{h \neq i} \mathbb{I}(z_h = k)$, $s_{kj}^{[i]} = \sum_{h \neq i} \mathbb{I}(z_h = k) x_{hj}$, $k = 1, \dots, K^{(t)}$; $j = 1, \dots, d$.

(b) Update $z_i^{(t)} | \mathbf{z}_{[-i]}^{(t)}$ according to Equation (14).

2. Propose Metropolis-Hastings moves M_1, M_2 and M_3 to update $\mathbf{z}^{(t)}$.
 3. Propose an Absorption/Ejection move to update $\{K^{(t)}, \mathbf{z}^{(t)}\}$.

Note in step 1.(a):

$$z_h = \begin{cases} z_h^{(t)}, & h < i \\ z_h^{(t-1)}, & h > i. \end{cases}$$

Finally, we mention that after the last step of Algorithm 1 we can also simulate the component-specific parameters \mathbf{p} and $\boldsymbol{\theta}$ from their full conditional posterior distributions given in (11) and (12), respectively. Although this is not demanded in case that the user is only interested in inferring $K, \mathbf{z} | \mathbf{x}$, it will produce an (approximate) MCMC sample from the full posterior distribution of $K, \mathbf{p}, \boldsymbol{\theta}, \mathbf{z} | \mathbf{x}$. If the observed data contains missing entries an extra step is implemented in order to simulate the corresponding values. For this purpose we use the full conditional distribution derived from Equation (3), taking only into account the subset of $\{1, \dots, d\}$ that contains missing values for a given $i = 1, \dots, n$.

Label switching issue and identifiability

Label switching ([Redner and Walker, 1984](#)) is a well known identifiability problem occurring in MCMC outputs of mixture models, arising from the symmetry of the likelihood with respect to permutations of components' labels. A set of sufficient conditions under a general framework of missing data models that lead to label switching and its consequences is given in [Papastamoulis and Iliopoulos \(2013\)](#). If an MCMC sample exhibits label switching, the standard practice of estimating the posterior means and other parametric functions by ergodic averages becomes meaningless. In order to deal with this identifiability problem we have considered two versions of ECR algorithm ([Papastamoulis and Iliopoulos, 2010; Papastamoulis, 2014; Rodríguez and Walker, 2014](#)) as well as the KL algorithm ([Stephens, 2000b](#)). These algorithms are quite efficient and in most cases exhibit almost identical results, but ECR is significantly faster and computationally lightweight compared to KL. The implementation was performed in the R package `label.switching` ([Papastamoulis, 2016](#)).

Note here that in the case that $d = 1$, Equation (1) collapses to a single Bernoulli distribution. Hence, there are two types of identifiability issues in mixture models: the first one is related to the fact that the model is identifiable only up to a permutation of the parameters (label switching). The second one is strict non-identifiability which relates to the fact that for a mixture of discrete distributions (such as the multivariate Bernoulli) totally different parameter values can correspond to the same distribution. We are not dealing with this second source of identifiability problems since it has been empirically demonstrated that estimation can still produce meaningful results in practice ([Carreira-Perpiñán and Renals, 2000](#)). In addition, [Allman et al. \(2009\)](#) showed that finite mixtures of Bernoulli products are in fact generically identifiable despite their lack of strict identifiability.

Metropolis-coupled MCMC sampler

There are various strategies for improving MCMC sampling, see e.g. chapter 6 in [Gilks et al. \(1996\)](#). In this study, the Metropolis-coupled MCMC (MC^3) ([Geyer, 1991; Geyer and Thompson, 1995; Altekar et al., 2004](#)) strategy is adopted. An MC^3 sampler runs m chains with different posterior distributions $f_i(\xi); i = 1, \dots, m$. The target posterior distribution corresponds to $i = 1$, that is, $f_1(\xi) = f(\xi)$, while the rest of them are chosen in a way that the mixing is improved. This is typically achieved by considering *heated* versions of the original target, that is, $f_i(\xi) = f(\xi)^{h_i}$ where $h_1 = 1$ and $0 < h_i < 1$

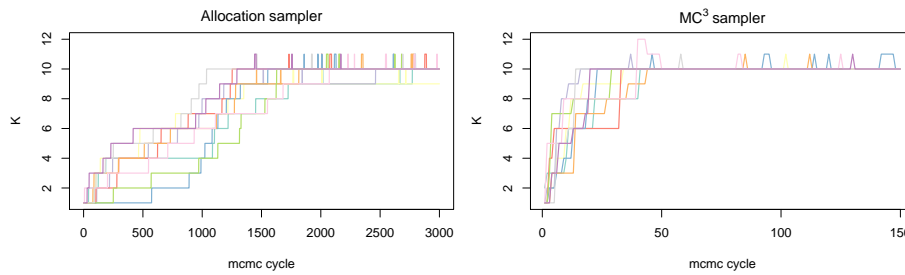


Figure 2: Trace of sampled values of the number of clusters (K) for 10 different runs, using one of the synthetic datasets of Section V.6.1 with $K_{\text{true}} = 10$. Each run was initialized from $K = 1$ and every 10th iteration is displayed (labeled as MCMC cycle on the x axis) until most chains start to explore $K = 10$. Left: standard allocation sampler, right: MC^3 sampler (using 4 heated chains).

for $i = 2, \dots, m$ represents the heat value of the chain. Note that when raising the posterior distribution to a power $0 < h_i < 1$ makes the modified posterior surface flatter, thus, easier to explore compared to $f(\xi)$. Only the chain that corresponds to the target posterior distribution is used for the posterior inference, however, after each iteration a proposal attempts to swap the states of two randomly chosen chains. This improves the mixing of the chain since it is possible that an accepted swap between the cold and a heated chain will make the former move to another mode.

Let $\xi_i^{(t)}$ denote the state of chain i at iteration t and that a swap between chains i and j is proposed. Note that in our setup $\xi = (K, z)$ and f is given in (13) (up to a normalizing constant). The proposed move is accepted with probability

$$\min \left\{ 1, \frac{f_i(\xi_j^{(t)}) f_j(\xi_i^{(t)})}{f_i(\xi_i^{(t)}) f_j(\xi_j^{(t)})} \right\} = \min \left\{ 1, \frac{f(\xi_j^{(t)})^{h_i} f(\xi_i^{(t)})^{h_j}}{f(\xi_i^{(t)})^{h_i} f(\xi_j^{(t)})^{h_j}} \right\}.$$

Figure 2 sketches the difference in convergence speed between the standard allocation sampler and an MC^3 sampler which are used to infer the same posterior distribution of the number of clusters. Although a single MCMC cycle of MC^3 is more expensive than a cycle of the allocation sampler, it is evident that the MC^3 sampler can recover the true number of clusters ($K_{\text{true}} = 10$) in a remarkably smaller number of iterations than the standard allocation sampler. In addition, the standard allocation sampler rarely switches between the symmetric modes of the posterior distribution, a fact which typically indicates poor mixing of MCMC samplers in mixture models (Marin et al., 2005). On the contrary, the MC^3 sampler produces a chain where label switching occurs in a rate proportional to the swap acceptance rate.

In order to take full advantage of computing power in modern-day computers, our MC^3 sampler utilizes parallel computing in multiple cores. This is achieved by running each chain in parallel using the R packages `foreach` (Revolution Analytics and Weston, 2014) and `doParallel` (Revolution Analytics and Weston, 2015). Every 10-th iteration a swap is proposed between a pair of chains.

Using package `BayesBinMix`

The main function of the `BayesBinMix` package is `coupledMetropolis`, with its arguments shown in Table 1. This function takes as input a binary data array (possibly containing missing values) and runs the allocation sampler for a series of heated chains which run in parallel while swaps between pairs of chains are proposed. In the case that the most probable number of mixture components is larger than 1, the label switching algorithms are applied.

As the function runs it prints some basic information on the screen such as the progress of the sampler as well as the acceptance rate of proposed swaps between chains. The output which is returned to the user mainly consists of "mcmc" objects, a class imported from the `coda` package (Plummer et al., 2006). More specifically, the `coupledMetropolis()` function returns the objects detailed in Table 2. We note that this is just a subset of the full output of the sampler which consists of several additional quantities, such as the raw MCMC values corresponding to the whole set of generated values of K . Usually this information is not necessary to the average user, thus, it is saved to a separate set of files in the folder specified by `outPrefix`.

Argument	Description
Kmax	Maximum number of clusters (integer, at least equal to two).
nChains	Number of parallel (heated) chains.
heats	nChains-dimensional vector specifying the temperature of each chain: the 1st entry should always be equal to 1 and the rest of them lie on the set:(0, 1].
binaryData	The observed binary data (array). Missing values are allowed as long as the corresponding entries are denoted as NA.
outPrefix	The name of the produced output folder. An error is thrown if the directory exists.
ClusterPrior	Character string specifying the prior distribution of the number of clusters. Available options: poisson or uniform. It defaults to the (truncated) Poisson distribution.
m	The number of MCMC cycles. At the end of each cycle a swap between a pair of heated chains is attempted. Each cycle consists of 10 iterations.
alpha	First shape parameter of the Beta prior distribution (strictly positive). Defaults to 1.
beta	Second shape parameter of the Beta prior distribution (strictly positive). Defaults to 1.
gamma	Kmax-dimensional vector (positive) corresponding to the parameters of the Dirichlet prior of the mixture weights. Default value: rep(1, Kmax).
z.true	An optional vector of cluster assignments considered as the ground-truth clustering of the observations. It is only used to obtain a final permutation of the labels (after the label switching algorithms) in order to maximise the similarity between the resulting estimates and the real cluster assignments. Useful for simulations.
ejectionAlpha	Probability of ejecting an empty component. Defaults to 0.2.
burn	Optional integer denoting the number of MCMC cycles that will be discarded as burn-in period.

Table 1: Arguments of the `coupledMetropolis()` function.

Examples

In this section the usage of **BayesBinMix** package is described and various benchmarks are presented. At first we demonstrate a typical implementation on a single simulated dataset and inspect the simulated parameter values and estimates. Then we perform an extensive study on the number of estimated clusters and compare our findings to the **FlexMix** package (Leisch, 2004; Grün and Leisch, 2007, 2008). An application to a real dataset is provided next.

Simulation study

At first, a single simulated dataset is used in order to give a brief overview of the implementation. We simulated $n = 200$ observations from the multivariate Bernoulli mixture model (1). The true number of clusters is set to $K = 6$ and the dimensionality of the multivariate distribution is equal to $d = 100$. The mixture weights are drawn from a Dirichlet $\mathcal{D}(1, 1, 1, 1, 1, 1)$ distribution resulting in (50, 46, 30, 36, 12, 26) generated observations from each cluster. For each cluster, true values for the probability of success were generated from a Uniform distribution, that is, $\theta_{kj} \sim \mathcal{U}(0, 1)$, independently for $k = 1, \dots, K; j = 1, \dots, d$. Furthermore, we introduce some missing values to the generated data: each row is allowed to contain missing values with probability 0.2: for such a row the total number of missing entries is drawn from the binomial distribution $B(100, 0.3)$. Finally, the observed data is saved to the 200×100 array `x` which contains a total of 1038 missing values corresponding to 34 rows.

We will run 4 parallel chains with the following temperatures: (1, 0.8, 0.6, 0.4). Observe that the first chain should correspond to the actual posterior distribution, so its temperature equals to 1. Now apply the `coupledMetropolis()` function as follows.

```
> library('BayesBinMix')
> nChains <- 4
> heats <- seq(1, 0.4, length = nChains)

# using the truncated Poisson prior distribution on the number of clusters
> cm1 <- coupledMetropolis(Kmax = 20, nChains = nChains, heats = heats,
  binaryData = x, outPrefix = "bbm-poisson", ClusterPrior = "poisson",
```

Object	Description
K.mcmc	Object of class "mcmc" (see <code>coda</code> package) containing the simulated values (after burn-in) of the number of clusters for the cold chain.
parameters.ecr.mcmc	Object of class "mcmc" containing the simulated values (after burn-in) of θ_{kj} (probability of success per cluster k and feature j) and π_k (weight of cluster k) for $k = 1, \dots, K_{\text{map}}$; $j = 1, \dots, d$, where K_{map} denotes the most probable number of clusters. The output is reordered according to ECR algorithm.
allocations.ecr.mcmc	Object of class "mcmc" containing the simulated values (after burn-in) of z_i (allocation variables) for $i = 1, \dots, n$, given $K = K_{\text{map}}$. The output is reordered according to ECR algorithm.
classificationProbabilities.ecr	Data frame of the reordered classification probabilities per observation after reordering the most probable number of clusters with the ECR algorithm.
clusterMembershipPerMethod	Data frame of the most probable allocation of each observation after reordering the MCMC sample which corresponds to the most probable number of clusters according to ECR, STEPHENS and ECR-ITERATIVE-1 methods.
K.allChains	$m \times n$ Chains matrix containing the simulated values of the number of clusters (K) per chain.
chainInfo	Number of parallel chains, cycles, burn-in period and acceptance rate of swap moves.

Table 2: Output returned to the user of the `coupledMetropolis()` function.

```

m = 1100, z.true = z.true, burn = 100)

# using the uniform prior distribution on the number of clusters
> cm2 <- coupledMetropolis(Kmax = 20, nChains = nChains, heats = heats,
  binaryData = x, outPrefix = "bbm-uniform", ClusterPrior = "uniform",
  m = 1100, z.true = z.true, burn = 100)

```

Note that we have called the function twice using either the truncated Poisson or the Uniform prior on the set $\{1, \dots, 20\}$. The total number of MCMC cycles corresponds to $m = 1100$ and the first 100 cycles will be discarded as burn-in period. Recall that each cycle contains 10 usual MCMC iterations, so this is equivalent to keeping every 10th iteration of a chain with 11,000 iterations. Since we are interested to compare against the true values used to generate the data, we also supply `z.true` which contains the true allocation of each observation. It is only used for making the inferred clusters agree to the labelling of the true values and it has no impact on the MCMC or label switching algorithms.

Printing, summarizing and plotting the output

In this section we illustrate summaries of the basic output returned to the user, using only the run which corresponds to the Poisson prior distribution (`cm1`). The `print()` method of the package returns a basic summary of the fitted model:

```

> print(cm1)

* Run information:
  Number of parallel heated chains: 4
  Swap acceptance rate: 63.5%
  Total number of iterations: 11000
  Burn-in period: 1000
  Thinning: 10.

* Estimated posterior distribution of the number of clusters:

  6      7      8
0.971 0.026 0.003

* Most probable model: K = 6 with P(K = 6|data) = 0.971

```

* Estimated number of observations per cluster conditionally on $K = 6$ (3 label switching algorithms):

```
STEPHENS ECR ECR.ITERATIVE.1
1      50  50      50
2      46  46      46
3      30  30      30
4      36  36      36
5      12  12      12
6      26  26      26
```

* Posterior mean of probability of success per feature and cluster (ECR algorithm):

```
cluster_1 cluster_2 cluster_3 cluster_4 cluster_5 cluster_6
theta_1 0.33364058 0.8465393 0.7023264 0.3340989 0.08364937 0.8933767
theta_2 0.71919239 0.6653526 0.3227822 0.3982836 0.22369486 0.5936094
theta_3 0.49869339 0.2285653 0.3605507 0.3570447 0.07206039 0.1883581
theta_4 0.22360156 0.9148123 0.3359406 0.7889224 0.15476900 0.5924109
theta_5 0.01867034 0.8296381 0.8107050 0.1121773 0.78051586 0.1442368
<+ 95 more rows>
```

Next we present summaries of the marginal posterior distributions of the (reordered) MCMC sample of parameters conditionally on the selected number of clusters. The reordered MCMC sample of θ_{kj} and p_k (after burn-in) is returned to the "mcmc" object `parameters.ecr.mcmc`. Hence we can use the `summary()` method of the `coda` package, which prints empirical means, standard deviations, as well the quantiles for each variable. This is done with the following command.

```
> summary(cm1$parameters.ecr.mcmc)
```

1. Empirical mean and standard deviation for each variable, plus standard error of the mean:

```
      Mean      SD Naive SE Time-series SE
theta.1.1 0.33364 0.06504 0.0020874      0.0020874
...
theta.6.1 0.89338 0.05552 0.0017816      0.0017816
<+ 99 blocks of 6 rows>
p.1      0.24663 0.02869 0.0009208      0.0009208
...
p.6      0.13270 0.02276 0.0007304      0.0007304
```

2. Quantiles for each variable:

```
      2.5%      25%      50%      75%      97.5%
theta.1.1 0.2115064 0.289971 0.32896 0.37502 0.46542
...
theta.6.1 0.7676282 0.859599 0.90175 0.93405 0.97556
<+ 99 blocks of 6 rows>
p.1      0.1950401 0.225938 0.24436 0.26611 0.31012
...
p.6      0.0905194 0.117203 0.13206 0.14795 0.17993
```

The user can also visualize the output with a trace of the sampled values and a density estimate for each variable in the chain using the `plot()` method of the `coda` package. For illustration, the following example plots the trace and histogram for θ_{kj} and p_k for cluster $k = 2$ and feature $j = 1$. The produced plot is shown in Figure 3.

```
mat <- matrix(c(1:4), byrow = TRUE, ncol = 2)
layout(mat, widths = rep(c(2, 1), 2), heights = rep(1, 4))
mcmcSubset <- cm1$parameters.ecr.mcmc[, c("theta.2.1", "p.2")]
plot(mcmcSubset, auto.layout = FALSE, ask = FALSE, col = "gray40")
```

The reader is also referred to the `coda` package which provides various other functions for calculating and plotting MCMC diagnostics.

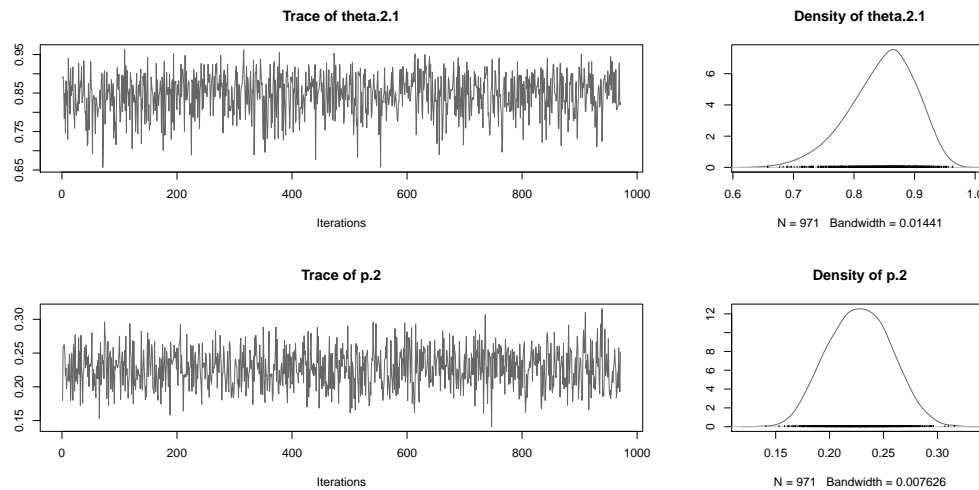


Figure 3: MCMC trace and density estimate for the reordered values of θ_{kj} and p_k for cluster $k = 2$ and feature $j = 1$, conditionally on the selected number of clusters ($K = 6$).

Further inspection of the output

Figures 4.(a) and 4.(b) illustrate the sampled values of K per chain according to the Poisson and uniform prior distribution, respectively. This information is returned to the user as an $m \times n$ Chains array named `K.allChains`. The actual posterior distribution corresponds to the blue line. Note that as the temperature increases the posterior distribution of K has larger variability. In both cases, the most probable state corresponds to $K = 6$ clusters, that is, the true value.

Next we inspect the MCMC output conditionally on the event that the number of clusters equals 6 and compare to the true parameter values. At first, we can inspect the raw MCMC output, which is not identifiable due to the label switching problem. Thus, this information is not directly returned to the user, however it is saved to the file `'rawMCMC.mapK.6.txt'` in the output directory specified by the output argument. For illustration we plot the raw values of mixture weights. As shown in Figures 4.(c) and 4.(d), the sample is mixing very well to the symmetric posterior areas, since in every iteration labels are changing. The corresponding reordered values (according to the ECR algorithm) are returned to the user as an `"mcmc"` object named `parameters.ecr.mcmc`, shown in Figures 4.(e) and 4.(f). Note that the high posterior density areas are quite close to the true values of relative frequencies of generated observations per cluster (indicated by horizontal lines). Finally, Figures 4.(g) and 4.(h) display the posterior mean estimates (arising from the reordered MCMC sample) versus the true values of θ_{kj} , $k = 1, \dots, 6$; $j = 1, \dots, 100$.

Model selection study

Next we are dealing with model selection issues, that is, selecting the appropriate number of clusters. For this reason we compare **BayesBinMix** with the EM-algorithm implementation provided in **FlexMix**. Under a frequentist framework, the selection of the number of mixture components is feasible using penalized likelihood criteria, such as the BIC (Schwarz, 1978) or ICL (Biernacki et al., 2000), after fitting a mixture model for each possible value of K . We used the ICL criterion since it has been shown to be more robust than BIC, see e.g. Papastamoulis et al. (2016). We considered that the true number of clusters ranges in the set $\{1, 2, \dots, 10\}$ and for each case we simulated 10 datasets using the same data generation procedure as previously but without introducing any missing values due to the fact that **FlexMix** does not handle missing data. The number of observations varies in the set $n \in \{200, 300, 400, 500\}$. For each simulated data the general call is the following.

```
> library("BayesBinMix")
> library("flexmix")
> nChains <- 8
> heats <- seq(1, 0.4, length = nChains)
> cm <- coupledMetropolis(Kmax = 20, nChains = nChains, heats = heats, binaryData = x,
  outPrefix = "sampler", ClusterPrior = "poisson", m = 330, burn = 30)
# now run flexmix for binary data clustering
```

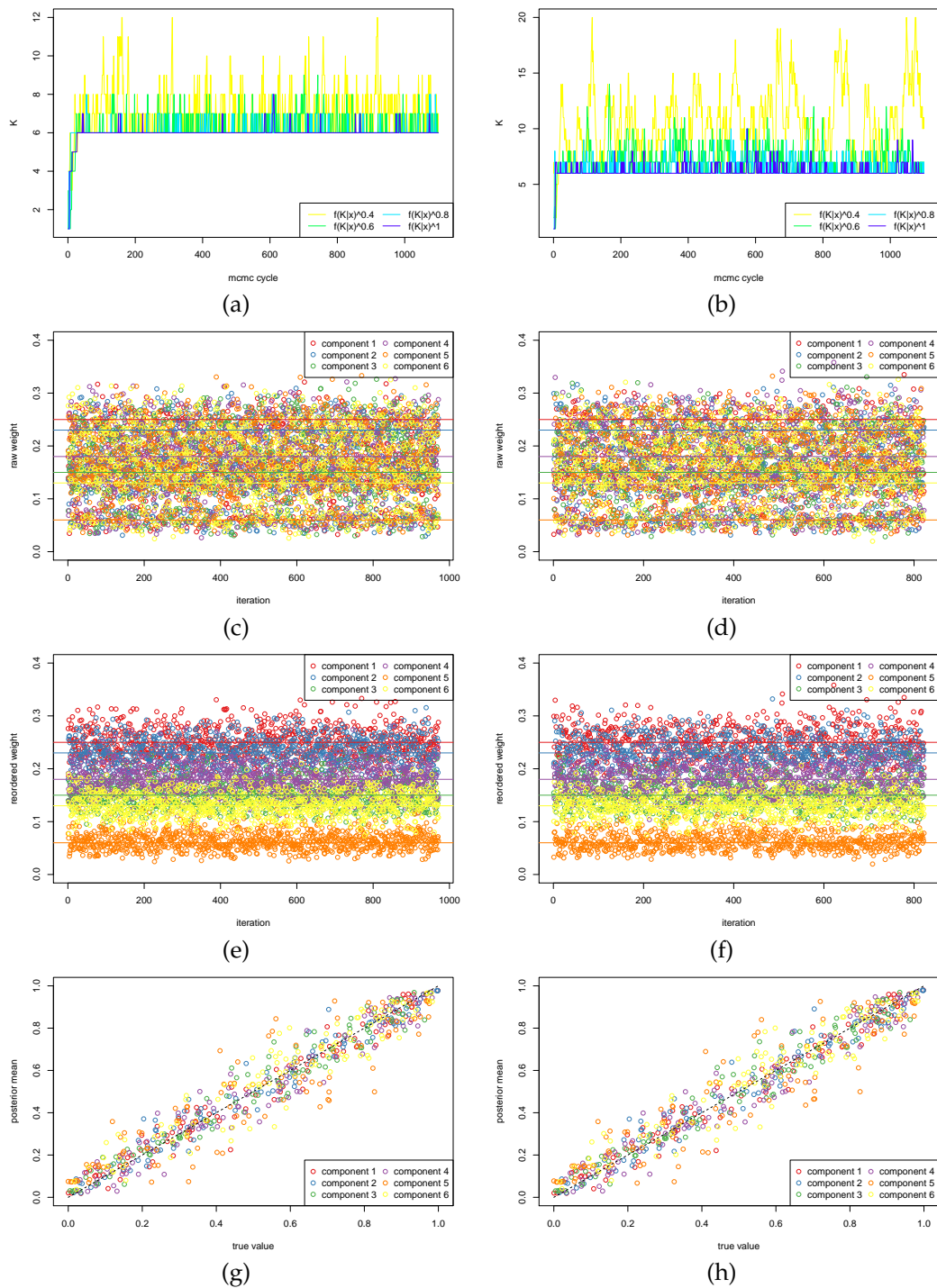


Figure 4: Results for simulated data with $K_{\text{true}} = 6$ using the Poisson (left) or uniform (right) prior distribution on the number of clusters (K). (a) and (b): generated values of K per heated chain. (c) and (d): raw output of p_1, \dots, p_K conditionally on $K = 6$. (e) and (f): reordered sample according to ECR algorithm. Horizontal lines indicate true values of relative number of observations per cluster. (g) and (h): posterior mean estimates of Bernoulli parameters per cluster versus true values.

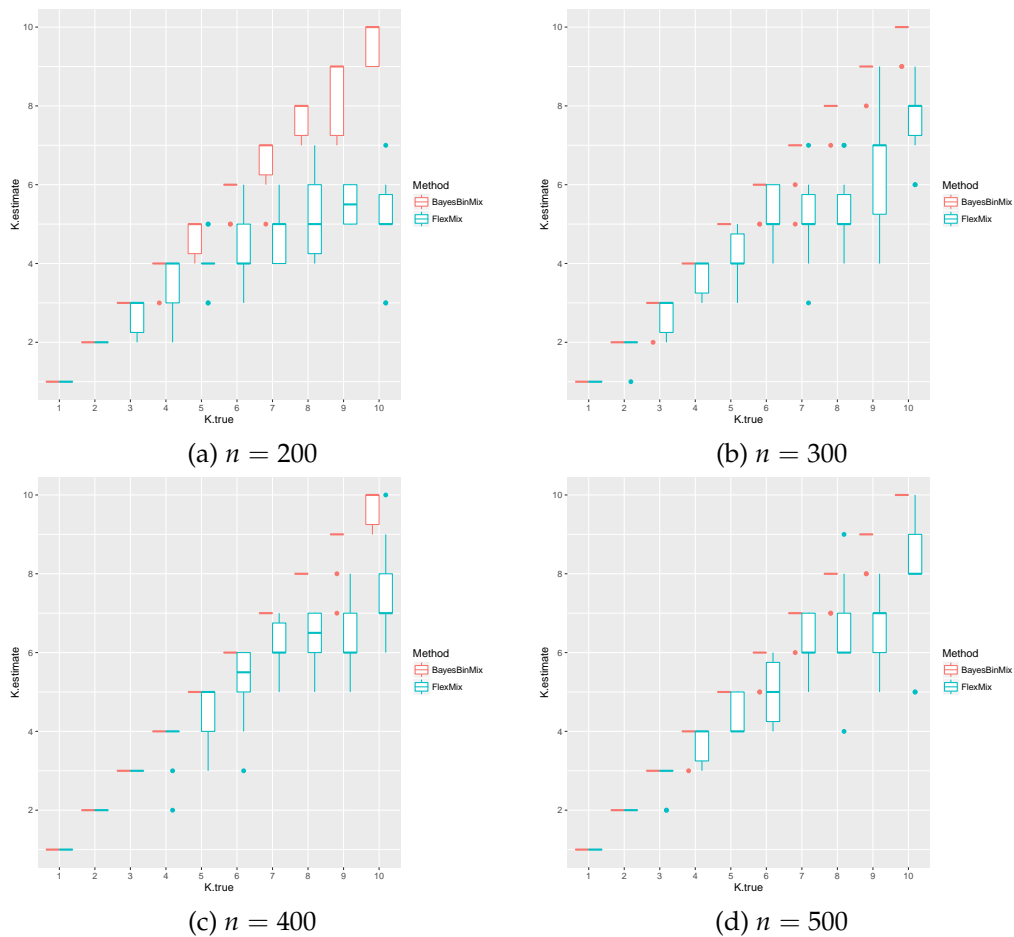


Figure 5: Model selection comparison between **BayesBinMix** and **FlexMix**. The x axis corresponds to the true number of clusters and the y axis to the estimated value. Each boxplot corresponds to 10 simulated datasets from a mixture of Bernoulli distributions.

```
> ex <- initFlexmix(x ~ 1, k = 1:20, model = FLXMCmvbinary(),
  control = list(minprior = 0), nrep = 10)
```

Note that for both algorithms the number of clusters varies in the set $\{1, \dots, 20\}$. Eight heated chains are considered for the MCMC scheme, while each run of the EM algorithm is initialised using $nrep = 10$ different starting points in **FlexMix**. Here we used a total of only $m = 330$ MCMC cycles in order to show that reliable estimates can be obtained using small number of iterations. Figure 5 displays the most probable number of mixture components estimated by **BayesBinMix** and the selected number of clusters using **FlexMix**, for each possible value of the true number of clusters used to simulate the data. Observe that when the number of clusters is less than 5 both methods are able to estimate the true number of mixture components. However, **FlexMix** tends to underestimate the number of clusters when $K \geq 5$, while **BayesBinMix** is able to recover the true value in most cases.

Real data

We consider the zoo database available at the UC Irvine Machine Learning Repository (Lichman, 2013). The database contains 101 animals, each of which has 15 boolean attributes and 1 discrete attribute (legs). The partition of animals into a total of 7 classes (mammal, bird, reptile, fish, amphibian, insect and invertebrate) can be considered as the ground-truth clustering of the data, provided in the vector `z.ground_truth`. Following Li (2005), the discrete variable legs is transformed into six binary features, which correspond to 0, 2, 4, 5, 6 and 8 legs, respectively. Also we eliminate one of the two entries corresponding to frog, as suggested by Li (2005). In total we consider an 100×21 binary array `x` as the input data.

Recall that the Bernoulli mixture in Equation (1) assumes that each cluster consists of a product of independent Bernoulli distributions. Here this assumption is not valid due to the fact that the six new binary variables arising from legs are not independent: they should sum to 1. Nevertheless, it is

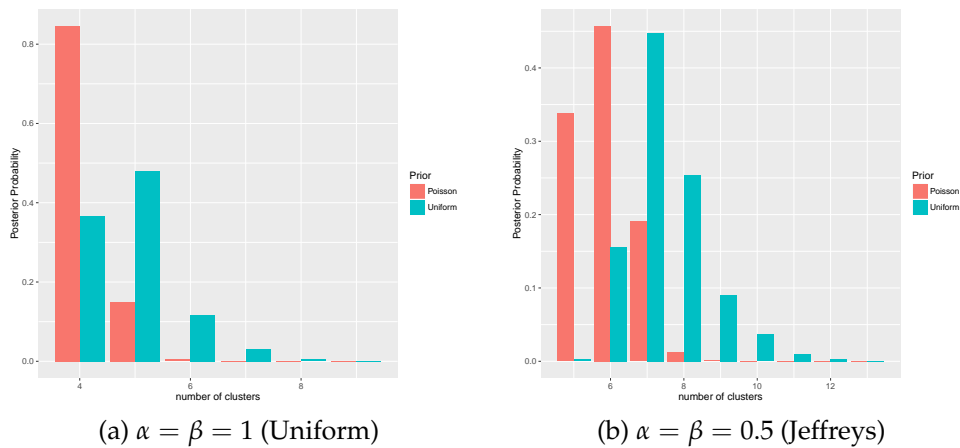


Figure 6: Zoo dataset: Estimated posterior distribution of the number of clusters when using different parameters (α, β) on the beta prior of θ_{kj} . Both choices of the prior on the number of clusters are considered: truncated Poisson (red) and uniform (green).

interesting to see how our method performs in cases that the data is not generated by the assumed model.

We test our method considering both prior assumptions on the number of clusters, as well as different hyper-parameters on the prior distribution of θ_{kj} in Equation (8): we consider $\alpha = \beta = 1$ (default values) as well as $\alpha = \beta = 0.5$. Note that the second choice corresponds to the Jeffreys prior (Jeffreys, 1946) for a Bernoulli trial. Figure 6 displays the estimated posterior distribution of the number of clusters K when $K \in \{1, \dots, 20\}$. This is done with the following commands.

```
# read data
> xOriginal <- read.table("zoo.data", sep = ",")
> x <- xOriginal[, -c(1, 14, 18)]
> x <- x[-27, ] # delete 2nd frog
# now transform v14 into six binary variables
> v14 <- xOriginal[-27, 14]
> newV14 <- array(data = 0, dim = c(100, 6))
> for(i in 1:100){
+   if( v14[i] == 0 ){ newV14[i,1] = 1 }
+   if( v14[i] == 2 ){ newV14[i,2] = 1 }
+   if( v14[i] == 4 ){ newV14[i,3] = 1 }
+   if( v14[i] == 5 ){ newV14[i,4] = 1 }
+   if( v14[i] == 6 ){ newV14[i,5] = 1 }
+   if( v14[i] == 8 ){ newV14[i,6] = 1 }
+ }
> x <- as.matrix(cbind(x, newV14))

# apply BayesBinMix using 8 heated chains
> library("BayesBinMix")
> nChains <- 8
> heats <- seq(1, 0.6, length = nChains)

# K ~ P{1,...,20}, theta_{kj} ~ Beta(1, 1)
> c1 <- coupledMetropolis(Kmax = 20, nChains = nChains, heats =
+   heats, binaryData = x,
+   outPrefix = "poisson-uniform", ClusterPrior = "poisson",
+   m = 4400, burn = 400, z.true = z.ground_truth)

# K ~ U{1,...,20}, theta_{kj} ~ Beta(1, 1)
> c2 <- coupledMetropolis(Kmax = 20, nChains = nChains, heats = heats, binaryData = x,
+   outPrefix = "uniform-uniform", ClusterPrior = "uniform",
+   m = 4400, burn = 400, z.true = z.ground_truth)

# K ~ P{1,...,20}, theta_{kj} ~ Beta(0.5, 0.5)
> c3 <- coupledMetropolis(Kmax = 20, nChains = nChains, heats = heats, binaryData = x,
```

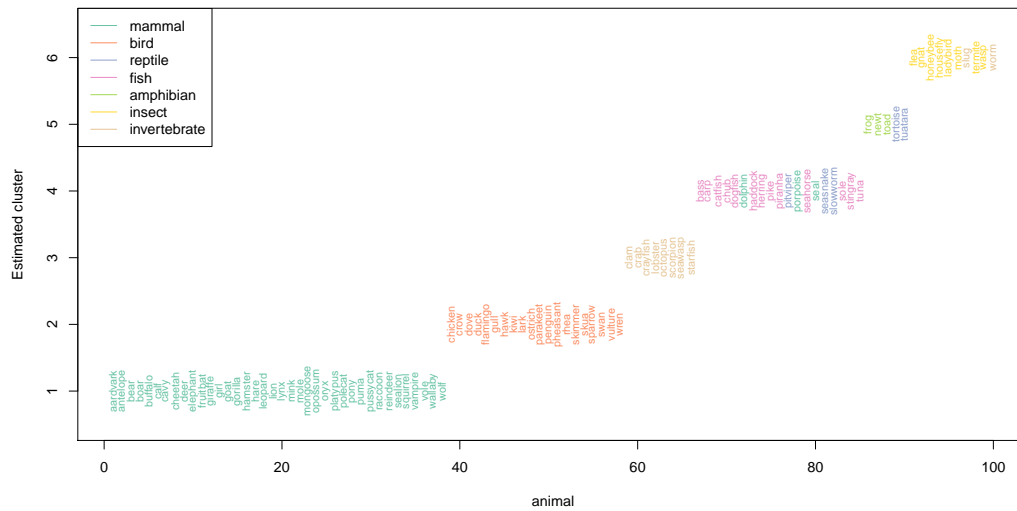


Figure 7: Zoo dataset clustering conditionally on the most probable number of clusters $K = 6$ when using the truncated Poisson prior on K and the Jeffreys prior on θ_{kj} . The ground-truth classification of the animals to seven classes is illustrated using different colour for each class.

```

+                               outPrefix = "poisson-jeffreys", ClusterPrior = "poisson",
+                               m = 4400, burn = 400, z.true = z.ground_truth)

# K ~ U{1,...,20}, theta_{kj} ~ Beta(0.5, 0.5)
> c4 <- coupledMetropolis(Kmax = 20, nChains = nChains, heats = heats, binaryData = x,
+                               outPrefix = "uniform-jeffreys", ClusterPrior = "uniform",
+                               m = 4400, burn = 400, z.true = z.ground_truth)
    
```

Next, we compare the estimated clusters (for the most probable value of K) with the classification of the data into 7 classes (given in the vector `z.ground_truth`). For this reason we provide the `rand` index (adjusted or not) based on the confusion matrix between the estimated and ground-truth clusters, using the package `flexclust` (Leisch, 2006).

```

> library("flexclust")
> z <- array(data = NA, dim = c(100, 4))
> z[, 1] <- c1$clusterMembershipPerMethod$ECR
> z[, 2] <- c2$clusterMembershipPerMethod$ECR
> z[, 3] <- c3$clusterMembershipPerMethod$ECR
> z[, 4] <- c4$clusterMembershipPerMethod$ECR
> rand.index <- array(data = NA, dim = c(4, 3))
> rownames(rand.index) <- c("poisson-uniform", "uniform-uniform",
+                           "poisson-jeffreys", "uniform-jeffreys")
> colnames(rand.index) <- c("K_map", "rand_index", "adjusted_rand_index")
> findMode <- function(x){ as.numeric( names(sort(-table(x$K.mcmc)))[1] ) }
> rand.index[, 1] <- c( findMode(c1), findMode(c2), findMode(c3), findMode(c4) )
> for(i in 1:4){
+     rand.index[i, 2] <- randIndex(table(z[, i], z.ground_truth), correct = FALSE)
+     rand.index[i, 3] <- randIndex(table(z[, i], z.ground_truth))
+ }
> rand.index
      K_map rand_index adjusted_rand_index
poisson-uniform      4  0.9230303          0.7959666
uniform-uniform      5  0.9408081          0.8389208
poisson-jeffreys     6  0.9505051          0.8621216
uniform-jeffreys     7  0.9490909          0.8525556
    
```

Note that both `rand` indices (raw and adjusted) are larger for `z[, 3]`, that is, the six-component mixture model that corresponds to the Poisson prior on K and the Jeffreys prior on θ_{kj} . A detailed view on the estimated clusters for this particular model is shown in Figure 7. We conclude that the estimated groups are characterized by animals belonging to the same taxonomy with very small deviations from the true clusters. Interestingly, in the case that an animal is wrongly assigned to a cluster, notice that

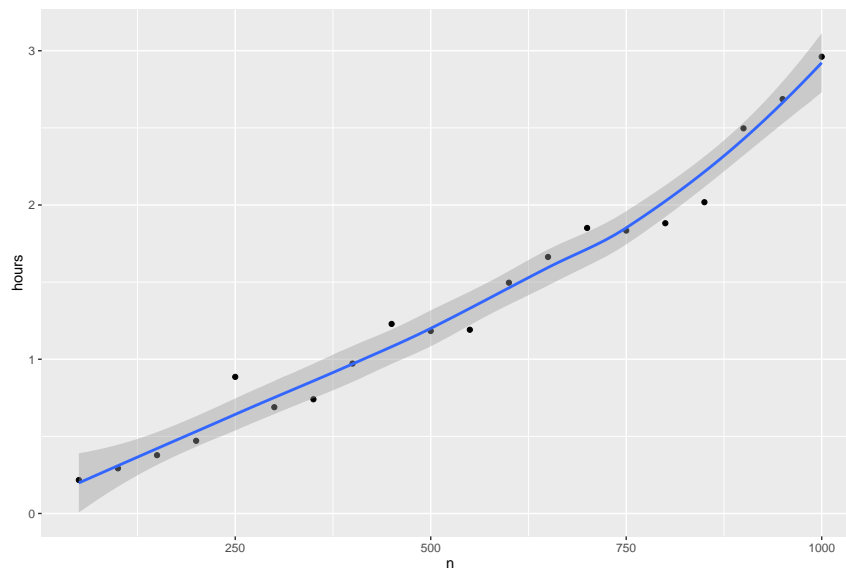


Figure 8: Wall clock time of the `coupledMetropolis()` function when running `nChains = 4` heated chains on the same number of parallel threads for $m = 1100$ cycles, with each cycle consisting of 10 MCMC iterations. At most $k_{\max} = 20$ clusters are allowed. For all sample sizes (n), the dimension of the multivariate data equals to $d = 100$.

the estimated grouping might still make sense: e.g. the sea mammals dolphin, porpoise and seal are assigned to the fourth cluster which is mainly occupied by the group “fish”.

Summary and remarks

The **BayesBinMix** package for fitting mixtures of Bernoulli distributions with an unknown number of components has been presented. The pipeline consists of a fully Bayesian treatment for the clustering of multivariate binary data: it allows the joint estimation of the number of clusters and model parameters, deals with identifiability issues as well as produces a rapidly mixing chain. Using a simulation study we concluded that the method outperforms the EM algorithm in terms of estimating the number of clusters and at the same time produces accurate estimates of the underlying model parameters. In the real dataset we explored the flexibility provided by using different prior assumptions and concluded that the estimated clusters are strongly relevant to the natural grouping of the data.

For the prior distribution on the number of clusters our experience suggests that the truncated Poisson distribution performs better than the uniform (see also [Nobile and Fearnside \(2007\)](#)). Regarding the prior distribution on the Bernoulli parameters we recommend to try both the uniform distribution (default choice) as well as the Jeffreys prior, especially when the sample size is small. An important parameter is the number of heated chains which run in parallel, as well as the temperature of each chain. We suggest to run at least `nChains = 4` heated chains. The heat parameter for each presented example achieved an acceptance ratio of proposed swaps between pairs of chains between 10% and 70%. The default choice for the temperature vector is `heats = seq(1, 0.3, length = nChains)`, however we advise to try different values in case that the swap acceptance ratio is too small (e.g. $< 2\%$) or too large (e.g. $> 90\%$). Finally, we recommend running the algorithm using at least $m = 1100$ and `burn = 100` for total number of MCMC cycles and burn-in period, respectively. For these particular values of `nChains` and m , [Figure 8](#) displays the wall clock time demanded by the `coupledMetropolis()` function.

Acknowledgements

Research was funded by MRC award MR/M02010X/1. The authors would like to thank Rebecca Howard and Dr. Lijing Lin (University of Manchester) for using the software and reporting bugs to earlier versions. We also thank an anonymous reviewer and Roger Bivand, Editor of the R Journal, for their valuable comments and suggestions that considerably improved the package and the presentation of our findings.

Bibliography

- L. Abel, J.-L. Golmard, and A. Mallet. An autologistic model for the genetic analysis of familial binary data. *American journal of human genetics*, 53(4):894, 1993. [p403]
- Y. Al-Ohali, M. Cheriet, and C. Suen. Databases for recognition of handwritten Arabic cheques. *Pattern Recognition*, 36(1):111–121, 2003. URL [https://doi.org/10.1016/s0031-3203\(02\)00064-x](https://doi.org/10.1016/s0031-3203(02)00064-x). [p403]
- E. S. Allman, C. Matias, and J. A. Rhodes. Identifiability of parameters in latent structure models with many observed variables. *The Annals of Statistics*, pages 3099–3132, 2009. URL <https://doi.org/10.1214/09-aos689>. [p407]
- G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist. Parallel Metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference. *Bioinformatics*, 20(3):407–415, 2004. URL <https://doi.org/10.1093/bioinformatics/btg427>. [p407]
- C. Biernacki, G. Celeux, and G. Govaert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7):719–725, 2000. URL <https://doi.org/10.1109/34.865189>. [p403, 412]
- C. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006. [p404]
- D. Böhning. *Computer-Assisted Analysis of Mixtures and Applications: Meta-Analysis, Disease Mapping and Others*, volume 81. CRC press, 2000. [p403]
- M. Á. Carreira-Perpiñán and S. Renals. Practical identifiability of finite mixtures of multivariate Bernoulli distributions. *Neural Computation*, 12(1):141–152, 2000. URL <https://doi.org/10.1162/089976600300015925>. [p407]
- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society B*, 39:1–38, 1977. URL <http://www.jstor.org/stable/2984875>. [p403, 404]
- S. Frühwirth-Schnatter. *Finite Mixture and Markov Switching Models*. Springer-Verlag, New York, 2006. URL <https://doi.org/10.1007/978-0-387-35768-3>. [p403]
- A. E. Gelfand and A. F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of American Statistical Association*, 85:398–409, 1990. URL <https://doi.org/10.2307/2289776>. [p404]
- C. J. Geyer. Markov chain Monte Carlo maximum likelihood. In *Proceedings of the 23rd Symposium on the Interface*, pages 156–163. Interface Foundation, Fairfax Station, Va, 1991. [p407]
- C. J. Geyer and E. A. Thompson. Annealing Markov chain Monte Carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90(431):909–920, 1995. URL <https://doi.org/10.1080/01621459.1995.10476590>. [p407]
- W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, 1996. [p407]
- P. J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995. [p405]
- B. Grün and F. Leisch. Fitting finite mixtures of generalized linear regressions in R. *Computational Statistics & Data Analysis*, 51(11):5247–5252, 2007. URL <https://doi.org/10.1016/j.csda.2006.08.014>. [p409]
- B. Grün and F. Leisch. FlexMix version 2: Finite mixtures with concomitant variables and varying and constant parameters. *Journal of Statistical Software*, 28(4):1–35, 2008. URL <https://doi.org/10.18637/jss.v028.i04>. [p409]
- R. Ilin. Unsupervised learning of categorical data with competing models. *IEEE transactions on neural networks and learning systems*, 23(11):1726–1737, 2012. URL <https://doi.org/10.1109/tnnls.2012.2213266>. [p403]
- H. Jeffreys. An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 186:453–461, 1946. URL <https://doi.org/10.1098/rspa.1946.0056>. [p415]
- A. Juan and E. Vidal. On the use of Bernoulli mixture models for text classification. *Pattern Recognition*, 35(12):2705–2710, 2002. URL [https://doi.org/10.1016/s0031-3203\(01\)00242-4](https://doi.org/10.1016/s0031-3203(01)00242-4). [p403]

- F. Leisch. FlexMix: A general framework for finite mixture models and latent class regression in R. *Journal of Statistical Software*, 11(8):1–18, 2004. URL <https://doi.org/10.18637/jss.v011.i08>. [p409]
- F. Leisch. A toolbox for k-centroids cluster analysis. *Computational Statistics & Data Analysis*, 51(2): 526–544, 2006. URL <https://doi.org/10.1016/j.csda.2005.10.006>. [p416]
- T. Li. A general model for clustering binary data. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 188–197. ACM, 2005. URL <https://doi.org/10.1145/1081870.1081894>. [p403, 414]
- M. Lichman. UCI machine learning repository, 2013. URL <http://archive.ics.uci.edu/ml>. [p414]
- B. G. Lindsay. Mixture models: Theory, geometry and applications. In *NSF-CBMS Regional Conference Series in Probability and Statistics*, pages i–163. JSTOR, 1995. [p403]
- J. S. Liu. The collapsed Gibbs sampler in Bayesian computations with applications to a gene regulation problem. *Journal of the American Statistical Association*, 89(427):958–966, 1994. URL <https://doi.org/10.2307/2290921>. [p405]
- J. M. Marin, K. Mengersen, and C. P. Robert. Bayesian modelling and inference on mixtures of distributions. *Handbook of Statistics*, 25(1):577–590, 2005. URL [https://doi.org/10.1016/s0169-7161\(05\)25016-2](https://doi.org/10.1016/s0169-7161(05)25016-2). [p403, 408]
- J. G. McLachlan and D. Peel. *Finite Mixture Models*. John Wiley & Sons, New York, 2000. URL <https://doi.org/10.1002/0471721182>. [p403]
- A. Nobile and A. T. Fearnside. Bayesian finite mixtures with an unknown number of components: The allocation sampler. *Statistics and Computing*, 17(2):147–162, 2007. URL <https://doi.org/10.1007/s11222-006-9014-7>. [p403, 405, 406, 407, 417]
- P. Papastamoulis. Handling the label switching problem in latent class models via the ECR algorithm. *Communications in Statistics-Simulation and Computation*, 43(4):913–927, 2014. URL <https://doi.org/10.1080/03610918.2012.718840>. [p407]
- P. Papastamoulis. Label.switching: An R package for dealing with the label switching problem in MCMC outputs. *Journal of Statistical Software*, 69(1):1–24, 2016. URL <https://doi.org/10.18637/jss.v069.c01>. [p407]
- P. Papastamoulis and G. Iliopoulos. Reversible jump MCMC in mixtures of normal distributions with the same component means. *Computational Statistics & Data Analysis*, 53(4):900–911, 2009. URL <https://doi.org/10.1016/j.csda.2008.10.022>. [p405]
- P. Papastamoulis and G. Iliopoulos. An artificial allocations based solution to the label switching problem in Bayesian analysis of mixtures of distributions. *Journal of Computational and Graphical Statistics*, 19:313–331, 2010. URL <https://doi.org/10.1198/jcgs.2010.09008>. [p407]
- P. Papastamoulis and G. Iliopoulos. On the convergence rate of random permutation sampler and ECR algorithm in missing data models. *Methodology and Computing in Applied Probability*, 15(2): 293–304, 2013. ISSN 1573-7713. URL <https://doi.org/10.1007/s11009-011-9238-7>. [p407]
- P. Papastamoulis, M.-L. Martin-Magniette, and C. Maugis-Rabusseau. On the estimation of mixtures of Poisson regression models with large number of components. *Computational Statistics & Data Analysis*, 93:97–106, 2016. URL <https://doi.org/10.1016/j.csda.2014.07.005>. [p412]
- M. Plummer, N. Best, K. Cowles, and K. Vines. CODA: Convergence diagnosis and output analysis for MCMC. *R News*, 6(1):7–11, 2006. URL <https://journal.r-project.org/archive/>. [p408]
- R. A. Redner and H. F. Walker. Mixture densities, maximum likelihood and the EM algorithm. *SIAM review*, 26(2):195–239, 1984. URL <https://doi.org/10.1137/1026034>. [p403, 407]
- Revolution Analytics and S. Weston. *foreach: Foreach Looping Construct for R*, 2014. URL <http://CRAN.R-project.org/package=foreach>. R package version 1.4.2. [p408]
- Revolution Analytics and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2015. URL <http://CRAN.R-project.org/package=doParallel>. R package version 1.0.10. [p408]
- S. Richardson and P. J. Green. On Bayesian analysis of mixtures with an unknown number of components. *Journal of the Royal Statistical Society B*, 59(4):731–758, 1997. URL <https://doi.org/10.1111/1467-9868.00095>. [p403, 405]

- C. E. Rodríguez and S. G. Walker. Label switching in Bayesian mixture models: Deterministic relabeling strategies. *Journal of Computational and Graphical Statistics*, 23(1):25–45, 2014. URL <https://doi.org/10.1080/10618600.2012.735624>. [p407]
- G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. URL <https://doi.org/10.1214/aos/1176344136>. [p403, 412]
- M. Stephens. Bayesian analysis of mixture models with an unknown number of components – an alternative to reversible jump methods. *The Annals of Statistics*, 28(1):40–74, 2000a. [p403, 405]
- M. Stephens. Dealing with label switching in mixture models. *Journal of the Royal Statistical Society B*, 62(4):795–809, 2000b. URL <https://doi.org/10.1111/1467-9868.00265>. [p407]
- Z. Sun, O. Rosen, and A. R. Sampson. Multivariate Bernoulli mixture models with application to postmortem tissue studies in schizophrenia. *Biometrics*, 63(3):901–909, 2007. ISSN 1541-0420. URL <https://doi.org/10.1111/j.1541-0420.2007.00762.x>. [p403]
- M. Tanner and W. Wong. The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82(398):528–540, 1987. URL <https://doi.org/10.2307/2289457>. [p404]
- A. White, J. Wyse, and T. B. Murphy. Bayesian variable selection for latent class analysis using a collapsed Gibbs sampler. *Statistics and Computing*, 26(1-2):511–527, 2016. URL <https://doi.org/10.1007/s11222-014-9542-5>. [p403, 405]

Panagiotis Papastamoulis
University of Manchester
Oxford road, Manchester, M13 9PL
United Kingdom
panagiotis.papastamoulis@manchester.ac.uk

Magnus Rattray
University of Manchester
Oxford road, Manchester, M13 9PL
United Kingdom
magnus.rattray@manchester.ac.uk

pdp: An R Package for Constructing Partial Dependence Plots

by Brandon M. Greenwell

Abstract Complex nonparametric models—like neural networks, random forests, and support vector machines—are more common than ever in predictive analytics, especially when dealing with large observational databases that don't adhere to the strict assumptions imposed by traditional statistical techniques (e.g., multiple linear regression which assumes linearity, homoscedasticity, and normality). Unfortunately, it can be challenging to understand the results of such models and explain them to management. Partial dependence plots offer a simple solution. Partial dependence plots are low-dimensional graphical renderings of the prediction function so that the relationship between the outcome and predictors of interest can be more easily understood. These plots are especially useful in explaining the output from black box models. In this paper, we introduce **pdp**, a general R package for constructing partial dependence plots.

Introduction

Harrison and Rubinfeld (1978) were among the first to analyze the well-known Boston housing data. One of their goals was to find a housing value equation using data on median home values from $n = 506$ census tracts in the suburbs of Boston from the 1970 census; see Harrison and Rubinfeld (1978, Table IV) for a description of each variable. The data violate many classical assumptions like linearity, normality, and constant variance. Nonetheless, Harrison and Rubinfeld—using a combination of transformations, significance testing, and grid searches—were able to find a reasonable fitting model ($R^2 = 0.81$). Part of the payoff for their time and efforts was an interpretable prediction equation which is reproduced in Equation (1).

$$\begin{aligned} \log(\widehat{MV}) = & 9.76 + 0.0063RM^2 + 8.98 \times 10^{-5}AGE - 0.19 \log(DIS) + 0.096 \log(RAD) \\ & - 4.20 \times 10^{-4}TAX - 0.031PTRATIO + 0.36(B - 0.63)^2 - 0.37 \log(LSTAT) \\ & - 0.012CRIM + 8.03 \times 10^{-5}ZN + 2.41 \times 10^{-4}INDUS + 0.088CHAS \\ & - 0.0064NOX^2. \end{aligned} \quad (1)$$

Nowadays, many supervised learning algorithms can fit the data automatically in seconds—typically with higher accuracy. (We will revisit the Boston housing data in Section X.2.) The downfall, however, is some loss of interpretation since these algorithms typically do not produce simple prediction formulas like Equation (1). These models can still provide insight into the data, but it is not in the form of simple equations. For example, quantifying predictor importance has become an essential task in the analysis of "big data", and many supervised learning algorithms, like tree-based methods, can naturally assign variable importance scores to all of the predictors in the training data.

While determining predictor importance is a crucial task in any supervised learning problem, ranking variables is only part of the story and once a subset of "important" features is identified it is often necessary to assess the relationship between them (or subset thereof) and the response. This can be done in many ways, but in machine learning it is often accomplished by constructing *partial dependence plots* (PDPs); see Friedman (2001) for details. PDPs help visualize the relationship between a subset of the features (typically 1-3) and the response while accounting for the average effect of the other predictors in the model. They are particularly effective with black box models like random forests and support vector machines.

Let $x = \{x_1, x_2, \dots, x_p\}$ represent the predictors in a model whose prediction function is $\hat{f}(x)$. If we partition x into an interest set, z_s , and its complement, $z_c = x \setminus z_s$, then the "partial dependence" of the response on z_s is defined as

$$f_s(z_s) = E_{z_c} [\hat{f}(z_s, z_c)] = \int \hat{f}(z_s, z_c) p_c(z_c) dz_c, \quad (2)$$

where $p_c(z_c)$ is the marginal probability density of z_c : $p_c(z_c) = \int p(x) dz_s$. Equation (2) can be estimated from a set of training data by

$$\bar{f}_s(z_s) = \frac{1}{n} \sum_{i=1}^n \hat{f}(z_s, z_{i,c}), \quad (3)$$

where $z_{i,c}$ ($i = 1, 2, \dots, n$) are the values of z_c that occur in the training sample; that is, we average out the effects of all the other predictors in the model.

Constructing a PDP (3) in practice is rather straightforward. To simplify, let $z_s = x_1$ be the predictor variable of interest with unique values $\{x_{11}, x_{12}, \dots, x_{1k}\}$. The partial dependence of the response on x_1 can be constructed as follows:

Algorithm 1 A simple algorithm for constructing the partial dependence of the response on a single predictor x_1 .

1. For $i \in \{1, 2, \dots, k\}$:
 - (a) Copy the training data and replace the original values of x_1 with the constant x_{1i} .
 - (b) Compute the vector of predicted values from the modified copy of the training data.
 - (c) Compute the average prediction to obtain $\bar{f}_1(x_{1i})$.
 2. Plot the pairs $\{x_{1i}, \bar{f}_1(x_{1i})\}$ for $i = 1, 2, \dots, k$.
-

Algorithm 1 can be quite computationally intensive since it involves k passes over the training records. Fortunately, the algorithm can be parallelized quite easily (more on this in Section X.2.4). It can also be easily extended to larger subsets of two or more features as well.

Limited implementations of Friedman's PDPs are available in packages **randomForest** (Liaw and Wiener, 2002) and **gbm** (Ridgeway, 2017), among others; these are limited in the sense that they only apply to the models fit using the respective package. For example, the `partialPlot` function in **randomForest** only applies to objects of class "randomForest" and the `plot` function in **gbm** only applies to "gbm" objects. While the **randomForest** implementation will only allow for a single predictor, the **gbm** implementation can deal with any subset of the predictor space. Partial dependence functions are not restricted to tree-based models; they can be applied to any supervised learning algorithm (e.g., generalized additive models and neural networks). However, to our knowledge, there is no general package for constructing PDPs in R. For example, PDPs for a conditional random forest as implemented by the `cforest` function in the **party** and **partykit** packages; see Hothorn et al. (2017) and Hothorn and Zeileis (2016), respectively. The **pdp** (Greenwell, 2017) package tries to close this gap by offering a general framework for constructing PDPs that can be applied to several classes of fitted models.

The **plotmo** package (Milborrow, 2017b) is one alternative to **pdp**. According to Milborrow, **plotmo** constructs "a poor man's partial dependence plot." In particular, it plots a model's response when varying one or two predictors while holding the other predictors in the model constant (continuous features are fixed at their median value, while factors are held at their first level). These plots allow for up to two variables at a time. They are also less accurate than PDPs, but are faster to construct. For additive models (i.e., models with no interactions), these plots are identical in shape to PDPs. As of **plotmo** version 3.3.0, there is now support for constructing PDPs, but it is not the default. The main difference is that **plotmo**, rather than applying step 1. (a)-(c) in Algorithm 1, accumulates all the data at once thereby reducing the number of internal calls to `predict`. The trade-off is a slight increase in speed at the expense of using more memory. So, why use the **pdp** package? As will be discussed in the upcoming sections, **pdp**:

- contains only a few functions with relatively few arguments;
- does not produce a plot by default;
- can be used more efficiently with "gbm" objects (see Section X.2.4);
- produces graphics based on **lattice** (Sarkar, 2008), which are more flexible than base R graphics;
- defaults to using false color level plots for multivariate displays (see Section X.2.2);
- contains options to mitigate the risks associated with extrapolation (see Section X.2.4);
- has the option to display progress bars (see Section X.2.4);
- has the option to construct PDPs in parallel (see Section X.2.4);
- is extremely flexible in the types of PDPs that can be produced (see Section X.2.6),

PDPs can be misleading in the presence of substantial interactions (Goldstein et al., 2015). To overcome this issue Goldstein, Kapelner, Bleich, and Pitkin developed the concept of *individual conditional expectation* (ICE) plots—available in the **ICEbox** package. ICE plots display the estimated relationship between the response and a predictor of interest for each observation. Consequently, the

PDP for a predictor of interest can be obtained by averaging the corresponding ICE curves across all observations. In Section X.2.6, it is shown how to obtain ICE curves using the **pdp** package. It is also possible to display the PDP for a single predictor with **ICEbox**; see `?ICEbox::plot.ice` for an example. **ICEbox** only allows for one variable at a time (i.e., no multivariate displays), though color can be used effectively to display information about an additional predictor. The ability to construct centered ICE (c-ICE) plots and derivative ICE (d-ICE) plots is also available in **ICEbox**; c-ICE plots help visualize heterogeneity in the modeled relationship between observations, and d-ICE plots help to explore interaction effects.

Many other techniques exist for visualizing relationships between the predictors and the response based on a fitted model. For example, the **car** package (Fox and Weisberg, 2011) contains many functions for constructing *partial-residual* and *marginal-model* plots. *Effect displays*, available in the **effects** package (Fox, 2003), provide tabular and graphical displays for the terms in parametric models while holding all other predictors at some constant value—similar in spirit to **plotmo**'s marginal model plots. However, these methods were designed for simpler parametric models (e.g., linear and generalized linear models), whereas **plotmo**, **ICEbox**, and **pdp** are more useful for black box models (although, they can be used for simple parametric models as well).

Constructing PDPs in R

The **pdp** package is useful for constructing PDPs for many classes of fitted models in R. PDPs are especially useful for visualizing the relationships discovered by complex machine learning algorithms such as a random forest. The latest stable release is available from CRAN. The development version is located on GitHub: <https://github.com/bgreenwell/pdp>. Bug reports and suggestions are appreciated and should be submitted to <https://github.com/bgreenwell/pdp/issues>. The two most important functions exported by **pdp** are:

- `partial`
- `plotPartial`

The `partial` function evaluates the partial dependence (3) from a fitted model over a grid of predictor values; the fitted model and predictors are specified using the `object` and `pred.var` arguments, respectively—these are the only required arguments. If `plot = FALSE` (the default), `partial` returns an object of class "partial" which inherits from the class "data.frame"; put another way, by default, `partial` returns a data frame with an additional class that is recognized by the `plotPartial` function. The columns of the data frame are labeled in the same order as the features supplied to `pred.var`, and the last column is labeled \hat{y} ¹ and contains the values of the partial dependence function $\bar{f}_s(z_s)$. If `plot = TRUE`, then `partial` makes an internal call to `plotPartial` (with fewer plotting options) and returns the PDP in the form of a **lattice** plot (i.e., a "trellis" object). **Note:** it is recommended to call `partial` with `plot = FALSE` and store the results; this allows for more flexible plotting, and the user will not have to waste time calling `partial` again if the default plot is not sufficient.

The `plotPartial` function can be used for displaying more advanced PDPs; it operates on objects of class "partial" and has many useful plotting options. For example, `plotPartial` makes it straight forward to add a LOESS smooth, or produce a 3-D surface instead of a false color level plot (the default). Of course, since the default output produced by `partial` is still a data frame, the user can easily use any plotting package he/she desires to visualize the results—**ggplot2** (Wickham, 2009), for instance (see Section X.2.5 and Section X.2.6 for examples).

Note: as mentioned above, **pdp** relies on **lattice** for its graphics. **lattice** itself is built on top of **grid** (R Core Team, 2017). **grid** graphics behave a little differently than traditional R graphics, and two points are worth making (see `?lattice` for more details):

1. **lattice** functions return a "trellis" object, but do not display it; the `print` method produces the actual display. However, due to R's automatic printing rule, the result is automatically printed when using these functions in the command line. If `plotPartial` is called inside of source or inside a loop (e.g., `for` or `while`), an explicit `print` statement is required to display the resulting graph; hence, the same is true when using `partial` with `plot = TRUE`.
2. Setting graphical parameters via `par` typically has no effect on **lattice** plots. Instead, **lattice** provides its own `trellis.par.set` function for modifying graphical parameters.

A consequence of the second point is that the `par` function cannot be used to control the layout of multiple **lattice** (and hence **pdp**) plots. Simple solutions are available in packages **latticeExtra** (Sarkar and Andrews, 2016) and **gridExtra** (Auguie, 2016). For convenience, **pdp** imports the `grid.arrange`

¹There is one exception to this. When a function supplied via the `pred.fun` argument returns multiple predictions, the second to last and last columns will be labeled `yhat` and `yhat.id`, respectively (see Section X.2.6).

function from **gridExtra** which makes it easy to display multiple **grid**-based graphical objects on a single plot (these include graphics produced using **lattice** (hence, **pdp**) and **ggplot2**). This is demonstrated in multiple examples throughout this paper.

Currently supported models are described in Table 1. In these cases, the user does not need to supply a prediction function (more on this in Section X.2.6) or a value for the type argument (i.e., "regression" or "classification"). In other situations, the user may need to specify one or both of these arguments. This allows **partial** to be flexible enough to handle many of the model types not listed in Table 1; for example, neural networks from the **nnet** package (Venables and Ripley, 2002).

Type of model	R package	Object class
Decision tree	C50 (Kuhn et al., 2015)	"C5.0"
	party	"BinaryTree"
	partykit	"party"
	rpart (Therneau et al., 2017)	"rpart"
Bagged decision trees	adabag (Alfaro et al., 2013)	"bagging"
	ipred (Peters and Hothorn, 2017)	"classbagg", "regbagg"
Boosted decision trees	adabag (Alfaro et al., 2013)	"boosting"
	gbm	"gbm"
	xgboost	"xgb.Booster"
Cubist	Cubist (Kuhn et al., 2016)	"cubist"
Discriminant analysis	MASS (Venables and Ripley, 2002)	"lda", "qda"
Generalized linear model	stats	"glm", "lm"
Linear model	stats	"lm"
Nonlinear least squares	stats	"nls"
Multivariate adaptive regression splines (MARS)	earth (Milborrow, 2017a)	"earth"
	mda (Leisch et al., 2016)	"mars"
Projection pursuit regression	stats	"ppr"
Random forest	randomForest	"randomForest"
	party	"RandomForest"
	partykit	"cforest"
	ranger (Wright, 2017)	"ranger"
Support vector machine	e1071 (Meyer et al., 2017)	"svm"
	kernlab (Karatzoglou et al., 2004)	"ksvm"

Table 1: Models specifically supported by the **pdp** package. **Note:** for some of these cases, the user may still need to supply additional arguments in the call to **partial**.

The **partial** function also supports objects of class "train" produced using the **train** function from the well-known **caret** package (Kuhn, 2017). This means that **partial** can be used with any classification or regression model that has been fit using **caret**'s **train** function; see <http://topepo.github.io/caret/available-models.html> for a current list of models supported by **caret**. An example is given in Section X.2.7.

Another important argument to **partial** is **train**. If **train** = NULL (the default), **partial** tries to extract the original training data from the fitted model object. For objects that typically store a copy of the training data (e.g., objects of class "BinaryTree", "RandomForest", and "train"), this is straightforward. Otherwise, **partial** will attempt to extract the call stored in object (if available) and use that to evaluate the training data in the same environment from which **partial** was called. This can cause problems when, for example, the training data have been changed after fitting the model, but before calling **partial**. Hence, it is good practice to always supply the training data via the **train** argument in the call to **partial**². If **train** = NULL and the training data can not be extracted from the fitted model, the user will be prompted with an informative error message (this will occur, for example, when using **partial** with "ksvm" and "xgb.Booster" objects):

```
Error: The training data could not be extracted from object. Please supply
the raw training data using the `train` argument in the call to `partial`.
```

²For brevity, we ignore this option in most of the examples in this paper.

For illustration, we will use a corrected version of the Boston housing data analyzed in [Harrison and Rubinfeld \(1978\)](#); the data are available in the `pdp` package (see `?pdp::boston` for details). We begin by loading the data and fitting a random forest with default tuning parameters and 500 trees:

```
data(boston, package = "pdp") # load the (corrected) Boston housing data
library(randomForest) # for randomForest, partialPlot, and varImpPlot functions
set.seed(101) # for reproducibility
boston.rf <- randomForest(cmedv ~ ., data = boston, importance = TRUE)
varImpPlot(boston.rf) # Figure 1
```

The model fit is reasonable, with an *out-of-bag* (pseudo) R^2 of 0.89. The variable importance scores are displayed in Figure 1. Both plots indicate that the percentage of lower status of the population (`lstat`) and the average number of rooms per dwelling (`rm`) are highly associated with the median value of owner-occupied homes (`cmedv`). The question then arises, "What is the nature of these associations?" To help answer this, we can look at the partial dependence of `cmedv` on `lstat` and `rm`, both individually and together.

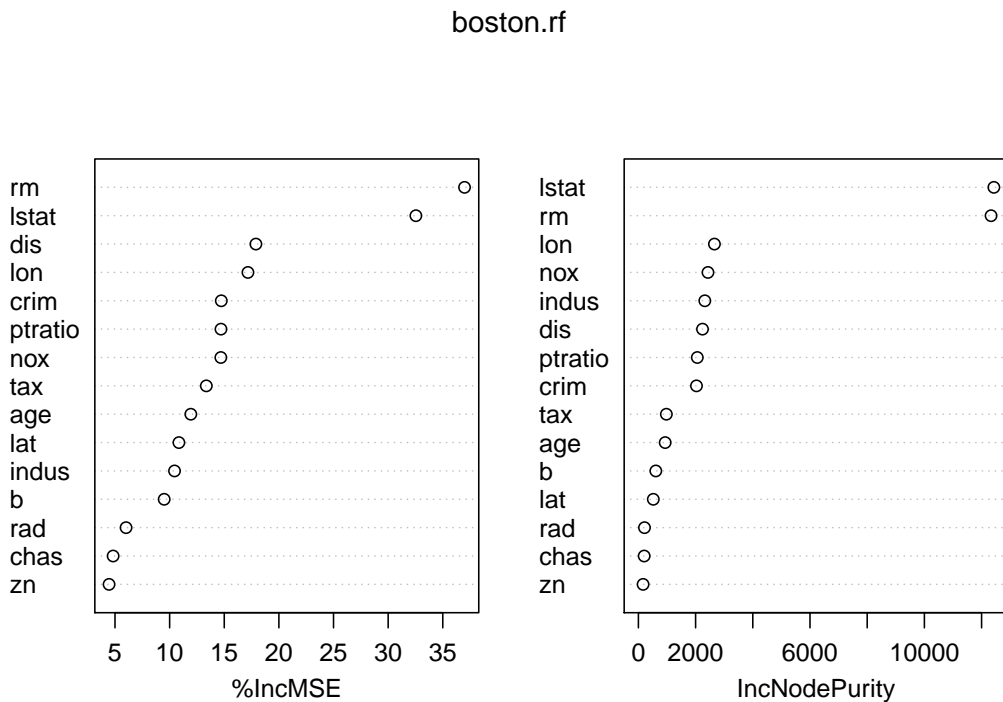


Figure 1: Dotchart of variable importance scores for the Boston housing data based on a random forest with 500 trees.

Single predictor PDPs

As previously mentioned, the `randomForest` package has its own `partialPlot` function for visualizing the partial dependence of the response on a single predictor—the keywords here are "single predictor". For example, the following snippet of code plots the partial dependence of `cmedv` on `lstat`:

```
partialPlot(boston.rf, pred.data = boston, x.var = "lstat")
```

The same plot can be achieved using the `partial` function and setting `plot = TRUE` (see the left side of Figure 2):

```
library(pdp) # for partial, plotPartial, and grid.arrange functions
partial(boston.rf, pred.var = "lstat", plot = TRUE) # Figure 2 (left)
```

The only difference is that `pdp` uses the `lattice` graphics package to produce all of its displays.

For a more customizable plot, we can set `plot = FALSE` in the call to `partial` and then use the `plotPartial` function on the resulting data frame. This is illustrated in the example below which increases the line width, adds a LOESS smooth, and customizes the *y*-axis label. The result is displayed

in the right side of Figure 2. **Note:** to encourage writing more readable code, the *pipe* operator `%>%` provided by the `magrittr` package (Bache and Wickham, 2014) is exported whenever `pdp` is loaded.

```
# Figure 2 (right)
boston.rf %>% # the %>% operator is read as "and then"
  partial(pred.var = "lstat") %>%
  plotPartial(smooth = TRUE, lwd = 2, ylab = expression(f(lstat)))
```

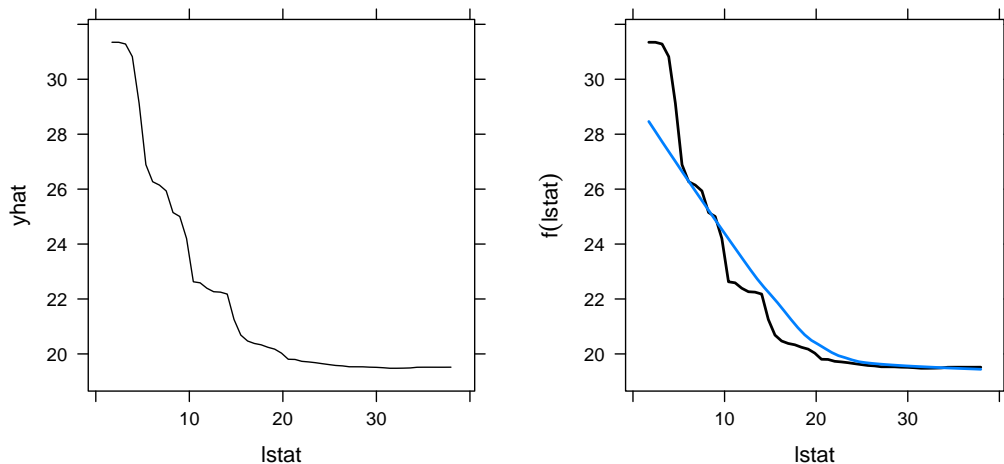


Figure 2: Partial dependence of `cmedv` on `lstat` based on a random forest. *Left:* Default plot. *Right:* Customized plot obtained using the `plotPartial` function.

Multi-predictor PDPs

The benefit of using `partial` is threefold: (1) it is a flexible, generic function that can be used to obtain different kinds of PDPs for various types of fitted models (not just random forests), (2) it will allow for any number of predictors to be used (e.g., multivariate displays), and (3) it can utilize any of the parallel backends supported by the `foreach` package (Revolution Analytics and Weston, 2015c); we discuss parallel execution in a later section. For example, the following code chunk uses the random forest model to assess the joint effect of `lstat` and `rm` on `cmedv`. The `grid.arrange` function is used to display three PDPs, which make use of various `plotPartial` options³, on the same graph. The results are displayed in Figure 3.

```
# Compute partial dependence data for lstat and rm
pd <- partial(boston.rf, pred.var = c("lstat", "rm"))

# Default PDP
pdp1 <- plotPartial(pd)

# Add contour lines and use a different color palette
rwb <- colorRampPalette(c("red", "white", "blue"))
pdp2 <- plotPartial(pd, contour = TRUE, col.regions = rwb)

# 3-D surface
pdp3 <- plotPartial(pd, levelplot = FALSE, zlab = "cmedv", drape = TRUE,
  colorkey = TRUE, screen = list(z = -20, x = -60))

# Figure 3
grid.arrange(pdp1, pdp2, pdp3, ncol = 3)
```

Note: the default color map for level plots is the color blind-friendly `matplotlib` (Hunter, 2007) ‘`viridis`’ color map provided by the `viridis` package (Garnier, 2017).

³See Section X.2.4 for an example of how to add a label to the `colorkey` in these types of graphs.

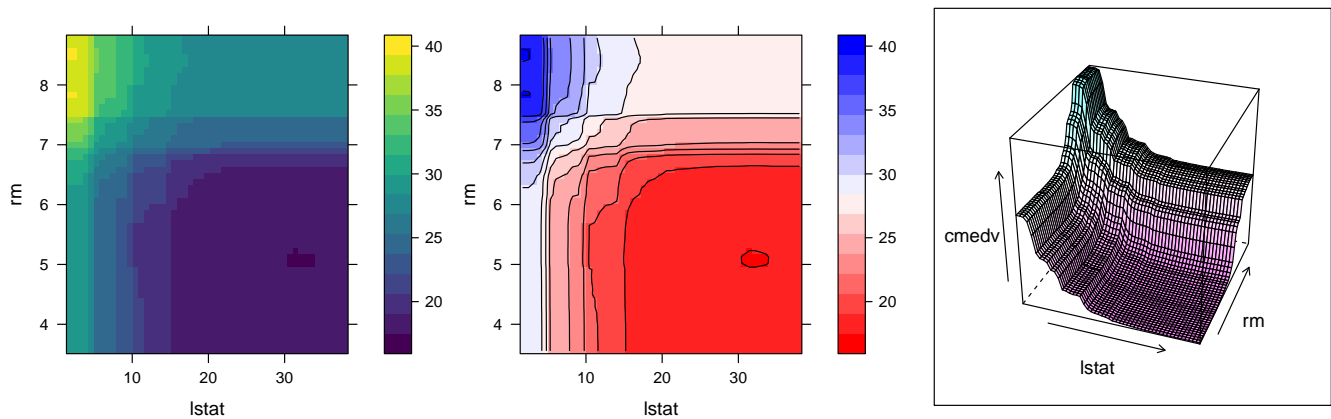


Figure 3: Partial dependence of `cmedv` on `lstat` and `rm` based on a random forest. *Left:* Default plot. *Middle:* With contour lines and a different color palette. *Right:* Using a 3-D surface.

Avoiding extrapolation

It is not wise to draw conclusions from PDPs in regions outside the area of the training data. Here we describe two ways to mitigate the risk of extrapolation in PDPs: rug displays and convex hulls. Rug displays are one-dimensional plots added to the axes. Both `partial` and `plotPartial` have a `rug` option that, when set to `TRUE`, will display the deciles of the distribution (as well as the minimum and maximum values) for the predictors on the horizontal and vertical axes. The following snippet of code produces the left display in Figure 4.

```
# Figure 4 (left)
partial(boston.rf, pred.var = "lstat", plot = TRUE, rug = TRUE)
```

In two or more dimensions, plotting the convex hull is more informative; it outlines the region of the predictor space that the model was trained on. When `chull = TRUE`, the convex hull of the first two dimensions of z_s (i.e., the first two variables supplied to `pred.var`) is computed; for example, if you set `chull = TRUE` in the call to `partial` only the region within the convex hull of the first two variables is plotted. Over interpreting the PDP outside of this region is considered extrapolation and is ill-advised. The right display in Figure 4 was produced using:

```
# Figure 4 (right)
partial(boston.rf, pred.var = c("lstat", "rm"), plot = TRUE, chull = TRUE)
```

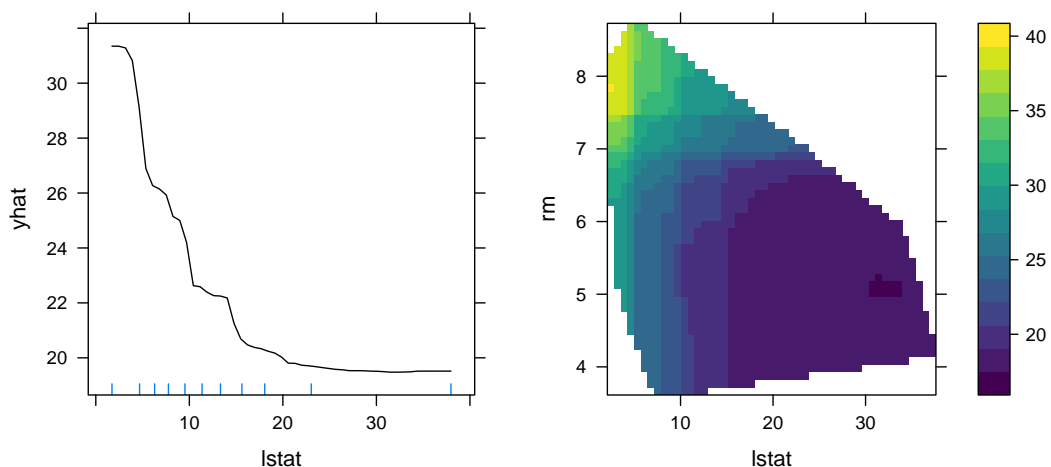


Figure 4: Examples of PDPs with the addition of a rug display (left) and a convex hull (right).

Addressing computational concerns

Constructing PDPs can be quite computationally expensive⁴ Several strategies are available to ease the computational burden in larger problems. For example, there is no need to compute partial dependence of `cmdev` using each unique value of `rm` in the training data (which would require $k = 446$ passes over the data!). We could get very reasonable results using a reduced number of points. Current options are to use a grid of equally spaced values in the range of the variable of interest; the number of points can be controlled using the `grid.resolution` option in the call to `partial`. Alternatively, a user-specified grid of values (e.g., containing specific quantiles of interest) can be supplied through the `pred.grid` argument. To demonstrate, the following snippet of code computes the partial dependence of `cmdev` on `rm` using each option; `grid.arrange` is used to display all three PDPs on the same graph, side by side. The results are displayed in Figure 5.

```
# Figure 5
grid.arrange(
  partial(boston.rf, "rm", plot = TRUE),
  partial(boston.rf, "rm", grid.resolution = 30, plot = TRUE),
  partial(boston.rf, "rm", pred.grid = data.frame(rm = 3:9), plot = TRUE),
  ncol = 3
)
```

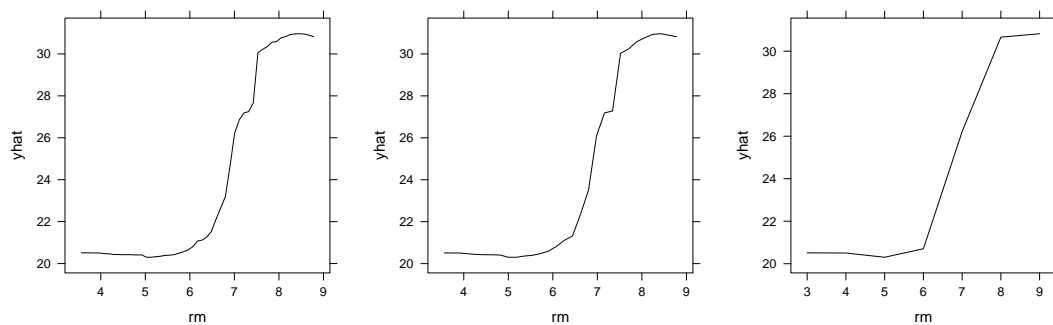


Figure 5: Partial dependence of `cmdev` on `rm`. *Left:* Default plot. *Middle:* Using a reduced grid size. *Right:* Using a user-specified grid.

The partial function relies on the `plyr` package (Wickham, 2011), rather than R's built-in for loops. This makes it easy to request progress bars (e.g., `progress = "text"`) or run `partial` in parallel. In fact, `partial` can use any of the parallel backends supported by the `foreach` package. To use this functionality, we must first load and register a supported parallel backend [e.g., `doMC` (Revolution Analytics and Weston, 2015a) or `doParallel` (Revolution Analytics and Weston, 2015b)].

To illustrate, we will use the Los Angeles ozone pollution data described in Breiman and Friedman (1985). The data contain daily measurements of ozone concentration (`ozone`) along with eight meteorological quantities for 330 days in the Los Angeles basin in 1976.⁵ The following code chunk loads the data into R:

```
ozone <- read.csv(paste0("http://statweb.stanford.edu/~tibs/ElemStatLearn/",
  "datasets/LAozone.data"), header = TRUE)
```

Next, we use the multivariate adaptive regression splines (MARS) algorithm introduced in Friedman (1991) to model ozone concentration as a nonlinear function of the eight meteorological variables plus day of the year; we allow for up to three-way interactions.

```
library(earth) # for earth function (i.e., MARS algorithm)
ozone.mars <- earth(ozone ~ ., data = ozone, degree = 3)
summary(ozone.mars)
```

The MARS model produced a generalized R^2 of 0.79, similar to what was reported in Breiman and Friedman (1985). A single three-way interaction was found involving the predictors

⁴The exception is regression trees based on single-variable splits which can make use of the efficient weighted tree traversal method described in Friedman (2001), however, only the `gbm` package seems to make use of this approach; consequently, `pdp` can also exploit this strategy when used with `gbm` models (see `?partial` for details).

⁵The data are available from <http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/LAozone.data>. Details, including variable information, are available from <http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/LAozone.info>.

- wind: wind speed (mph) at Los Angeles International Airport (LAX)
- temp: temperature ($^{\circ}F$) at Sandburg Air Force Base
- dpg: the pressure gradient (mm Hg) from LAX to Dagget, CA

To understand this interaction, we can use a PDP. However, since the partial dependence between three continuous variables can be computationally expensive, we will run `partial` in parallel.

Setting up a parallel backend is rather straightforward. To demonstrate, the following snippet of code sets up the partial function to run in parallel on both Windows and Unix-like systems using the `doParallel` package.

```
library(doParallel) # load the parallel backend
cl <- makeCluster(4) # use 4 workers
registerDoParallel(cl) # register the parallel backend
```

Now, to run `partial` in parallel, all we have to do is invoke the `parallel = TRUE` and `paropts` options and the rest is taken care of by the internal call to `plyr` and the parallel backend we loaded⁶. This is illustrated in the code chunk below which obtains the partial dependence of ozone on wind, temp, and dpg in parallel. The last three lines of code add a label to the colorkey. The result is displayed in Figure 6. **Note:** it is considered good practice to shut down the workers by calling `stopCluster` when finished.

```
partial(ozone.mars, pred.var = c("wind", "temp", "dpg"), plot = TRUE,
        chull = TRUE, parallel = TRUE, paropts = list(.packages = "earth")) # Figure 6
stopCluster(cl) # good practice
```

```
# Add a label to the colorkey
lattice::trellis.focus("legend", side = "right", clipp.off = TRUE, highlight = FALSE)
grid::grid.text("ozone", x = 0.2, y = 1.05, hjust = 0.5, vjust = 1)
lattice::trellis.unfocus()
```

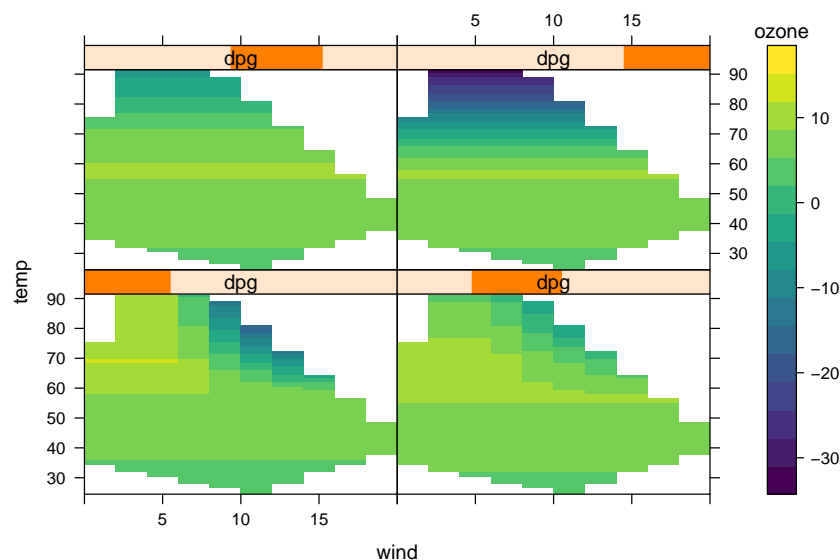


Figure 6: Partial dependence of ozone on wind, temp, and dpg. Since dpg is continuous, it is first converted to a shingle; in this case, four groups with 10% overlap.

It is important to note that when using more than two predictor variables, `plotPartial` produces a trellis display. The first two variables given to `pred.var` are used for the horizontal and vertical axes, and additional variables define the panels. If the panel variables are continuous, then shingles⁷ are produced first using the equal count algorithm (see, for example, `?lattice::equal.count`). Hence, it will be more effective to use categorical variables to define the panels in higher dimensional displays when possible.

⁶Notice we have to pass the names of external packages that the tasks depend on via the `paropts` argument; in this case, "earth". See `?plyr::adply` for details.

⁷A shingle is a special Trellis data structure that consists of a numeric vector along with intervals that define the "levels" of the shingle. The intervals may be allowed to overlap.

Classification problems

Traditionally, for classification problems, partial dependence functions are on a scale similar to the logit; see, for example, [Hastie et al. \(2009, pp. 369–370\)](#). Suppose the response is categorical with K levels, then for each class we compute

$$f_k(x) = \log [p_k(x)] - \frac{1}{K} \sum_{k=1}^K \log [p_k(x)], \quad k = 1, 2, \dots, K, \quad (4)$$

where $p_k(x)$ is the predicted probability for the k -th class. Plotting $f_k(x)$ helps us understand how the log-odds for the k -th class depends on different subsets of the predictor variables.

To illustrate, we consider Edgar Anderson's iris data from the `datasets` package. The `iris` data frame contains the sepal length, sepal width, petal length, and petal width (in centimeters) for 50 flowers from each of three species of iris: `setosa`, `versicolor`, and `virginica`. We fit a support vector machine with a Gaussian radial basis function kernel to the data using the `svm` function in the `e1071` package (the tuning parameters were determined using 5-fold cross-validation).

```
library(e1071) # for svm function
iris.svm <- svm(Species ~ ., data = iris, kernel = "radial", gamma = 0.75,
               cost = 0.25, probability = TRUE)
```

Note: the partial function has to be able to extract the predicted probabilities for each class, so it is necessary to set `probability = TRUE` in the call to `svm`.

Next, we plot the partial dependence of `Species` on both `Petal.Width` and `Petal.Length` for each of the three classes. The result is displayed in [Figure 7](#).

```
pd <- NULL
for (i in 1:3) {
  tmp <- partial(iris.svm, pred.var = c("Petal.Width", "Petal.Length"),
                which.class = i, grid.resolution = 101, progress = "text")
  pd <- rbind(pd, cbind(tmp, Species = levels(iris$Species)[i]))
}

# Figure 7
library(ggplot2)
ggplot(pd, aes(x = Petal.Width, y = Petal.Length, z = yhat, fill = yhat)) +
  geom_tile() +
  geom_contour(color = "white", alpha = 0.5) +
  scale_fill_distiller(name = "Centered\nlogit", palette = "Spectral") +
  theme_bw() +
  facet_grid(~ Species)
```

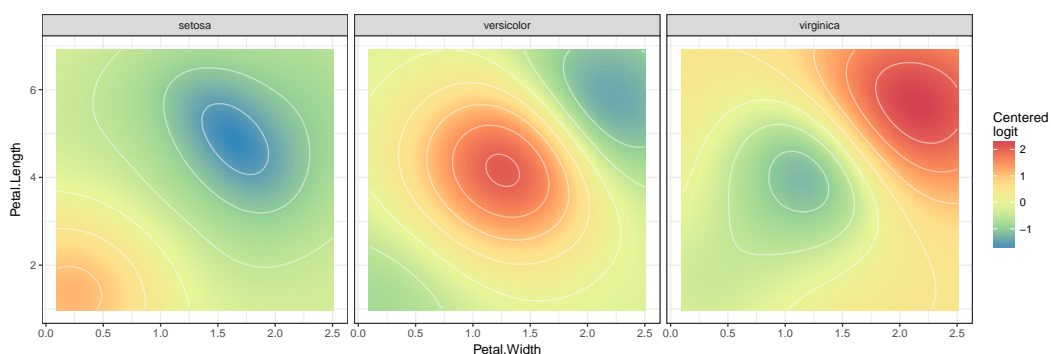


Figure 7: Partial dependence of `Species` on `Petal.Width` and `Petal.Length` for the iris data.

User-defined prediction functions

PDPs are essentially just averaged predictions; this is apparent from step 1. (c) in [Algorithm 1](#). Consequently, as pointed out by [Goldstein et al. \(2015\)](#), strong heterogeneity can conceal the complexity of the modeled relationship between the response and predictors of interest. This was part of the motivation behind [Goldstein, Kapelner, Bleich, and Pitkin's](#) ICE plot procedure.

With `partial` it is possible to replace the mean in step 1. (c) of Algorithm 1 with any other function (e.g., the median or trimmed mean), or obtain PDPs for classification problems on the probability scale. It is even possible to obtain ICE curves. This flexibility is due to the new `pred.fun` argument in `partial` (starting with `pdp` version 0.4.0). This argument accepts an optional prediction function that requires two arguments: `object` and `newdata`. The supplied prediction function must return either a single prediction or a vector of predictions. Returning the mean of all the predictions will result in the traditional PDP. Returning a vector of predictions (i.e., one for each observation) will result in a set of ICE curves. The examples below illustrate.

Using the `pred.fun` argument, it is possible to obtain PDPs for classification problems on the probability scale. We just need to write a function that computes the predicted class probability of interest averaged across all observations. The function below can be used with the fitted SVM from the iris example of Section X.2.5 to extract the average predicted probability of belonging to the Setosa class.

```
pred.prob <- function(object, newdata) { # see ?predict.svm
  pred <- predict(object, newdata, probability = TRUE)
  prob.setosa <- attr(pred, which = "probabilities")[, "setosa"]
  mean(prob.setosa)
}
```

Next, we simply pass this function via the `pred.fun` argument in the call to `partial`. The following chunk of code uses `pred.prob` to obtain PDPs for `Petal.Width` and `Petal.Length` on the probability scale. The results are displayed in Figure 8.

```
# PDPs for Petal.Width and Petal.Length on the probability scale
pdp.pw <- partial(iris.svm, pred.var = "Petal.Width", pred.fun = pred.prob,
  plot = TRUE)
pdp.pl <- partial(iris.svm, pred.var = "Petal.Length", pred.fun = pred.prob,
  plot = TRUE)
pdp.pw.pl <- partial(iris.svm, pred.var = c("Petal.Width", "Petal.Length"),
  pred.fun = pred.prob, plot = TRUE)
```

```
# Figure 8
grid.arrange(pdp.pw, pdp.pl, pdp.pw.pl, ncol = 3)
```

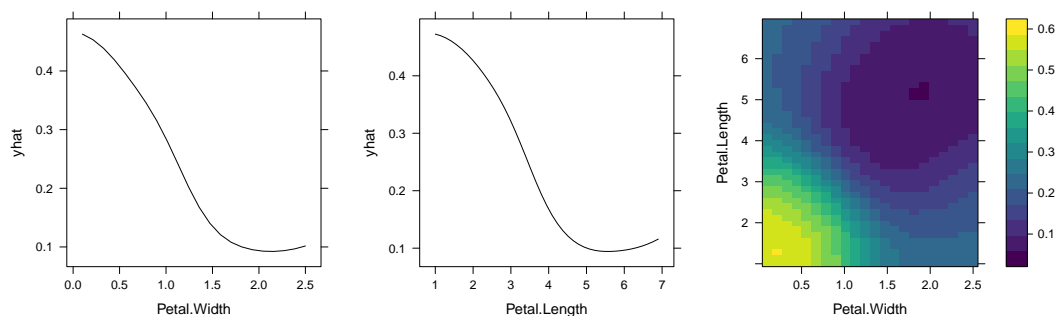


Figure 8: Partial dependence of Species on `Petal.Width` and `Petal.Length` plotted on the probability scale; in this case, the probability of belonging to the setosa species.

For regression problems, the default prediction function is essentially

```
pred.fun <- function(object, newdata) {
  mean(predict(object, newdata), na.rm = TRUE)
}
```

This corresponds to step 1. (c) in Algorithm 1. Suppose we would like ICE curves instead. To accomplish this we need to pass a prediction function that returns a vector of predictions, one for each observation in `newdata` (i.e., just remove the call to `mean` in `pred.fun`). The code snippet below illustrates this for the Boston housing example using the predictor `rm`. The result is displayed in Figure 9. **Note:** when the function supplied to `pred.fun` returns multiple predictions, the data frame returned by `partial` includes an additional column, `yhat.id`, that indicates which curve a point belongs to; in the following code chunk, there will be one curve for each observation in `boston`.

```
# Use partial to obtain ICE curves
pred.ice <- function(object, newdata) predict(object, newdata)
```

```
rm.ice <- partial(boston.rf, pred.var = "rm", pred.fun = pred.ice)

# Figure 9
plotPartial(rm.ice, rug = TRUE, train = boston, alpha = 0.3)
```

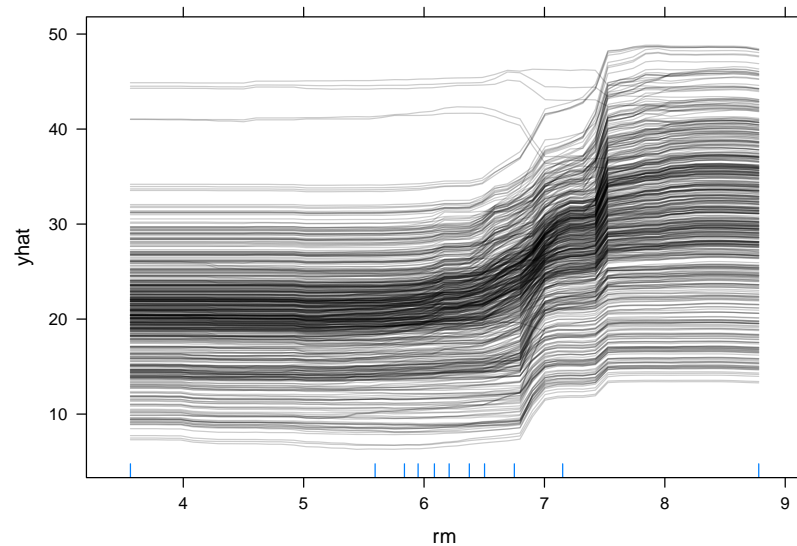


Figure 9: ICE curves depicting the relationship between `cmedv` and `rm` for the Boston housing example. Each curve corresponds to a different observation.

The curves in Figure 9 indicate some heterogeneity in the fitted model (i.e., some of the curves depict the opposite relationship). Such heterogeneity can be easier to spot using *c*-ICE curves; see Equation (4) on page 49 of Goldstein et al. (2015). Using `dplyr` (Wickham and Francois, 2016), it is rather straightforward to post-process the output from `partial` to obtain *c*-ICE curves (similar to the construction of *raw change scores* (Fitzmaurice et al., 2011, pg. 130) for longitudinal data). This is shown below.

```
# Post-process rm.ice to obtain c-ICE curves
library(dplyr) # for group_by and mutate functions
rm.ice <- rm.ice %>%
  group_by(yhat.id) %>% # perform next operation within each yhat.id
  mutate(yhat.centered = yhat - first(yhat)) # so each curve starts at yhat = 0
```

Since the PDP is just the average of the corresponding ICE curves, it is quite simple to display both on the same plot. This is easily accomplished using the `stat_summary` function from the `ggplot2` package to average the ICE curves together. The code snippet below plots the ICE curves and *c*-ICE curves, along with their averages, for the predictor `rm` in the Boston housing example. The results are displayed in Figure 10.

```
# ICE curves with their average
p1 <- ggplot(rm.ice, aes(rm, yhat)) +
  geom_line(aes(group = yhat.id), alpha = 0.2) +
  stat_summary(fun.y = mean, geom = "line", col = "red", size = 1)

# c-ICE curves with their average
p2 <- ggplot(rm.ice, aes(rm, yhat.centered)) +
  geom_line(aes(group = yhat.id), alpha = 0.2) +
  stat_summary(fun.y = mean, geom = "line", col = "red", size = 1)

# Figure 10
grid.arrange(p1, p2, ncol = 2)
```

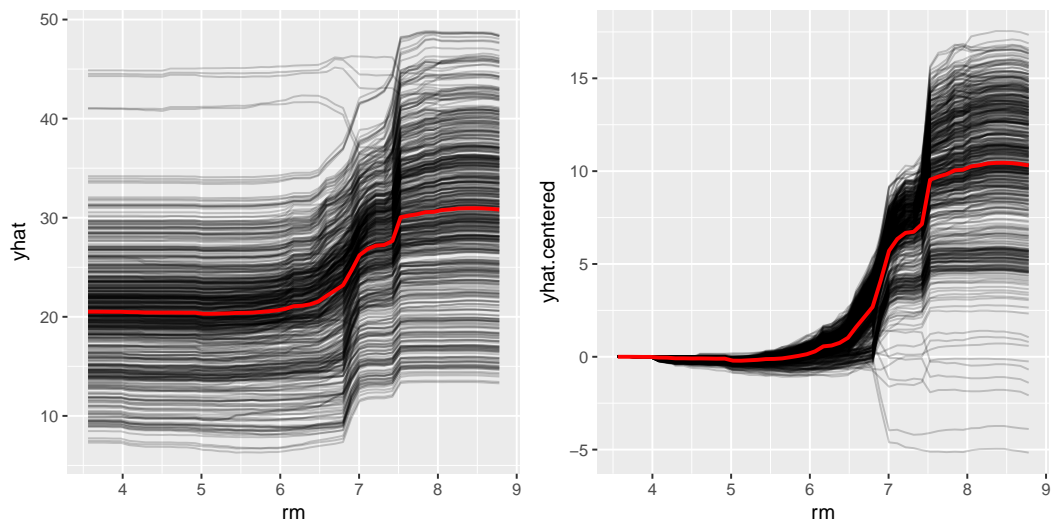



Figure 10: ICE curves (black curves) and their average (red curve) depicting the relationship between `cmdev` and `rm` for the Boston housing example. *Left:* Uncentered (here the red curve is just the traditional PDP). *Right:* Centered.

Using partial with the XGBoost library

To round out our discussion, we provide one last example using a recently popular (and successful!) machine learning tool. XGBoost, short for eXtreme Gradient Boosting, is a popular library providing optimized distributed gradient boosting that is specifically designed to be highly efficient, flexible and portable. The associated R package `xgboost` has been used to win a number of [Kaggle competitions](#). It has been shown to be many times faster than the well-known `gbm` package. However, unlike `gbm`, `xgboost` does not have built-in functions for constructing PDPs. Fortunately, the `pdp` package can be used to fill this gap.

For illustration, we return to the Boston housing example. The code chunk below uses `caret` to tune an `xgboost` model using 10-fold cross-validation. (After loading `caret`, use `getModelInfo("xgbTree")` for information on tuning `xgboost` models.) **Warning:** The following code chunk may take a few minutes to run.

```
# Tune an XGBoost model using 10-fold cross-validation
library(caret) # functions related to classification and regression training
set.seed(202) # for reproducibility
boston.xgb <- train(x = data.matrix(subset(boston, select = -cmdev)),
                  y = boston$cmdev, method = "xgbTree", metric = "Rsquared",
                  trControl = trainControl(method = "cv", number = 10),
                  tuneLength = 10)
```

The optimal model had a cross-validated R^2 of 0.902 (use `print(boston.xgb$bestTune)` to view the optimum tuning parameters). The next snippet of code computes the partial dependence of `cmdev` on both `rm` and `lstat`, individually and together. The results are displayed in Figure 11.

```
# PDPs for lstat and rm
pdp.lstat <- partial(boston.xgb, pred.var = "lstat", plot = TRUE, rug = TRUE)
pdp.rm <- partial(boston.xgb, pred.var = "rm", plot = TRUE, rug = TRUE)
pdp.lstat.rm <- partial(boston.xgb, pred.var = c("lstat", "rm"),
                      plot = TRUE, chull = TRUE)
```

```
# Figure 11
grid.arrange(pdp.lstat, pdp.rm, pdp.lstat.rm, ncol = 3)
```

The `train` function creates objects of class `"train"`, whereas the `xgboost` function creates objects of class `"xgb.Booster"`. Since `train` defaults to storing a copy of the training data as part of the `"train"` object, there is no need to supply it in the call to `partial` in this example. However, this is not the case when using the `xgboost` package directly. To illustrate, we fit the same model using the `xgboost` function with the optimum tuning parameters found previously using `caret`.

```
library(xgboost) # for xgboost function
set.seed(203) # for reproducibility
```

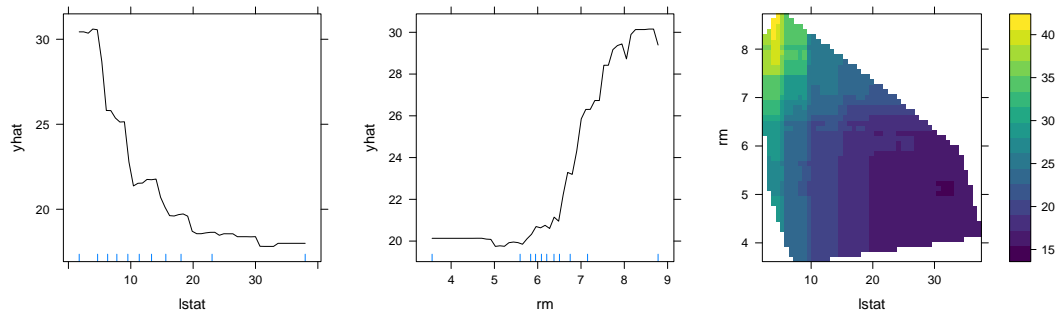


Figure 11: PDPs for the top two most important variables in the Boston housing data using **xgboost**. Compare this to the random forest results displayed in Figures 4-5.

```
boston.xgb <- xgboost(data = data.matrix(subset(boston, select = -cmedv)),
  label = boston$cmedv, objective = "reg:linear",
  nrounds = 100, max_depth = 5, eta = 0.3, gamma = 0,
  colsample_bytree = 0.8, min_child_weight = 1,
  subsample = 0.9444444)
```

To use `partial` with "xgb.Booster" objects, we need to supply the original training data (minus the response) in the call to `partial`. The following snippet of code computes the partial dependence of `cmedv` on `rm` (plot not shown). (Make sure you are using version 0.6-0 or later of **xgboost**: <https://github.com/dmlc/xgboost/tree/master/R-package>.) **Note:** while **xgboost** requires the training data to be an object of class "matrix", "dgCMatrix", or "xgb.DMatrix", `partial` requires a "data.frame" that does not contain the response column.

```
partial(boston.xgb, pred.var = "rm", plot = TRUE, rug = TRUE,
  train = subset(boston, select = -cmedv))
```

Summary

PDPs can be used to graphically examine the dependence of the response on low cardinality subsets of the features, accounting for the average effect of the other predictors. In this paper, we showed how to construct PDPs for various types of black box models in R using the **pdp** package. We also briefly discussed related approaches available in other R packages. Suggestions to avoid extrapolation and high execution times were discussed and demonstrated via examples.

This paper is based on **pdp** version 0.4.0. For updates that have occurred since then, see the package's [NEWS file](#). In terms of future development, **pdp** can be expanded in a number of ways. For example, it would be useful to have the ability to construct PDPs for black box survival models—like conditional random forests with censored response. It would also be worthwhile to implement the partial dependence-based *H*-statistic (Friedman and Popescu, 2008) for assessing the strength of interaction between predictors.

Acknowledgments

The author would like to thank two anonymous reviewers and the Editor for their helpful comments and suggestions.

Bibliography

- E. Alfaro, M. Gámez, and N. García. *adabag*: An R package for classification with boosting and bagging. *Journal of Statistical Software*, 54(2):1–35, 2013. URL <https://doi.org/10.18637/jss.v054.i02>. [p424]
- B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2016. URL <https://CRAN.R-project.org/package=gridExtra>. R package version 2.2.1. [p423]
- S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2014. URL <https://CRAN.R-project.org/package=magrittr>. R package version 1.5. [p426]

- L. Breiman and J. H. Friedman. Estimating optimal transformations for multiple regression and correlation. *Journal of the American Statistical Association*, 80(391):580–598, 1985. URL <https://doi.org/10.1080/01621459.1985.10478157>. [p428]
- G. M. Fitzmaurice, N. M. Laird, and J. H. Ware. *Applied Longitudinal Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2011. [p432]
- J. Fox. Effect displays in R for generalised linear models. *Journal of Statistical Software*, 8(15):1–27, 2003. URL <https://doi.org/10.18637/jss.v008.i15>. [p423]
- 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>. [p423]
- J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991. URL <https://doi.org/10.1214/aos/1176347963>. [p428]
- J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29:1189–1232, 2001. URL <https://doi.org/10.1214/aos/1013203451>. [p421, 428]
- J. H. Friedman and B. E. Popescu. Predictive learning via rule ensembles. *Annals of Applied Statistics*, 2(3):916–954, 2008. URL <https://doi.org/10.1214/07-aos148>. [p434]
- S. Garnier. *viridis: Default Color Maps from 'matplotlib'*, 2017. URL <https://CRAN.R-project.org/package=viridis>. R package version 0.4.0. [p426]
- A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015. URL <https://doi.org/10.1080/10618600.2014.907095>. [p422, 430, 432]
- B. Greenwell. *pdp: Partial Dependence Plots*, 2017. URL <https://CRAN.R-project.org/package=partial>. R package version 0.4.0. [p422]
- D. Harrison and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, 1978. URL [https://doi.org/10.1016/0095-0696\(78\)90006-2](https://doi.org/10.1016/0095-0696(78)90006-2). [p421, 425]
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer-Verlag, 2009. [p430]
- T. Hothorn and A. Zeileis. *partykit: A Laboratory for Recursive Partytioning*, 2016. URL <https://CRAN.R-project.org/package=partykit>. R package version 1.1-1. [p422]
- T. Hothorn, K. Hornik, C. Strobl, and A. Zeileis. *party: A Laboratory for Recursive Partytioning*, 2017. URL <https://CRAN.R-project.org/package=party>. R package version 1.2-3. [p422]
- J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. URL <https://doi.org/10.1109/mcse.2007.55>. [p426]
- A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. Kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004. URL <https://doi.org/10.18637/jss.v011.i09>. [p424]
- M. Kuhn. *caret: Classification and Regression Training*, 2017. URL <https://CRAN.R-project.org/package=caret>. R package version 6.0-76. [p424]
- M. Kuhn, S. Weston, N. Coulter, and M. Culp. *C50: C5.0 Decision Trees and Rule-Based Models*, 2015. URL <https://CRAN.R-project.org/package=C50>. R package version 0.1.0-24. [p424]
- M. Kuhn, S. Weston, C. Keefer, and N. Coulter. *Cubist: Rule- And Instance-Based Regression Modeling*, 2016. URL <https://CRAN.R-project.org/package=Cubist>. R package version 0.0.19. [p424]
- F. Leisch, K. Hornik, and B. D. Ripley. *mda: Mixture and Flexible Discriminant Analysis*, 2016. URL <https://CRAN.R-project.org/package=mda>. R package version 0.4-9. [p424]
- A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL <http://CRAN.R-project.org/doc/Rnews/>. [p422]
- D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2017. URL <https://CRAN.R-project.org/package=e1071>. R package version 1.6-8. [p424]

- S. Milborrow. *earth: Multivariate Adaptive Regression Splines*, 2017a. URL <https://CRAN.R-project.org/package=earth>. R package version 4.5.0. [p424]
- S. Milborrow. *plotmo: Plot a Model's Response and Residuals*, 2017b. URL <https://CRAN.R-project.org/package=plotmo>. R package version 3.3.3. [p422]
- A. Peters and T. Hothorn. *ipred: Improved Predictors*, 2017. URL <https://CRAN.R-project.org/package=ipred>. R package version 0.9-6. [p424]
- 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/>. [p423]
- Revolution Analytics and S. Weston. *doMC: Foreach Parallel Adaptor for 'parallel'*, 2015a. URL <https://CRAN.R-project.org/package=doMC>. R package version 1.3.4. [p428]
- Revolution Analytics and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2015b. URL <https://CRAN.R-project.org/package=doParallel>. R package version 1.0.10. [p428]
- Revolution Analytics and S. Weston. *foreach: Provides Foreach Looping Construct for R*, 2015c. URL <https://CRAN.R-project.org/package=foreach>. R package version 1.4.3. [p426]
- G. Ridgeway. *gbm: Generalized Boosted Regression Models*, 2017. URL <https://CRAN.R-project.org/package=gbm>. R package version 2.1.3. [p422]
- D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York, 2008. URL <http://lmdvr.r-forge.r-project.org>. ISBN 978-0-387-75968-5. [p422]
- D. Sarkar and F. Andrews. *latticeExtra: Extra Graphical Utilities Based on Lattice*, 2016. URL <https://CRAN.R-project.org/package=latticeExtra>. R package version 0.6-28. [p423]
- T. Therneau, B. Atkinson, and B. Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2017. URL <https://CRAN.R-project.org/package=rpart>. R package version 4.1-11. [p424]
- 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>. [p424]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. URL <http://ggplot2.org>. [p423]
- H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1): 1–29, 2011. URL <https://doi.org/10.18637/jss.v040.i01>. [p428]
- H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2016. URL <https://CRAN.R-project.org/package=dplyr>. R package version 0.5.0. [p432]
- M. N. Wright. *ranger: A Fast Implementation of Random Forests*, 2017. URL <https://CRAN.R-project.org/package=ranger>. R package version 0.7.0. [p424]

Brandon M. Greenwell
Infoscitex, a DCS Company
4027 Colonel Glenn Highway
Suite 210
Dayton, OH 45431-1672
United States of America
greenwell.brandon@gmail.com

checkmate: Fast Argument Checks for Defensive R Programming

by Michel Lang

Abstract Dynamically typed programming languages like R allow programmers to write generic, flexible and concise code and to interact with the language using an interactive Read-eval-print-loop (REPL). However, this flexibility has its price: As the R interpreter has no information about the expected variable type, many base functions automatically convert the input instead of raising an exception. Unfortunately, this frequently leads to runtime errors deeper down the call stack which obfuscates the original problem and renders debugging challenging. Even worse, unwanted conversions can remain undetected and skew or invalidate the results of a statistical analysis. As a resort, assertions can be employed to detect unexpected input during runtime and to signal understandable and traceable errors. The package **checkmate** provides a plethora of functions to check the type and related properties of the most frequently used R objects and variable types. The package is mostly written in C to avoid any unnecessary performance overhead. Thus, the programmer can conveniently write concise, well-tested assertions which outperforms custom R code for many applications. Furthermore, **checkmate** simplifies writing unit tests using the framework **testthat** (Wickham, 2011) by extending it with plenty of additional expectation functions, and registered C routines are available for package developers to perform assertions on arbitrary SEXP's (internal data structure for R objects implemented as struct in C) in compiled code.

Defensive programming in R

Most dynamic languages utilize a weak type system where the type of variable must not be declared, and R is no exception in this regard. On the one hand, a weak type system generally reduces the code base and encourages rapid prototyping of functions. On the other hand, in comparison to strongly typed languages like C/C++, errors in the program flow are much harder to detect. Without the type information, the R interpreter just relies on the called functions to handle their input in a meaningful way. Unfortunately, many of R's base functions are implemented with the REPL in mind. Thus, instead of raising an exception, many functions silently try to auto-convert the input. E.g., instead of assuming that the input NULL does not make sense for the function `mean()`, the value NA of type numeric is returned and additionally a warning message is signaled. While this behaviour is acceptable for interactive REPL usage where the user can directly react to the warning, it is highly unfavorable in packages or non-interactively executed scripts. As the generated missing value is passed to other functions deeper down the call stack, it will eventually raise an error. However, the error will be reported in a different context and associated with different functions and variable names. The link to origin of the problem is missing and debugging becomes much more challenging. Furthermore, the investigation of the call stack with tools like `traceback()` or `browser()` can result in an overwhelming number of steps and functions. As the auto-conversions cascade nearly unpredictably (as illustrated in Table 1), this may lead to undetected errors and thus to misinterpretation of the reported results.

Input	Return value of			
	mean(x)	median(x)	sin(x)	min(x)
numeric(0)	NaN	NA	numeric(0)	Inf (w)
character(0)	NA_real_ (w)	NA_character_	[exception]	NA_character_ (w)
NA	NA_real_	NA	NA_real_	NA_integer_
NA_character_	NA_real_ (w)	NA_character_	[exception]	NA_character_
NaN	NaN	NA	NaN	NaN
NULL	NA (w)	NULL (w)	[exception]	Inf (w)

Table 1: Input and output for some simple mathematical functions from the **base** package (R-3.4.0). Outputs marked with "(w)" have issued a warning message.

As a final motivating example, consider the following function which uses the base functions `diff()` and `range()` to calculate the range $r = x_{\max} - x_{\min}$ of a numerical input vector `x`:

```
myrange <- function(x) {
  diff(range(x))
}
```

This function is expected to return a single numerical value, e.g. `myrange(1:10)` returns 9. However, the calls `myrange(NULL)` and `myrange(integer(0))` also return such an unsuspecting, single numeric value: `-Inf`. Both calls signal warnings, but warnings can rather easily be overlooked and are hard to track down. It would arguably be much better to directly get an error message informing the user that something unintended is happening. In the worst case the warnings do not surface and – if this piece of code is embedded in a larger analysis – wrong conclusions are drawn.

The described problems lead to a concept called “defensive programming” where the programmer is responsible for manually checking function arguments. Reacting to unexpected input as soon as possible by signaling errors instantaneously with a helpful error message is the key aspect of this programming paradigm. A similar concept is called “design by contract” which demands the definition of formal, precise and verifiable input and in return guarantees a sane program flow if all preconditions hold. For `myrange()`, it would be necessary to insist that `x` is a numeric vector with at least one element to ensure a meaningful result. Additionally, missing values must be dealt with, either by completely prohibit them or by ensuring a meaningful return value. The package `checkmate` assists the programmer in writing such assertions in a concise way for the most important R variable types and objects. For `myrange()`, adding the line

```
assertNumeric(x, min.len = 1, any.missing = FALSE)
```

would be sufficient to ensure a sane program flow.

Related work

Many packages contain custom code to perform argument checks. These either rely on (a) the base function `stopifnot()` or (b) hand-written cascades of `if-else` blocks containing calls to `stop()`. Option (a) can be considered a quick hack because the raised error messages lack helpful details or instructions for the user. Option (b) is the natural way of doing argument checks in R but quickly becomes tedious. For this reason many packages have their own functions included, but there are also some packages on CRAN whose sole purpose are argument checks.

The package `assertthat` (Wickham, 2017) provides the “drop-in replacement” `assert_that()` for R’s `stopifnot()` while generating more informative help messages. This is achieved by evaluating the expression passed to the function `assert_that()` in an environment where functions and operators from the base package (e.g. `as.numeric()` or ``==``) are overloaded by more verbose counterparts. E.g., to check a variable to be suitable to pass to the `log()` function, one would require a numeric vector with all positive elements and no missing values:

```
assert_that(is.numeric(x), length(x) > 0, all(!is.na(x)), all(x >= 0))
```

For example, the output of the above statement for the input `c(1,NA,3)` reads “Error: Elements 2 of `!is.na(x)` are not true”. Additionally, `assertthat` offers some additional convenience functions like `is.flag()` to check for single logical values or `has_name()` to check for presence of specific names. These functions also prove useful if used with `see_if()` instead of `assert_that()` which turns the passed expression into a predicate function returning a logical value.

The package `assertive` (Cotton, 2016) is another popular package for argument checks. Its functionality is split over 16 packages containing over 400 functions, each specialized for a specific class of assertions: For instance, `assertive.numbers` specializes on checks of numbers and `assertive.sets` offers functions to work with sets. The functions are grouped into functions starting with `is_` for predicate functions and functions starting with `assert_` to perform `stopifnot()`-equivalent operations. The author provides a “checklist of checks” as package vignette to assist the user in picking the right functions for common situations like checks for numeric vectors or for working with files. Picking up the `log()` example again, the input check with `assertive` translates to:

```
assert_is_numeric(x)
assert_is_non_empty(x)
assert_all_are_not_na(x)
assert_all_are_greater_than_or_equal_to(x, 0)
```

The error message thrown by the first failing assertion for the input `c(1,NA,3)` reads “`is_not_na`: The values of `x` are sometimes NA.”. Additionally, a `data.frame` is printed giving detailed information about “bad values”:

```
There was 1 failure:
  Position Value Cause
1         2    NA missing
```

Moreover, the package **assertr** (Fischetti, 2016) focuses on assertions for **magrittr** (Bache and Wickham, 2014) pipelines and data frame operations in **dplyr** (Wickham and Francois, 2016), but is not intended for generic runtime assertions.

The checkmate package

Design goals

The package has been implemented with the following goals in mind:

Runtime To minimize any concern about the extra computation time required for assertions, most functions directly jump into compiled code to perform the assertions directly on the SEXP. The functions also have been extensively optimized to first perform inexpensive checks in order to be able to skip the more expensive ones.

Memory In many domains the user input can be rather large, e.g. long vectors and high dimensional matrices are common in text mining and bioinformatics. Basic checks, e.g. for missingness, are already quite time consuming, but if intermediate objects of the same dimension have to be created, runtimes easily get out of hand. For example, `any(x < 0)` with `x` being a large numeric matrix internally first allocates a logical matrix `tmp` with the same dimensions as `x`. The matrix `tmp` is then passed in a second step to `any()` which aggregates the logical matrix to a single logical value and `tmp` is marked to be garbage collected. Besides a possible shortage of available memory, which may cause the machine to swap or the R interpreter to terminate, runtime is wasted with unnecessary memory management. **checkmate** solves this problem by looping directly over the elements and thereby avoiding any intermediate objects.

Code completion The package aims to provide a single function for all frequently used R objects and their respective characteristics and attributes. This way, the built-in code completion of advanced R editors assist in finding the suitable further restrictions. For example, after typing the function name and providing the object to check (`assertNumeric(x, "`), many editors look up the function and suggest additional function arguments via code completion. In this example, checks to restrict the length, control the missingness or setting lower and upper bounds are suggested. These suggestions are restrictions on `x` specific for the respective base type (numeric). Such context-sensitive assistance helps writing more concise assertions.

The focus on runtime and memory comes at the price of error messages being less informative in comparison to **assertthat** or **assertive**. Picking up the previous example with input `c(1, NA, 3)`, **checkmate**'s `assertNumeric(x, any.missing = FALSE, lower = 0)` immediately raises an exception as soon as the NA is discovered at position 2. The package refrains from reporting an incomplete list of positions of missing values, instead the error message just reads "Assertion on 'x' failed: Contains missing values.". Thus, the implementations in **assertive** or **assertr** are better suited to find bad values in data and especially in data frames. **checkmate** is designed to amend functions with quick assertions in order to ensure a sane program flow. Nevertheless, the provided information is sufficient to quickly locate the error and start investigations with the debugging tools provided by R.

Naming scheme

The core functions of the package follow a specific naming scheme: The first part (prefix) of a function name determines the action to perform w.r.t. the outcome of the respective check while the second part of a function name (suffix) determines the base type of the object to check. The first argument of all functions is always the object `x` to check and further arguments specify additional restrictions on `x`.

Prefixes

There are currently four families of functions, grouped by their prefix, implemented in **checkmate**:

assert* Functions prefixed with "assert" throw an exception if the corresponding check fails and the checked object is returned invisibly on success. This family of functions is suitable for many different tasks. Besides argument checks of user input, this family of functions can also be used as a drop-in replacement for `stopifnot()` in unit tests using the internal test mechanism of R as described in Writing R Extensions (R Core Team, 2016), Subsection 1.1.5. Furthermore, as the object to check is returned invisibly, the functions can also be used inside **magrittr** pipelines.

test* Functions prefixed with "test" are predicate functions which return TRUE if the respective check is successful and FALSE otherwise. This family of functions is best utilized if different checks must be combined in a non-trivial manner or custom error messages are required.

expect* Functions prefixed with “expect” are intended to be used together with `testthat` (Wickham, 2011): the check is translated to an expectation which is then forwarded to the active `testthat` reporter. This way, `checkmate` extends the facilities of `testthat` with dozens of powerful helper functions to write efficient and comprehensive unit tests. Note that `testthat` is an optional dependency and the expect-functions only work if `testthat` is installed. Thus, to use `checkmate` as an `testthat` extension, `checkmate` must be listed in `Suggests` or `Imports` of a package.

check* Functions prefixed with “check” return the error message as a string if the respective check fails, and TRUE otherwise. Functions with this prefix are the workhorses called by the “assert”, “test” and “expect” families of functions and prove especially useful to implement custom assertions. They can also be used to collect error messages in order to generate reports of multiple check violations at once.

The prefix and the suffix can be combined in both “camelBack” and “underscore_case” fashion. In other words, `checkmate` offers all functions with the “assert”, “test” and “check” prefix in both programming style flavors: `assert_numeric()` is a synonym for `assertNumeric()` the same way `testDataFrame()` can be used instead of `test_data_frame()`. By supporting the two most predominant coding styles for R, most programmers can stick to their favorite style while implementing runtime assertions in their packages.

Suffixes

While the prefix determines the action to perform on a successful or failed check, the second part of each function name defines the base type of the first argument `x`, e.g. `integer`, `character` or `matrix`. Additional function arguments restrict the object to fulfill further properties or attributes.

Atomics and Vectors The most important built-in atomics are supported via the suffixes `*Logical`, `*Numeric`, `*Integer`, `*Complex`, `*Character`, `*Factor`, and `*List` (strictly speaking, “numeric” is not an atomic type but a naming convention for objects of type `integer` or `double`). Although most operations that work on real values also are applicable to natural numbers, the contrary is often not true. Therefore numeric values frequently need to be converted to integer, and `*Integerish` ensures a conversion without surprises by checking double values to be “nearby” an integer w.r.t. a machine-dependent tolerance. Furthermore, the object can be checked to be a vector, an atomic or an atomic vector (a vector, but not NULL).

All functions can optionally test for missing values (any or all missing), length (exact, minimum and maximum length) as well as names being (a) not present, (b) present and not NA/empty, (c) present, not NA/empty and unique, or (d) present, not NA/empty, unique and additionally complying to R’s variable naming scheme. There are more type-specific checks, e.g. bound checks for numerics or regular expression matching for characters. These are documented in full detail in the manual.

Scalars Atomics of length one are called scalars. Although R does not differentiate between scalars and vectors internally, scalars deserve particular attention in assertions as arguably most function arguments are expected to be scalar. Although scalars can also be checked with the functions that work on atomic vectors and additionally restricting to length 1 via argument `len`, `checkmate` provides some useful abbreviations: `*Flag` for logical scalars, `*Int` for an integerish value, `*Count` for a non-negative integerish values, `*Number` for numeric scalars and `*String` for scalar character vectors. Missing values are prohibited for all scalar values by default as scalars are usually not meant to hold data where missingness occurs naturally (but can be allowed explicitly via argument `na.ok`). Again, additional type-specific checks are available which are described in the manual.

Compound types The most important compound types are matrices/arrays (vectors of type logical, numeric or character with attribute `dim`) and data frames (lists with attribute `row.names` and class `data.frame` storing atomic vectors of same length). The package also includes checks for the popular data.frame alternatives `data.table` (Dowle et al., 2017) and `tibble` (Wickham et al., 2017). Some checkable characteristics conclude the internal type(s), missingness, dimensions or dimension names.

Miscellaneous On top of the already described checks, there are functions to work with sets (`*Subset`, `*Choice` and `*SetEqual`), environments (`*Environment`) and objects of class “Date” (`*Date`). The `*Function` family checks R functions and its arguments and `*OS` allows to check if R is running on a specific operating system. The functions `*File` and `*Directory` test for existence and access rights of files and directories, respectively. The function `*PathForOutput` allows to check whether a directory can be used to store files in it. Furthermore, `checkmate` provides functions to check the class or names of arbitrary R objects with `*Class` and `*Names`.

Custom checks Extensions are possible by writing a `check*` function which returns `TRUE` on success and an informative error message otherwise. The exported functionals `makeAssertionFunction()`, `makeTestFunction()` and `makeExpectationFunction()` can wrap this custom check function to create the required counterparts in such a way that they seamlessly fit into the package. The vignette demonstrates this with a check function for square matrices.

DSL for argument checks

Most basic checks can alternatively be performed using an implemented Domain Specific Language (DSL) via the functions `qassert()`, `qtest()` or `qexpect()`. All three functions have two arguments: The arbitrary object `x` to check and a “rule” which determines the checks to perform provided as a single string. Each rules consist of up to three parts:

1. The first character determines the expected class of `x`, e.g. “n” for numeric, “b” for boolean, “f” for a factor or “s” for a string (more can be looked up in the manual). By using a lowercase letter, missing values are permitted while an uppercase letter disallows missingness.
2. The second part is the length definition. Supported are “?” for length 0 or length 1, “+” for length ≥ 1 as well as arbitrary length specifications like “1”/“=1” for exact length 1 or “<10” for length < 10 .
3. The third part triggers a range check, if applicable, in interval notation (e.g., “[0,1)” for values $0 \leq x < 1$). If the boundary value on an open side of the interval is missing, all values of `x` will be checked for being $> -\infty$ or $< \infty$, respectively.

Although this syntax requires some time to familiarize with, it allows to write extensive argument checks with very few keystrokes. For example, the previous check for the input of `log()` translates to the rule “N+[0,]”. More examples can be found in Table 2

Base R	DSL
Single string <code>is.character(x) && length(x) == 1</code>	“s1”
Factor with minimum length 1, no missing values <code>is.factor(x) && length(x) >= 1 && all(!is.na(x))</code>	“F+”
List with no missing elements (NULL interpreted as missing for lists) <code>is.list(x) && !any(sapply(x, is.null))</code>	“L”
Integer of length 3 with positive elements <code>is.integer(x) && length(x) == 3 && all(x >= 0)</code>	“i3[0]”
Single number representing a proportion ($x \in [0,1]$) <code>is.numeric(x) && length(x) == 1 && !is.na(x) && x >= 0 && x <= 1</code>	“N1[0,1]”
Numeric vector with non-missing, positive, finite elements <code>is.numeric(x) && all(!is.na(x) & x >= 0 & is.finite(x))</code>	“N[0,)”
NULL or a single string with at least one character <code>is.null(x) (is.character(x) && length(x) == 1 && nzchar(x))</code>	“0”, “s1[1]”

Table 2: Exemplary checks using base R and the abbreviations implemented in the DSL.

As the function signature is really simplistic, it is perfectly suited to be used from compiled code written in C/C++ to check arbitrary SEXP. For this reason **checkmate** provides header files which third-party packages can link against. Instructions can be found in the package vignette.

Benchmarks

This small benchmark study picks up the `log()` example once again: testing a vector to be numeric with only positive, non-missing values.

Implementations

Now we compare **checkmate**’s `assertNumeric()` and `qassert()` (as briefly described in the previous Section [DSL for argument checks](#)) with counterparts written with R’s `stopifnot()`, **assertthat**’s `assert_that()` and a series of **assertive**’s `assert_*`() functions:

```

checkmate <- function(x) { assertNumeric(x, any.missing = FALSE, lower = 0) }
qcheckmate <- function(x) { qassert(x, "N[0,]") }
R <- function(x) { stopifnot(is.numeric(x), all(!is.na(x)), all(x >= 0)) }
assertthat <- function(x) { assert_that(is.numeric(x), all(!is.na(x)), all(x >= 0)) }
assertive <- function(x) { assert_is_numeric(x); assert_all_are_not_na(x);
  assert_all_are_greater_than_or_equal_to(x, 0) }

```

To allow measurement of failed assertions, the wrappers are additionally wrapped into a `try()` statement. Note that all functions perform the checks in the same order: First they check for type numeric, then for missing values and finally for all elements being non-negative. The source code for this benchmark study is hosted on **checkmate**'s project page in the directory `inst/benchmarks`.

Setup

The benchmark was performed on an Intel i5-6600 with 16 GB running R-3.4.0 on a 64bit Arch Linux installation using a 4.10.11 kernel. The package versions are 1.8.2 for **checkmate**, 0.2 for **assertthat** and 0.3.5 for **assertive**. R, the linked OpenBLAS and all packages have been compiled with the GNU Compiler Collection (GCC) in version 6.3.1 and tuned with `march=native` on optimization level -O2. To compare runtime differences, **microbenchmark** (Mersmann, 2015) is set to do 100 replications. The implemented wrappers have also been compared to their byte-compiled version (using `compiler::cmpfun`) with no notable difference in performance. The just-in-time compiler which is enabled per default as of R-3.4.0 causes a small but non-crucial decrease in performance for all implementations. The presented results are extracted from the uncompiled versions of these wrappers, with the JIT enabled on the default level 3.

Memory consumption is measured with the **benchexec** (Beyer et al., 2015) framework, version 1.10. Unlike R's `gc()` which only keeps track of allocations of SEXPs, **benchexec** measures all allocations (e.g., using C's `malloc`) and also works well with threads and child processes. Note that setting an upper memory limit is mandatory to ensure comparable measurements for memory consumption. Here, all processes are started with an upper limit of 2 GB.

Results

The benchmark is performed on four different inputs and the resulting timings are presented in Figure 1. Note that the runtimes on the x -axis are on \log_{10} -scale and use different units of measurement.

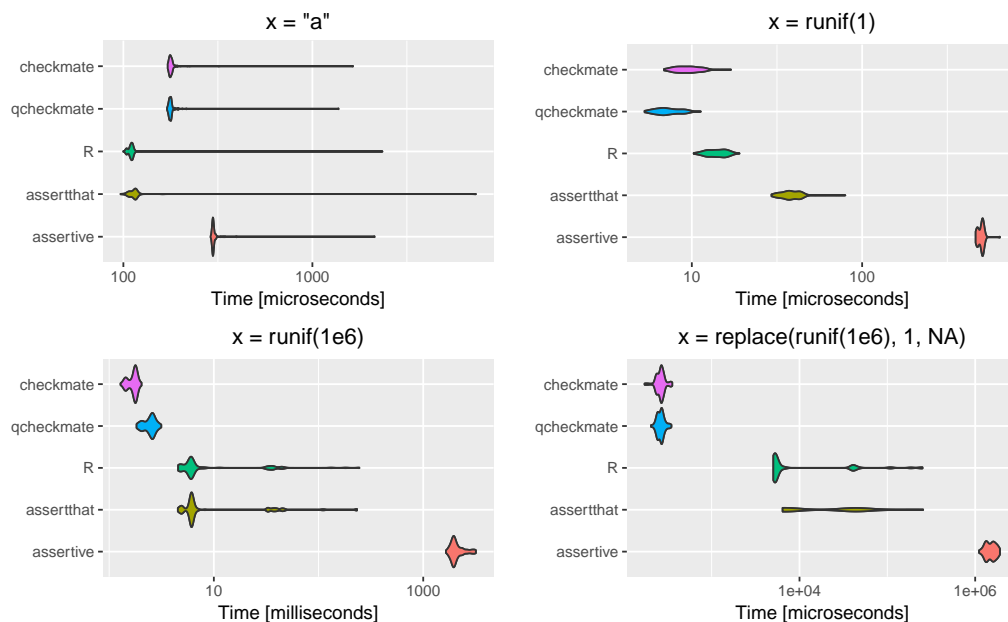


Figure 1: Violin plots of the runtimes on \log_{10} -scale of the assertion “ x must be a numeric vector with all elements positive and no missing values” on different input x .

top left Input x is a scalar character value, i.e. of wrong type. This benchmark serves as a measurement of overhead: the first performed (and cheapest) assertion on the type of x directly fails. In fact, all

assertion frameworks only require microseconds to terminate. R directly jumps into compiled code via a Primitive and therefore has the least overhead. **checkmate** on the other hand has to jump into the compiled code via the `.Call` interface which is comparably slower. The implementation in **assertthat** is faster than **checkmate** (as it also primarily calls primitives) but slightly slower than `stopifnot()`. The implementation in **assertive** is the slowest. However, in case of assertions (in comparison to tests returning logical values), the runtimes for a successful check are arguably more important than for a failed check because the latter raises an exception which usually is a rare event in the program flow and thus not time-critical. Therefore, the next benchmark might be more relevant for many applications.

top right Here, the input `x` is a valid numeric value. The implementations now additionally check for both missingness and negative values without raising an exception. The DSL variant `qassert()` is the fastest implementation, followed by **checkmate**'s `assertNumeric()`. Although `qassert()` and `assertNumeric()` basically call the same code internally, `qassert()` has less overhead due to its minimalist interface. R's `stopifnot()` is a tad slower (comparing the median runtimes) but still faster than **assertthat** (5x slowdown in comparison to `qassert()`). **assertive** is >70x slower than `qassert()`.

bottom left Input `x` is now a long vector with 10^6 numeric elements. This vector is generated using `runif()` and thus all values are valid (no negative or missing values). **checkmate** has the fastest versions with a speedup of approximately 3.5x compared to R's `stopifnot()` and `assert_that()`. In comparison to its alternatives, **checkmate** avoids intermediate objects as described in [Design goals](#): Instead of allocating a `logical(1e6)` vector first to aggregate it in a second step, **checkmate** directly operates on the numeric input. That is also the reason why `stopifnot()` and `assertthat()` have high variance in their runtimes: The garbage collector occasionally gets triggered to free memory which requires a substantial amount of time.

assertive is orders of magnitude slower for this input (>1200x) because it follows a completely different philosophy: Instead of focusing on speed, **assertive** gathers detailed information while performing the assertion. This yields report-like error messages (e.g., the index and reason why an assertion failed, for each element of the vector) but is comparably slow.

bottom right Input `x` is again a large vector, but the first element is a missing value. Here, all implementations first successfully check the type of `x` and then throw an error about the missing value at position 1. Again, **checkmate** avoids allocating intermediate objects which in this case yields an even bigger speedup: While the other packages first check 10^6 elements for missingness to create a `logical(1e6)` vector which is then passed to `any()`, **checkmate** directly stops after analyzing the first element of `x`. This obvious optimization yields a speedup of 25x in comparison to R and **assertthat** and a 7000x speedup in comparison to **assertive**.

Note that this is the best case scenario for early stopping to demonstrate the possible effect memory allocation can have on the runtime. Triggering the assertion on the last element of the vector results in runtimes roughly equivalent to the previous benchmark displayed at bottom left. A randomly placed bad value yields runtimes in the range of these two extremes.

Memory has been measured using `x = runif(1e7)` as input (similar to the setup of the benchmark shown in the bottom left of Figure 1). Measurements are repeated 100 times in independent calls of Rscript via the command line tool `runexec` and summarized by the mean and standard deviation. A no-operation (startup of R, creating the vector `x` and terminating) requires 105.0 ± 0.4 MB. The same script which additionally loads the **checkmate** package and performs the assertion with the above defined wrapper `checkmate()` does not increase the memory footprint (105.1 ± 0.2 MB) notably. Same for the previously defined wrapper `qcheckmate()` (105.0 ± 0.1 MB). The equivalent assertion using base R requires 185.1 ± 0.1 MB, about the same as the implementation in **assertthat** (185.1 ± 0.1 MB). Using **assertive**, 1601.8 ± 0.6 MB are required to run the script.

Summed up, **checkmate** is the fastest option to perform expensive checks and only causes a small decrease in performance for trivial, inexpensive checks which fail quickly (top left). Although the runtime differences seem insignificant for small input (top right), the saved microseconds can easily sum up to minutes or hours if the respective assertion is located in a hot spot of the program and therefore is called millions of times. By avoiding intermediate objects, assertions have virtually no memory overhead. This saves runtime in the garbage collection, and even becomes much more important as data grows bigger.

Conclusion

Runtime assertions are a necessity in R to ensure a sane program flow, but R itself offers very limited capabilities to perform these kind of checks. **checkmate** allows programmers and package developers to write assertions in a concise way without unnecessarily sacrificing runtime performance nor

increasing the memory footprint. Compared to the presented alternatives, assertions with **checkmate** are faster, tailored for bigger data and (with the help of code completion) more convenient to write. They generate helpful error messages, are extensively tested for correctness and suitable for large and extensive software projects (**mLr** (Bischl et al., 2016), **BatchJobs** (Bischl et al., 2015) and **batchtools** (Lang et al., 2017) already make heavy use of **checkmate**). Furthermore, **checkmate** offers capabilities to perform assertions on SEXP in compiled code via a domain specific language and extends the popular unit testing framework **testthat** with many helpful expectation functions.

Acknowledgments

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis”, project A3 (<http://sfb876.tu-dortmund.de>).

Bibliography

- S. M. Bache and H. Wickham. Magrittr: A Forward-Pipe Operator for R, 2014. URL <https://cran.r-project.org/package=magrittr>. [p439]
- D. Beyer, S. Löwe, and P. Wendler. Benchmarking and Resource Measurement. In *Model Checking Software*, pages 160–178. Springer-Verlag, 2015. URL https://doi.org/10.1007/978-3-319-23404-5_12. [p442]
- B. Bischl, M. Lang, O. Mersmann, J. Rahnenführer, and C. Weihs. BatchJobs and BatchExperiments: Abstraction Mechanisms for Using R in Batch Environments. *Journal of Statistical Software*, 64(11): 1–25, 2015. ISSN 1548-7660. URL <https://doi.org/10.18637/jss.v064.i11>. [p444]
- B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. MLr: Machine Learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL <http://www.jmlr.org/papers/v17/15-066.html>. [p444]
- R. Cotton. Assertive: Readable Check Functions to Ensure Code Integrity, 2016. URL <https://cran.r-project.org/package=assertive>. R package version 0.3-5. [p438]
- M. Dowle, A. Srinivasan, J. Gorecki, T. Short, S. Lianoglou, and E. Antonyan. Data.table: Extension of ‘data.frame’, 2017. URL <https://cran.r-project.org/package=data.table>. [p440]
- T. Fischetti. Assertive Programming for R Analysis Pipelines, 2016. URL <https://cran.r-project.org/package=assertr>. R package version 1.0.2. [p439]
- M. Lang, B. Bischl, and D. Surmann. Batchtools: Tools for R to work on batch systems. *The Journal of Open Source Software*, 2(10), 2017. ISSN 2475-9066. URL <https://doi.org/10.21105/joss.00135>. [p444]
- O. Mersmann. Microbenchmark: Accurate Timing Functions, 2015. URL <https://cran.r-project.org/package=microbenchmark>. R package version 1.4-2.1. [p442]
- R Core Team. Writing R Extensions, 2016. URL <https://cran.r-project.org/doc/manuals/r-release/R-exts.html>. [p439]
- H. Wickham. Testthat: Get Started with Testing. *The R Journal*, 3(1):5–10, 2011. URL https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf. [p437, 440]
- H. Wickham. Assertthat: Easy pre and post assertions, 2017. URL <https://cran.r-project.org/package=assertthat>. R package version 0.2. [p438]
- H. Wickham and R. Francois. Dplyr: A Grammar of Data Manipulation, 2016. URL <https://cran.r-project.org/package=dplyr>. R package version 0.5.0. [p439]
- H. Wickham, R. Francois, K. Müller, and RStudio. Tibble: Simple Data Frames, 2017. URL <https://cran.r-project.org/package=tibble>. [p440]

Michel Lang
TU Dortmund University, Faculty of Statistics
Vogelpothsweg 87, 44227 Dortmund

Germany
ORCID: 0000-0001-9754-0393
lang@statistik.tu-dortmund.de

milr: Multiple-Instance Logistic Regression with Lasso Penalty

by Ping-Yang Chen, Ching-Chuan Chen, Chun-Hao Yang, Sheng-Mao Chang, Kuo-Jung Lee

Abstract The purpose of the `milr` package is to analyze multiple-instance data. Ordinary multiple-instance data consists of many independent bags, and each bag is composed of several instances. The statuses of bags and instances are binary. Moreover, the statuses of instances are not observed, whereas the statuses of bags are observed. The functions in this package are applicable for analyzing multiple-instance data, simulating data via logistic regression, and selecting important covariates in the regression model. To this end, maximum likelihood estimation with an expectation-maximization algorithm is implemented for model estimation, and a lasso penalty added to the likelihood function is applied for variable selection. Additionally, an "milr" object is applicable to generic functions `fitted`, `predict` and `summary`. Simulated data and a real example are given to demonstrate the features of this package.

Introduction

Multiple-instance learning (MIL) is used to model the class labels which are associated with bags of observations instead of the individual observations. This technique has been widely used in solving many different real-world problems. In the early stage of the MIL application, [Dieterich et al. \(1997\)](#) studied the drug-activity prediction problem. A molecule is classified as a good drug if it is able to bind strongly to a binding site on the target molecule. The problem is: one molecule can adopt multiple shapes called the conformations and only one or a few conformations can bind the target molecule well. They described a molecule by a bag of its many possible conformations whose binding strength remains unknown. An important application of MIL is the image and text categorization, such as in [Maron and Ratan \(1998\)](#); [Andrews et al. \(2003\)](#); [Zhang et al. \(2007\)](#); [Zhou et al. \(2009\)](#); [Li et al. \(2011\)](#); [Kotzias et al. \(2015\)](#), to name a few. An image (bag) possessing at least one particular pattern (instance) is categorized into one class; otherwise, it is categorized into another class. For example, [Maron and Ratan \(1998\)](#) treated the natural scene images as bags, and, each bag is categorized as the scene of waterfall if at least one of its subimages is the waterfall. Whereas, [Zhou et al. \(2009\)](#) studied the categorization of collections (bags) of posts (instances) from different newsgroups corpus. A collection is a positive bag if it contains 3% posts from a target corpus category and the remaining 97% posts, as well as all posts in the negative bags, belong to the other corpus categories. MIL is also used in medical researches. The UCSB breast cancer study ([Kandemir et al., 2014](#)) is such a case. Patients (bags) were diagnosed as having or not having cancer by doctors; however, the computer, initially, had no knowledge of which patterns (instances) were associated with the disease. Furthermore, in manufacturing processes ([Chen et al., 2016](#)), a product (bag) is defective as long as one or more of its components (instances) are defective. In practice, at the initial stage, we only know that a product is defective, and we have no idea which component is responsible for the defect.

Several approaches have been offered to analyze datasets with multiple instances, e.g., [Maron \(1998\)](#); [Ray and Craven \(2005\)](#); [Xu and Frank \(2004\)](#); [Zhang and Goldman \(2002\)](#). From our point of view, the statuses of these components are missing variables, and thus, the Expectation-Maximization (EM) algorithm ([Dempster et al., 1977](#)) can play a role in multiple-instance learning. By now the toolboxes or libraries available for implementing MIL methods are developed by other computer softwares. For example, [Yang \(2008\)](#) and [Tax and Cheplygina \(2016\)](#) are implemented in MATLAB software, but neither of them carries the methods based on logistic regression model. [Settles et al. \(2008\)](#) provided the Java codes including the method introduced in [Ray and Craven \(2005\)](#). Thus, for R users, we are first to develop a MIL-related package based on logistic regression modelling which is called multiple-instance logistic regression (MILR). In this package, we first apply the logistic regression defined in [Ray and Craven \(2005\)](#) and [Xu and Frank \(2004\)](#), and then, we use the EM algorithm to obtain maximum likelihood estimates of the regression coefficients. In addition, the popular lasso penalty ([Tibshirani, 1996](#)) is applied to the likelihood function so that parameter estimation and variable selection can be performed simultaneously. This feature is especially desirable when the number of covariates is relatively large.

To fix ideas, we firstly define the notations and introduce the construction of the likelihood function. Suppose that the dataset consists of n bags and that there are m_i instances in the i th bag for $i = 1, \dots, n$. Let Z_i denote the status of the i th bag, and let Y_{ij} be the status of the j th instance in the i th bag along with $x_{ij} \in \mathbb{R}^p$ as the corresponding covariates. We assume that the Y_{ij} follow independent Bernoulli distributions with defect rates of p_{ij} , where $p_{ij} = g(\beta_0 + x_{ij}^T \beta)$ and $g(x) = 1 / (1 + e^{-x})$. We also

assume that the Z_i follow independent Bernoulli distributions with defect rates of π_i . Therefore, the bag-level likelihood function is

$$L(\beta_0, \beta) = \prod_{i=1}^n \pi_i^{z_i} (1 - \pi_i)^{1-z_i}. \tag{1}$$

To associate the bag-level defect rate π_i with the instance-level defect rates p_{ij} , several methods have been proposed. The bag-level status is defined as $Z_i = I(\sum_{j=1}^{m_i} Y_{ij} > 0)$. If the independence assumption among the Y_{ij} holds, the bag-level defect rate is $\pi_i = 1 - \prod_{j=1}^{m_i} (1 - p_{ij})$. On the other hand, if the independence assumption might not be held, [Xu and Frank \(2004\)](#) and [Ray and Craven \(2005\)](#) proposed the softmax function to associate π_i to p_{ij} , as follows:

$$s_i(\alpha) = \frac{\sum_{j=1}^{m_i} p_{ij} \exp\{\alpha p_{ij}\}}{\sum_{j=1}^{m_i} \exp\{\alpha p_{ij}\}}, \tag{2}$$

where α is a pre-specified nonnegative value. [Xu and Frank \(2004\)](#) used $\alpha = 0$, therein modeling π_i by taking the average of p_{ij} , $j = 1, \dots, m_i$, whereas [Ray and Craven \(2005\)](#) suggested $\alpha = 3$. We observe that the likelihood (1) applying neither the π_i function nor the $s_i(\alpha)$ function results in effective estimators.

Below, we begin by establishing the E-steps and M-steps required for the EM algorithm and then attach the lasso penalty for the estimation and feature selection. Several computation strategies applied are the same as those addressed in [Friedman et al. \(2010\)](#). Finally, we demonstrate the functions provided in the `milr` package via simulations and on a real dataset.

The multiple-instance logistic regression

EM algorithm

If the instance-level statuses, y_{ij} , are observable, the complete data likelihood is

$$\prod_{i=1}^n \prod_{j=1}^{m_i} p_{ij}^{y_{ij}} q_{ij}^{1-y_{ij}},$$

where $q_{ij} = 1 - p_{ij}$. An ordinary approach, such as the Newton method, can be used to solve this maximal likelihood estimate (MLE). However, considering multiple-instance data, we can only observe the statuses of the bags, $Z_i = I(\sum_{j=1}^{m_i} Y_{ij} > 0)$, and not the statuses of the instances Y_{ij} . As a result, we apply the EM algorithm to obtain the MLEs of the parameters by treating the instance-level labels as the missing data.

In the E-step, two conditional distributions of the missing data given the bag-level statuses Z_i are

$$Pr(Y_{i1} = 0, \dots, Y_{im_i} = 0 \mid Z_i = 0) = 1$$

and

$$Pr(Y_{ij} = y_{ij}, \quad j = 1, \dots, m_i \mid Z_i = 1) = \frac{\prod_{j=1}^{m_i} p_{ij}^{y_{ij}} q_{ij}^{1-y_{ij}} \times I(\sum_{j=1}^{m_i} y_{ij} > 0)}{1 - \prod_{l=1}^{m_i} q_{il}}.$$

Thus, the conditional expectations are

$$E(Y_{ij} \mid Z_i = 0) = 0 \quad \text{and} \quad E(Y_{ij} \mid Z_i = 1) = \frac{p_{ij}}{1 - \prod_{l=1}^{m_i} q_{il}} \equiv \gamma_{ij}.$$

The Q function at step t is $Q(\beta_0, \beta \mid \beta_0^t, \beta^t) = \sum_{i=1}^n Q_i(\beta_0, \beta \mid \beta_0^t, \beta^t)$, where Q_i is the conditional expectation of the complete log-likelihood for the i th bag given Z_i , which is defined as

$$\begin{aligned} Q_i(\beta_0, \beta \mid \beta_0^t, \beta^t) &= E \left(\sum_{j=1}^{m_i} y_{ij} \log(p_{ij}) + (1 - y_{ij}) \log(q_{ij}) \mid Z_i = z_i, \beta_0^t, \beta^t \right) \\ &= \sum_{j=1}^{m_i} z_i \gamma_{ij}^t (\beta_0 + x_{ij}^T \beta) - \log(1 + e^{\beta_0 + x_{ij}^T \beta}). \end{aligned}$$

Note that all the p_{ij} , q_{ij} , and γ_{ij} are functions of β_0 and β , and thus, we define these functions by substituting β_0 and β by their current estimates β_0^t and β^t to obtain p_{ij}^t , q_{ij}^t , and γ_{ij}^t , respectively.

In the M-step, we maximize this Q function with respect to (β_0, β) . Since the maximization of the nonlinear Q function is computationally expensive, following Friedman et al. (2010), the quadratic approximation to Q is applied. Taking the second-order Taylor expansion about β_0^t and β^t , we have $Q(\beta_0, \beta | \beta_0^t, \beta^t) = Q_Q(\beta_0, \beta | \beta_0^t, \beta^t) + C + R_2(\beta_0, \beta | \beta_0^t, \beta^t)$, where C is a constant in terms of β_0 and β , $R_2(\beta_0, \beta | \beta_0^t, \beta^t)$ is the remainder term of the expansion and

$$Q_Q(\beta_0, \beta | \beta_0^t, \beta^t) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^{m_i} w_{ij}^t [u_{ij}^t - \beta_0 - x_{ij}^T \beta]^2,$$

where $u_{ij}^t = \beta_0 + x_{ij}^T \beta^t + (z_i \gamma_{ij}^t - p_{ij}^t) / (p_{ij}^t q_{ij}^t)$ and $w_{ij}^t = p_{ij}^t q_{ij}^t$. In the `milr` package, instead of maximizing $Q(\beta_0, \beta | \beta_0^t, \beta^t)$, we maximize its quadratic approximation, $Q_Q(\beta_0, \beta | \beta_0^t, \beta^t)$. Since the objective function is quadratic, the roots of $\partial Q_Q / \partial \beta_0$ and $\partial Q_Q / \partial \beta$ have closed-form representations.

Variable selection with lasso penalty

We adopt the lasso method (Tibshirani, 1996) to identify active features in this MILR framework. The key is to add the L_1 penalty into the objective function in the M-step so that the EM algorithm is capable of performing estimation and variable selection simultaneously. To this end, we rewrite the objective function as

$$\min_{\beta_0, \beta} \left\{ -Q_Q(\beta_0, \beta | \beta_0^t, \beta^t) + \lambda \sum_{k=1}^p |\beta_k| \right\}. \tag{3}$$

Note that the intercept term β_0 is always kept in the model; thus, we do not place a penalty on β_0 . In addition, λ is the tuning parameter, and we will introduce how to determine this parameter later. We applied the shooting algorithm (Fu, 1998; Chen et al., 2016) to update (β_0^t, β^t) .

Implementation

The `milr` package contains a data generator, `DGP`, which is used to generate the multiple-instance data for the simulation studies, and two estimation approaches, `milr` and `softmax`, which are the main tools for modeling the multiple-instance data. In this section, we introduce the usage and default setups of these functions.

Data generator

The function `DGP` is the generator for the multiple-instance-type data under the MILR framework.

To use the `DGP` function, the user needs to specify an integer n as the number of bags, a vector m of length n as the number of instances in each bag, and a vector β of length p , with the desired number of covariates, and the regression coefficients, β , as in `DGP(n, m, beta)`. Note that one can set m as an integer for generating the data with an equal instance size m for each bag. Thus, the total number of observations is $N = \sum_{i=1}^n m_i$. The `DGP` simulates the labels of bags through the following steps:

1. Generate p mutually independent covariates of length N from the standard normal distribution as an $N \times p$ matrix, X .
2. Generate the binary response, Y_{ij} , for the j th instance of the i th bag from the Bernoulli distribution with

$$p_{ij} = 1 / \left(1 + \exp \left\{ -x_{ij}^T \beta \right\} \right)$$

where x_{ij} is the p -component vector in the row of X representing the j th instance of the i th bag.

3. Calculate the observed response for the i th bag by $Z_i = I \left(\sum_{j=1}^{m_i} Y_{ij} > 0 \right)$.
4. Return the indices of the bags, the covariate matrix X and the bag-level statuses Z .

The `milr` and `softmax` approaches

In the `milr` package, we provide two approaches to model the multiple-instance data: the proposed `milr` (Chen et al., 2016) and the `softmax` approach (Xu and Frank, 2004). To implement these two approaches, we assume that the number of observations and covariates are N and p , respectively. The

input data for both `milr` and `softmax` are separated into three parts: the bag-level statuses, y , as a vector of length N ; the $N \times p$ design matrix, x ; and `bag`, the vector of indices of length N , representing the indices of the bag to which each instance belongs.

```
milr(y, x, bag, lambda, numLambda, lambdaCriterion, nfold, maxit)
softmax(y, x, bag, alpha, ...)
```

For the `milr` function, specifying `lambda` in different ways controls whether and how the lasso penalty participates in parameter estimation. The default value of `lambda` is 0. With this value, the ordinary MLE is applied, i.e., no penalty term is considered. This is the suggested choice when the number of covariates p is small. When p is large or when variable selection is desired, users can specify a λ vector of length κ ; otherwise, by letting `lambda = -1`, the program automatically provides a λ vector of length $\kappa = \text{numLambda}$ as the tuning set. Following Friedman et al. (2010), the theoretical maximal value of λ in (3) is

$$\lambda_{max} = \left[\prod_{i=1}^n (m_i - 1) \right]^{\frac{1}{2}} \left[\prod_{i=1}^n m_i^{1-2z_i} \right]^{\frac{1}{2}}.$$

The automatically specified sequence of λ values ranges from $\lambda_{min} = \lambda_{max}/1000$ to λ_{max} in ascending order.

The default setting for choosing the optimal λ among these λ values is the Bayesian information criterion (BIC), $-2 \log(\text{likelihood}) + p^* \times \log(n)$, where p^* is the number of nonzero regression coefficients. Alternatively, the user can use the options `lambdaCriterion = "deviance"` and `nfold = K` with an integer K to obtain the best λ that minimizes the predictive deviance through "bag-wise" K -fold cross validation. The last option, `maxit`, indicates the maximal number of iterations of the EM algorithm; its default value is 500.

For the `softmax` function, the option `alpha` is a nonnegative real number for the α value in (2). The maximum likelihood estimators of the regression coefficients are obtained by the generic function `optim`. Note that no variable selection approach is implemented for this method.

Two generic accessory functions, `coef` and `fitted`, can be used to extract the regression coefficients and the fitted bag-level labels returned by `milr` and `softmax`. We also provide the significance test based on Wald's test for the `milr` estimations without the lasso penalty through the `summary` function. In addition, to predict the bag-level statuses for the new data set, the `predict` function can be used by assigning three items: `object` is the fitted model obtained by `milr` or `softmax`, `newdata` is the covariate matrix, and `bag_newdata` is the bag indices of the new dataset. Finally, the MIL model can be used to predict the bag-level labels and the instances-level labels. The option `type` in `fitted` and `predicted` functions controls the type of output labels. The default option is `type = "bag"` which results the bag-level prediction. Otherwise, by setting `type = "instance"`, the instances-level labels will be presented.

```
fitted(object, type)
predict(object, newdata, bag_newdata, type)
```

Examples

We illustrate the usage of the `milr` package via simulated and real examples.

Estimation and variable selection

We demonstrate how to apply the `milr` function for model estimation and variable selection. We simulate data with $n = 50$ bags, each containing $m = 3$ instances and regression coefficients $\beta = (-2, -1, 1, 2, 0.5, 0, 0, 0, 0)$. Specifically, the first four covariates are important.

```
library(magrittr)
library(milr)
set.seed(99)
# set the size of dataset
numOfBag <- 50
numOfInstsInBag <- 3
# set true coefficients: beta_0, beta_1, beta_2, beta_3
trueCoefs <- c(-2, -2, -1, 1, 2, 0.5, 0, 0, 0, 0)
trainData <- DGP(numOfBag, numOfInstsInBag, trueCoefs)
```

```
trainData$X %<>% set_colnames(paste0("X", 1:ncol(.)))
tapply(trainData$Z, trainData$ID, function(x) sum(x) > 0) %>% as.numeric

## [1] 1 1 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0
## [36] 1 1 0 0 0 1 1 0 0 0 1 1 0 1 1
```

Since the number of covariates is small, we then use the `milr` function to estimate the model parameters with $\lambda = 0$. One can apply `summary` to produce results including estimates of the regression coefficients and their corresponding standard error, testing statistics and the P-values under Wald's test. The regression coefficients are returned by the function `coef`.

```
# fit milr model
milrFit_EST <- milr(trainData$Z, trainData$X, trainData$ID, lambda = 0)
# call the Wald's test result
summary(milrFit_EST)

## Log-Likelihood: -14.005.

## Estimates:

##           Estimate Std.Err Z value Pr(>z)
## intercept -3.28671  1.16695 -2.8165 0.004855 **
## X1         -2.45529  0.92227 -2.6622 0.007762 **
## X2         -1.26351  0.67621 -1.8685 0.061689 .
## X3          0.94016  0.75173  1.2507 0.211054
## X4          3.84173  1.47862  2.5982 0.009372 **
## X5          0.22000  0.66579  0.3304 0.741073
## X6         -1.00740  0.73288 -1.3746 0.169262
## X7         -0.53063  0.59871 -0.8863 0.375463
## X8          0.25334  0.71596  0.3538 0.723451
## X9         -1.92753  0.92437 -2.0852 0.037047 *
## X10         0.12249  0.63054  0.1943 0.845972
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# call the regression coefficients
coef(milrFit_EST)

## intercept      X1      X2      X3      X4      X5
## -3.2867082 -2.4552903 -1.2635149  0.9401636  3.8417318  0.2199982
##          X6      X7      X8      X9      X10
## -1.0074012 -0.5306309  0.2533409 -1.9275338  0.1224893
```

The generic function `table` builds a contingency table of the counts for comparing the true bag-level statuses and the fitted bag-level statuses (obtained by the option `type = "bag"`) and the `predict` function is used to predict the labels of each bag with corresponding covariate X . On the other hand, The fitted and predicted instance-level statuses can also be found by setting `type = "instance"` in the fitted and predict functions.

```
fitted(milrFit_EST, type = "bag")

##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
##  1  1  0  1  1  1  0  1  1  0  1  1  0  0  1  1  0  1  1  1  1  0  0  1  0
## 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
##  0  1  0  1  0  1  1  1  0  0  1  1  0  0  0  1  1  1  0  0  0  1  0  1  1

%#fitted(milrFit_EST, type = "instance") # instance-level fitted labels
table(DATA = tapply(trainData$Z, trainData$ID, function(x) sum(x) > 0) %>% as.numeric,
      FITTED = fitted(milrFit_EST, type = "bag"))

##      FITTED
## DATA  0  1
##      0 18  4
##      1  3 25

# predict for testing data
testData <- DGP(numOfBag, numOfInstsInBag, trueCoefs)
```

```

testData$X %<>% set_colnames(paste0("X", 1:ncol(.)))
pred_EST <- predict(milrFit_EST, testData$X, testData$ID, type = "bag")
#predict(milrFit_EST, testData$X, testData$ID,
#       type = "instance") # instance-level prediction
table(DATA = tapply(testData$Z, testData$ID, function(x) sum(x) > 0) %>% as.numeric,
       PRED = pred_EST)

##      PRED
## DATA 0  1
##      0 13  6
##      1  8 23

```

Next, the $n < p$ cases are considered. We generate a data set with $n = 50$ bags, each with 3 instances and $p = 100$ covariates. Among these covariates, only the first five of them, X_1, \dots, X_5 , are active and their nonzero coefficients are the same as the previous example. First, we manually specify 50 λ values manually from 0.01 to 50. The `milr` function chooses the best tuning parameter which results in the smallest BIC. For this dataset, the chosen model is a constant model.

```

set.seed(99)
# Set the new coefficient vector (large p)
trueCoefs_Lp <- c(-2, -2, -1, 1, 2, 0.5, rep(0, 95))
# Generate the new training data with large p
trainData_Lp <- DGP(numOfBag, numOfInstsInBag, trueCoefs_Lp)
trainData_Lp$X %<>% set_colnames(paste0("X", 1:ncol(.)))
# variable selection by user-defined tuning set
lambdaSet <- exp(seq(log(0.01), log(50), length = 50))
milrFit_VS <- milr(trainData_Lp$Z, trainData_Lp$X, trainData_Lp$ID,
                  lambda = lambdaSet)
# grep the active factors and their corresponding coefficients
coef(milrFit_VS) %>% .[abs(.) > 0]

## intercept
## -0.9020893

```

Second, we try the auto-tuning feature implemented in `milr` by assigning `lambda = -1`. The total number of tuning λ values is indicated by setting `numLambda`. The following example shows the result of the best model chosen among 50 λ values. The slice `$lambda` shows the auto-tuned λ candidates and the slice `$BIC` returns the corresponding value of BIC for every candidate λ value. Again, the chosen model is a constant model.

```

# variable selection using auto-tuning
milrFit_auto_VS <- milr(trainData_Lp$Z, trainData_Lp$X, trainData_Lp$ID,
                       lambda = -1, numLambda = 50)
# the auto-selected lambda values
milrFit_auto_VS$lambda

## [1] 0.08041559 0.09259014 0.10660786 0.12274780 0.14133125
## [6] 0.16272815 0.18736444 0.21573056 0.24839117 0.28599645
## [11] 0.32929500 0.37914875 0.43655012 0.50264180 0.57873946
## [16] 0.66635795 0.76724148 0.88339831 1.01714075 1.17113118
## [21] 1.34843505 1.55258192 1.78763568 2.05827549 2.36988893
## [26] 2.72867921 3.14178869 3.61744105 4.16510498 4.79568271
## [31] 5.52172701 6.35769107 7.32021625 8.42846331 9.70449388
## [36] 11.17370961 12.86535784 14.81311383 17.05575111 19.63791336
## [41] 22.61100310 26.03420494 29.97566379 34.51384138 39.73907818
## [46] 45.75539179 52.68254760 60.65844293 69.84185212 80.41558721

# the values of BIC under each lambda value
milrFit_auto_VS$BIC

## [1] 196.54069 184.90606 161.51628 157.75005 118.76827 118.95214 115.27610
## [8] 115.55212 115.87195 112.32892 112.75439 113.24620 113.81466 114.48027
## [15] 119.19304 116.18877 121.15644 114.54700 112.02964 110.31200 103.92031
## [22] 101.90760 88.78977 83.57070 82.69910 82.13240 78.00261 74.25620
## [29] 74.19599 78.98716 77.39963 75.33387 78.16814 69.25384 69.25384
## [36] 69.25384 69.25384 69.25384 69.25384 69.25384 69.25384 69.25384
## [43] 69.25384 69.25384 69.25384 69.25384 69.25384 69.25384 69.25384
## [50] 69.25384

```

```
# grep the active factors and their corresponding coefficients
coef(milrFit_auto_VS) %>% .[abs(.) > 0]

## intercept
## -0.9020893
```

Instead of using BIC, a better way to choose the proper λ is using the cross validation by setting `lambdaCriterion = "deviance"`. The following example shows the best model chosen by minimizing the predictive deviance via “bag-wise” 10-fold cross validation. The results of the predictive deviance for every candidate λ can be found in the slice `$cv`. Twenty-nine covariates were identified including the first four true active covariates, X_1, \dots, X_4 .

```
# variable selection using auto-tuning with cross validation
milrFit_auto_CV <- milr(trainData_Lp$Z, trainData_Lp$X, trainData_Lp$ID,
  lambda = -1, numLambda = 50,
  lambdaCriterion = "deviance", nfold = 10)
# the values of predictive deviance under each lambda value
milrFit_auto_CV$cv

## [1] 10.013948  3.754961  3.132322  2.933881  2.433803  2.346058  2.752407
## [8]  3.248528  3.858600  4.392568  4.781208  5.249175  5.727995  6.030227
## [15]  6.393522  6.432488  6.379543  6.339838  6.317661  6.329531  5.551296
## [22]  5.222904  5.113070  5.006837  5.078377  5.106067  5.242165  5.579102
## [29]  5.786248  6.178347  6.414204  6.648448  6.659413  6.573462  6.547737
## [36]  6.547737  6.547737  6.547737  6.547737  6.547737  6.547737  6.547737
## [43]  6.547737  6.547737  6.547737  6.547737  6.547737  6.547737  6.547737
## [50]  6.547737

# grep the active factors and their corresponding coefficients
coef(milrFit_auto_CV) %>% .[abs(.) > 0]

## intercept          X1          X2          X3          X4
## -2.446119887 -0.362833108 -1.479388087  0.541861054  0.535400264
##          X7          X11          X14          X15          X17
##  1.448461978  0.334921736  0.004238594 -0.755908930  0.017708059
##          X18          X25          X26          X30          X32
## -0.586349577 -0.244962971  0.343205919  1.315468844 -0.845118964
##          X33          X37          X48          X58          X61
##  0.370261921 -0.493144745 -0.523001848 -0.044975426  0.208521105
##          X62          X71          X72          X74          X76
##  0.409946699  1.369814722  0.484713157  0.683531448  1.542186462
##          X77          X79          X85          X95          X100
## -0.656669320 -1.685794976 -0.369189815 -0.912145167 -0.135461219
```

According to another simulation study which is not shown in this paper, in contrast to cross-validation, BIC does not perform well for variable selection in terms of multiple-instance logistic regressions. However, it can be an alternative when performing cross-validation is too time consuming.

Real case study

Hereafter, we denote the proposed method with the lasso penalty by MILR-LASSO for brevity. In the following, we demonstrate the usage of MILR-LASSO and the `softmax` approach on a real dataset, called MUSK1. The MUSK1 data set consists of 92 molecules (bags) of which 47 are classified as having a musky smell and 45 are classified to be non-musks. The molecules are musky if at least one of their conformers (instances) were responsible for the musky smell. However, knowledge about which conformers are responsible for the musky smell is unknown. There are 166 features that describe the shape, or conformation, of the molecules. The goal is to predict whether a new molecules is musk or non-musk. This dataset is one of the popular benchmark datasets in the field of multiple-instance learning research and one can download the dataset from the following weblink.

```
dataName <- "MIL-Data-2002-Musk-Core1-Trec9.tgz"
dataUrl <- "http://www.cs.columbia.edu/~andrews/mil/data/"
```

We use the `untar` function to decompress the downloaded `.tgz` file and extract the MUSK1 dataset. Then, with the following data preprocessing, we reassemble the MUSK1 dataset in a “`data.frame`” format. The first 2 columns of the MUSK1 dataset are the bag indices and the bag-level labels of each observation. Starting with the third column, there are $p = 166$ covariates involved in the MUSK1 dataset.

```

filePath <- file.path(getwd(), dataName)
# Download MIL data sets from the url
download.file(paste0(dataUrl, dataName), filePath)
# Extract MUSK1 data file
untar(filePath, files = "MilData/Musk/musk1norm.svm")
# Read and Preprocess MUSK1
library(reshape2)
tmp <- read.table(file.path(getwd(), "MilData/Musk/musk1norm.svm"),
                  sep = " ", colClasses = "character")
MUSK1 <- colsplit(tmp[,1], ":", names = c("obs", "bag", "label"))[,2:3]
MUSK1 <- cbind(MUSK1, Reduce(cbind,
                            lapply(2:ncol(tmp),
                                    function(i) colsplit(tmp[,i], ":", names = paste0(c("num", "x"), i-1))[,2]
                                )))
MUSK1$bag <- MUSK1$bag + 1
MUSK1$label <- (MUSK1$label + 1)/2
MUSK1[,3:ncol(MUSK1)] <- scale(MUSK1[,3:ncol(MUSK1)])
Y <- tapply(MUSK1$label, MUSK1$bag, function(x) sum(x) > 0) %>% as.numeric
nc <- ncol(MUSK1)

```

To fit an MIL model without variable selection, the `milr` package provides two functions. The first is the `milr` function with `lambda = 0`. The second approach is the `softmax` function with a specific value of `alpha`. Here, we apply the approaches that have been introduced in [Xu and Frank \(2004\)](#) and [Ray and Craven \(2005\)](#), called the $s(0)$ ($\alpha=0$) and $s(3)$ ($\alpha=3$) methods, respectively. The optimization method in `softmax` is chosen as the default settings of the generic function `optim`, that is, the *Nelder-Mead* method.

As suggested by one reviewer, it is relevant to compare the computational efficiencies and convergence rates of the `milr` and `softmax` functions implemented in this package. Note that, the `milr` approach is written in C++ and so is the objective function in `softmax`, and, we only consider their performance affected by their common tuning parameter, `maxit`, the total number of iterations. For each approach, the total number of iterations are set from 5,000 to 25,000, and, the computation task was performed by a laptop with Intel Core M-5Y71 CPU 1.4 GHz and 8GB RAM. Moreover, the performance in model fitting is assessed based on the classification accuracy. We use the generic function `table` to produce the contingency tables and calculate the classification accuracy values accordingly.

The left panel of Figure 1 shows the computational cost of each approach along with the increment of the total number of iterations. As expected, the computational cost increases with the number of iterations linearly for both functions. However, the slope for the `milr` function is much flatter than the slope for the `softmax` function. A further result of MILR not shown here suggests that, for this dataset, the coefficient estimate of the MILR approach converges between 15,000 and 16,000 iterations. The resulting accuracy of each model is shown in the right panel of Figure 1 which indicates that the MILR approach requires fewer iterations to achieve the best fit.

```

# set the iterations from 5000 to 25000
itSet <- seq(5000, 25000, 2000)
runtime <- matrix(0, length(itSet), 3)
runacc <- matrix(0, length(itSet), 3)
for (it in 1:length(itSet)) {
  # record the computation time
  runtime[it,1] <- system.time(
    softmaxFit_0 <- softmax(MUSK1$label, MUSK1[,3:nc], MUSK1$bag, alpha = 0,
                          control = list(maxit = itSet[it]))
  )[3]
  runtime[it,2] <- system.time(
    softmaxFit_3 <- softmax(MUSK1$label, MUSK1[,3:nc], MUSK1$bag, alpha = 3,
                          control = list(maxit = itSet[it]))
  )[3]
  runtime[it,3] <- system.time(
    # use a very small lambda so that milr can do the estimation
    # without evaluating the Hessian matrix
    milrFit <- milr(MUSK1$label, MUSK1[,3:nc], MUSK1$bag, lambda = 1e-7,
                  maxit = itSet[it])
  )[3]
  # calculate the accuracy

```

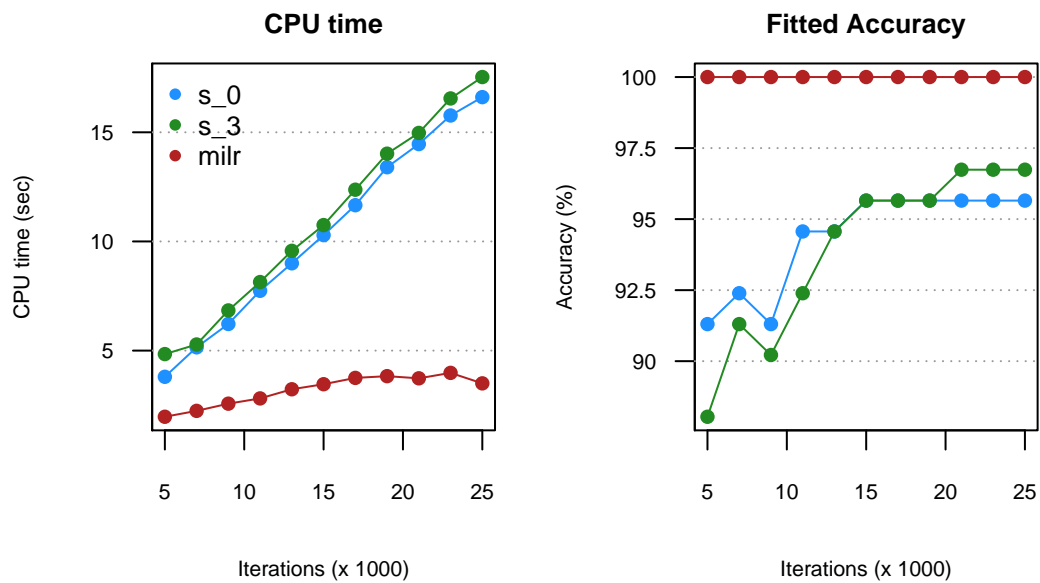


Figure 1: Computational efficiency of softmax methods and milr approach.

```

tmp <- table(DATA = Y, FIT_s0 = fitted(softmaxFit_0, type = "bag"))
runacc[it,1] <- sum(diag(tmp))/sum(tmp)
tmp <- table(DATA = Y, FIT_s3 = fitted(softmaxFit_3, type = "bag"))
runacc[it,2] <- sum(diag(tmp))/sum(tmp)
tmp <- table(DATA = Y, FIT_MILR = fitted(milrFit, type = "bag"))
runacc[it,3] <- sum(diag(tmp))/sum(tmp)
}

```

For variable selection, we apply the MILR-LASSO approach. First, the tuning parameter set is chosen automatically by setting $\lambda = -1$, and the best λ value is obtained by minimizing the predictive deviance with 10-fold cross validation among $n\lambda = 100$ candidates. In total it costs about 130 seconds to choose the optimal λ value and there are 19 active covariates detected by the MILR-LASSO approach. Using these active covariates, the reduced MILR model performs 89.13% classification accuracy.

```

# MILR-LASSO
milrSV <- milr(MUSK1$label, MUSK1[,3:nc], MUSK1$bag,
              lambda = -1, numLambda = 100,
              lambdaCriterion = "deviance", maxit = 16000)
sv_ind <- which(coef(milrSV)[-1] != 0) + 2
# show the detected active covariates
names(MUSK1)[sv_ind]

## [1] "V31" "V36" "V37" "V76" "V83" "V105" "V106" "V108" "V109" "V116"
## [11] "V118" "V124" "V126" "V129" "V132" "V136" "V147" "V162" "V163"

# use a very small lambda so that milr can do the estimation
# without evaluating the Hessian matrix
milrREFit <- milr(MUSK1$label, MUSK1[,sv_ind], MUSK1$bag,
                 lambda = 1e-7, maxit = 16000)
table(DATA = Y, FIT_MILR = fitted(milrREFit, type = "bag"))

##      FIT_MILR
## DATA  0  1
##      0 39  6
##      1  4 43

```

Following the discussion above, we use 10-fold cross validation and compare the prediction accuracy among four MIL models which are $s(0)$, $s(3)$, the MILR model with all covariates, and, the MILR model fitted by the selected covariates via MILR-LASSO. The resulting prediction accuracies are 83.70%, 77.17%, 75.00% and 81.52%, respectively.

```

predY <- matrix(0, length(Y), 4); colnames(predY) <- c("s0", "s3", "milr", "milr_sv")
set.seed(99)
folds <- 10; foldSize <- floor(length(Y)/folds)
foldBag <- c(rep(1:folds, foldSize), sample(1:folds, length(Y) - folds*foldSize))
foldBag <- sample(foldBag, length(foldBag))
foldIns <- rep(foldBag, table(MUSK1$bag))
for (i in 1:folds) {
  # prepare training and testing sets
  ind <- which(foldIns == i)
  training <- MUSK1[-ind,]; testing <- MUSK1[ind,]
  # train models
  fit_s0 <- softmax(training$label, training[,3:nc], training$bag,
                    alpha = 0, control = list(maxit = 25000))
  fit_s3 <- softmax(training$label, training[,3:nc], training$bag,
                    alpha = 3, control = list(maxit = 25000))
  # milr, use a very small lambda so that milr do the estimation
  # without evaluating the Hessian matrix
  fit_milr <- milr(training$label, training[,3:nc], training$bag,
                  lambda = 1e-7, maxit = 16000)
  fit_milr_sv <- milr(training$label, training[,sv_ind], training$bag,
                     lambda = 1e-7, maxit = 16000)
  # store the predicted labels
  ind2 <- which(foldBag == i)
  # predict function returns bag response in default
  predY[ind2,1] <- predict(fit_s0, as.matrix(testing[,3:nc]), testing$bag)
  predY[ind2,2] <- predict(fit_s3, as.matrix(testing[,3:nc]), testing$bag)
  predY[ind2,3] <- predict(fit_milr, as.matrix(testing[,3:nc]), testing$bag)
  predY[ind2,4] <- predict(fit_milr_sv, as.matrix(testing[,sv_ind]), testing$bag)
}

table(DATA = Y, PRED_s0 = predY[,1])

##      PRED_s0
## DATA  0  1
##      0 36  9
##      1  6 41

table(DATA = Y, PRED_s3 = predY[,2])

##      PRED_s3
## DATA  0  1
##      0 28 17
##      1  4 43

table(DATA = Y, PRED_MILR = predY[,3])

##      PRED_MILR
## DATA  0  1
##      0 32 13
##      1 10 37

table(DATA = Y, PRED_MILR_SV = predY[,4])

##      PRED_MILR_SV
## DATA  0  1
##      0 35 10
##      1  7 40

```

Summary

This article introduces the usage of the R package **milr** for analyzing multiple-instance data under the framework of logistic regression. In particular, the package contains two approaches: summarizing

the mean responses within each bag using the softmax function (Xu and Frank, 2004; Ray and Craven, 2005) and treating the instance-level statuses as hidden information as well as applying the EM algorithm for estimation (Chen et al., 2016). In addition, to estimate the MILR model, a lasso-type variable selection technique is incorporated into the latter approach. The limitations of the developed approaches are as follows. First, we ignore the potential dependency among instance statuses within one bag. Random effects can be incorporated into the proposed logistic regression to represent the dependency. Second, according to our preliminary simulation study, not shown in this paper, the maximum likelihood estimator might be biased when the number of instances in a bag is large, say, $m_i = 100$ or more. Bias reduction methods, such as Firth (1993) and Quenouille (1956), can be applied to alleviate this bias. These attempts are deferred to our future work.

Bibliography

- S. Andrews, I. Tsochantaris, and T. Hofmann. Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems 15*, pages 561–568, 2003. [p446]
- R.-B. Chen, K.-H. Cheng, S.-M. Chang, S.-L. Jeng, P.-Y. Chen, C.-H. Yang, and C.-C. Hsia. Multiple-instance logistic regression with lasso penalty. *arXiv preprint arXiv:1607.03615*, 2016. [p446, 448, 456]
- A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39(1):1–38, 1977. [p446]
- T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1):31–71, 1997. [p446]
- D. Firth. Bias reduction of maximum likelihood estimates. *Biometrika*, 80(1):27–38, 1993. [p456]
- J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. [p447, 448, 449]
- W. J. Fu. Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998. [p448]
- M. Kandemir, C. Zhang, and H. F. A. Empowering multiple instance histopathology cancer diagnosis by cell graphs. In *Medical Image Computing and Computer Assisted Intervention 17 (Pt 2)*, pages 228–235, 2014. [p446]
- D. Kotzias, M. Denil, N. de Freitas, and P. Smyth. From group to individual labels using deep features. In *ACM SigKDD International Conference on Knowledge Discovery and Data Mining 21*, pages 597–606, 2015. [p446]
- W. Li, L. Duan, D. Xu, and I. W.-H. Tsang. Text-based image retrieval using progressive multi-instance learning. In *International Conference on Computer Vision '11*, pages 2049–2055, 2011. [p446]
- O. Maron. *Learning from Ambiguity*. PhD thesis, Massachusetts Institute of Technology, 1998. [p446]
- O. Maron and A. L. Ratan. Multiple-instance learning for natural scene classification. In *International Conference on Machine Learning 15*, pages 341–349, 1998. [p446]
- M. H. Quenouille. Notes on bias in estimation. *Biometrika*, 43(3/4):353–360, 1956. [p456]
- S. Ray and M. Craven. Supervised versus multiple instance learning: An empirical comparison. In *International Conference on Machine Learning 22*, pages 697–704, 2005. [p446, 447, 453, 456]
- B. Settles, M. Craven, and S. Ray. Multiple-instance active learning. In *Advances in Neural Information Processing Systems*, pages 1289–1296, 2008. [p446]
- D. M. J. Tax and V. Cheplygina. MIL, a Matlab toolbox for multiple instance learning, 2016. URL <http://prlab.tudelft.nl/david-tax/mil.html>. version 1.2.1. [p446]
- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58(1):267–288, 1996. [p446, 448]
- X. Xu and E. Frank. Logistic regression and boosting for labeled bags of instances. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining '04*, pages 272–281, 2004. [p446, 447, 448, 453, 456]
- J. Yang. MILL: A multiple instance learning library, 2008. URL <http://www.cs.cmu.edu/~juny/MILL/>. version 1.00. [p446]

- J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *International Journal of Computer Vision*, 73(2): 213–238, 2007. [p446]
- Q. Zhang and S. A. Goldman. EM-DD: An improved multiple-instance learning technique. In *Advances in Neural Information Processing Systems 14*, pages 1073–1080, 2002. [p446]
- Z.-H. Zhou, Y.-Y. Sun, and Y.-F. Li. Multi-instance learning by treating instances as non-iid samples. In *Annual International Conference on Machine Learning 26*, pages 1249–1256, 2009. [p446]

Ping-Yang Chen

Department of Statistics, National Cheng Kung University
1 University Road,
Tainan 70101, Taiwan.
pychen.ping@gmail.com

Ching-Chuan Chen

Department of Statistics, National Cheng Kung University
1 University Road,
Tainan 70101, Taiwan.
zw12356@gmail.com

Chun-Hao Yang

Department of Statistics, University of Florida
University of Florida
Gainesville, FL 32611
chunhaoyang@ufl.edu

Sheng-Mao Chang

Department of Statistics, National Cheng Kung University
1 University Road,
Tainan 70101, Taiwan.
smchang@mail.ncku.edu.tw

Kuo-Jung Lee

Department of Statistics, National Cheng Kung University
1 University Road,
Tainan 70101, Taiwan.
kuojunglee@mail.ncku.edu.tw

spcadjust: An R Package for Adjusting for Estimation Error in Control Charts

by Axel Gandy and Jan Terje Kvaløy

Abstract In practical applications of control charts the in-control state and the corresponding chart parameters are usually estimated based on some past in-control data. The estimation error then needs to be accounted for. In this paper we present an R package, **spcadjust**, which implements a bootstrap based method for adjusting monitoring schemes to take into account the estimation error. By bootstrapping the past data this method guarantees, with a certain probability, a conditional performance of the chart. In **spcadjust** the method is implemented for various types of Shewhart, CUSUM and EWMA charts, various performance criteria, and both parametric and non-parametric bootstrap schemes. In addition to the basic charts, charts based on linear and logistic regression models for risk adjusted monitoring are included, and it is easy for the user to add further charts. Use of the package is demonstrated by examples.

Introduction

Control charts for statistical process monitoring are commonly used in a variety of different areas like industrial process control, medicine, finance, insurance, environmental science etc. See for instance Stoumbos et al. (2000), Woodall (2006), Frisé (2008), Schmid (2007a), and Schmid (2007b) for an overview.

A challenge in most practical applications of control charts is that the in-control state of the process to be monitored needs to be estimated. This introduces estimation error which needs to be accounted for. A common convention in many applications has been to assume the in-control distribution to be known and ignore the estimation error (e.g. Grigg and Farewell, 2004; Bottle and Aylin, 2008; Biswas and Kalbfleisch, 2008; Fouladirad et al., 2008; Gandy et al., 2010). However, there is an increasing awareness that the estimation error might have a detrimental effect on the performance of control charts and many authors have addressed this for various specific charts (e.g. Jones, 2002; Jones et al., 2004; Albers and Kallenberg, 2004, 2005; Albers et al., 2005; Jensen et al., 2006; Champ and Jones-Farmer, 2007; Chatterjee and Qiu, 2009; Capizzi and Masarotto, 2009; Zhang et al., 2011; Saleh et al., 2015; Zhang et al., 2016).

Gandy and Kvaløy (2013) presented a bootstrap based method for adjusting for estimation error which applies to a wide variety of control charts and different performance measures. The method can in particular be used to give a guaranteed conditional in-control performance of the chart. A typical application is to calculate an adjusted signal limit of a chart to guarantee with high probability that the in-control average run length or hitting probability is not below/above a specified value. Theoretical properties and conditions needed for the method are worked out in Gandy and Kvaløy (2013). A similar bootstrap approach was also briefly mentioned in Jones and Steiner (2012) in a study of risk-adjusted CUSUM charts.

We have developed an R package called **spcadjust** (Gandy and Kvaløy, 2016) which implements the bootstrap method for a number of different charts, with different performance measures and both parametric and non-parametric bootstrapping procedures. The package covers the most common set-ups for Shewhart, CUSUM and EWMA charts, including risk-adjusted charts. Moreover, it is easy for the user to add further charts, data models and estimation procedures.

There exist several other R packages for control charts. The **surveillance** package (Salmon et al., 2016) provides a variety of methods for monitoring, simulation and visualization of temporal and spatio-temporal data. The packages **spc** (Knot, 2016), **qcc** (Scrucca, 2004), **IQCC** (Scrucca, 2014) and **qcr** (Flores, 2016) provide functions for calculating various performance measures, signal limits, graphical displays etc. for a selection of classical control charts. The **edcc** package (Zhu and Park, 2013) has functions for economic design of control charts, while the **MSQC** package (Santos-Fernández, 2013) provides a tool kit for multivariate process monitoring. However, none of these packages include methods for taking into account the impact of estimation error on the performance of the charts, which is the main novelty of the **spcadjust** package.

In the next section, we give a brief introduction to the problem and describe the bootstrap approach for adjusting for estimation error. In the two sections thereafter we first illustrate basic use of the **spcadjust** package, then we describe details of the package and illustrate more advanced use. In the last section, a real data example is provided.

Stable versions of the package **spcadjust** are available on CRAN. Development versions are available in an open git repository (<https://bitbucket.org/agandy/spcadjust/>). We welcome con-

tributions to the functionality of the package.

Adjusting for estimation error in control charts

Control charts are a set of statistical techniques for monitoring a stream of data over time. A typical application is to monitor whether a stream of measurements follows a certain distribution, often called the in-control distribution, over time. If the distribution of the measurements deviates from the in-control distribution in a certain way the control chart should quickly detect this. In more advanced situations, regression adjustments are needed, and the monitoring is based on detecting deviations from the regression model.

In most common control charts like Shewhart, CUSUM and EWMA charts, a function of the observations is plotted for each new observation and a signal is given if this function crosses a certain threshold. Both the parameters of the function and the threshold are in most cases calculated according to estimates of the in-control distribution. This implies that estimation error will affect the performance of the control charts. For instance, measurement errors might lead to control charts that give false alarms far too often.

We first give a motivating example, and then our bootstrap method for handling estimation error is described.

Motivating example

We consider an example with a CUSUM chart for monitoring changes in the mean of a stream of normally distributed data. Assume first that we know that the stream consists of independent observations X_1, X_2, \dots following a $N(\mu, \sigma)$ distribution in the in-control situation, and that we want to quickly detect if there is a change in the mean to the out-of-control situation $\mu + \Delta$. A standard CUSUM chart for this would be to plot S_t versus t where

$$S_t = \max \left(0, S_{t-1} + \frac{X_t - \mu - \Delta/2}{\sigma} \right), \quad S_0 = 0, \tag{1}$$

where the chart signals once $S_t > c$. The threshold c is calculated to give a certain performance of the chart if no change occurs. Often, c is chosen to give a pre-specified in-control mean number of steps until a false alarm (often called average run length, ARL). Thus the CUSUM chart and the threshold c should be calculated using the true $N(\mu, \sigma)$ in-control distribution.

However, in practice, the true in-control distribution $N(\mu, \sigma)$ is usually unknown, but estimated based on n past in-control observations X_{-n}, \dots, X_{-1} . For example, we can estimate the in-control mean by $\hat{\mu} = \frac{1}{n} \sum_{i=-n}^{-1} X_i$ and the in-control variance by $\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=-n}^{-1} (X_i - \hat{\mu})^2$. The CUSUM chart is then given by

$$S_t = \max \left(0, S_{t-1} + \frac{X_t - \hat{\mu} - \Delta/2}{\hat{\sigma}} \right), \quad S_0 = 0,$$

which signals when $S_t > \hat{c}$ where \hat{c} is calculated using the estimated in-control distribution $N(\hat{\mu}, \hat{\sigma})$.

Since the future in-control data come from the true in-control distribution $N(\mu, \sigma)$ rather than the estimated distribution $N(\hat{\mu}, \hat{\sigma})$, the performance of the chart might be substantially wrong, leading to more (or less) false alarms than expected.

A typical application of our bootstrap method described in the next subsections will be to calculate an adjustment to the naive threshold estimate \hat{c} such that a certain in control behavior of the chart can be guaranteed with high probability.

As an illustration we generated $n = 100$ past data from a $N(0, 1)$ distribution giving estimates $\hat{\mu} = -0.028, \hat{\sigma} = 0.921$. We want to monitor for an increase in the mean of the distribution by $\Delta = 1$ and achieve an in-control ARL of 500.

Assuming (incorrectly) that the estimated parameters are the true parameters this would lead to a threshold of $\hat{c} = 4.1$. Our bootstrap method leads to an adjusted threshold of 5.5, calculated such that the ARL of 500 is achieved with a probability of 90%.

The result of running the CUSUM chart with these estimated parameters on a stream of $N(0, 1)$ in-control data is illustrated in the top row of Figure 1. There is a false alarm with the unadjusted threshold, but not with the adjusted threshold.

In the bottom row of Figure 1 the same CUSUM chart is run on data which are out of control from observation 81 and onwards. There is only a slight delay in the detection of the out-of-control situation with the adjusted threshold. Also, there are two false alarms in the first 80 observations with the unadjusted threshold.

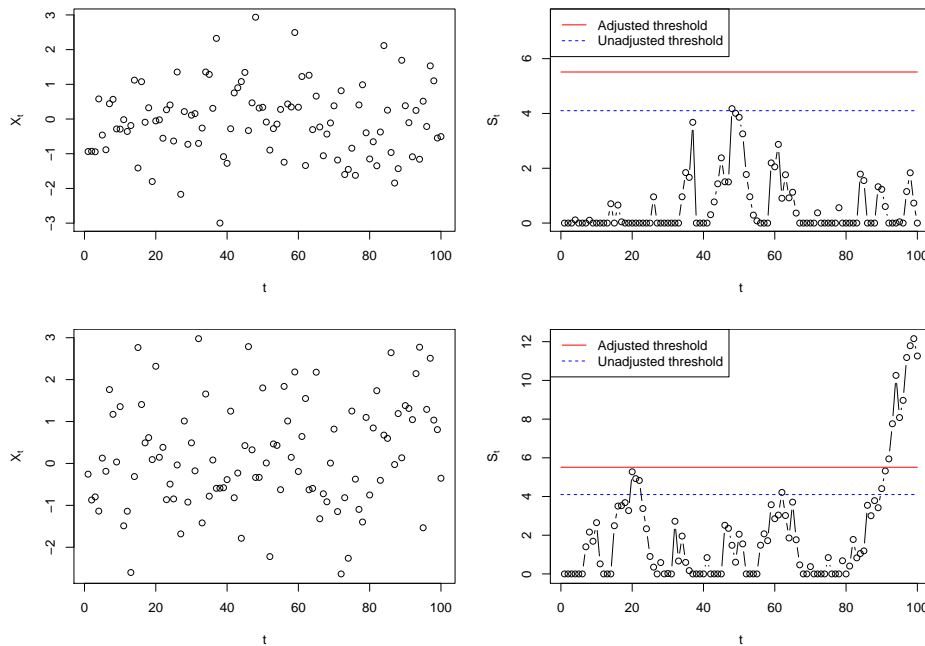


Figure 1: CUSUM charts for the motivating example. Top row: in-control data. Bottom row: data which switch to out-of-control from observation 81. The left figures show the observed data streams, the right the resulting CUSUM charts which were run with the estimated parameters. Adjusted and unadjusted thresholds are indicated.

Homogeneous observations

In this section we describe our bootstrap procedure for situations with homogeneous observations. Assume that in-control we have a stream of independent observations X_1, X_2, \dots , following a distribution P . A control chart is used to detect when observations are no longer coming from P , also called an out of control situation, for instance a shift in mean or variance.

To run such a chart, certain parameters, ξ , calculated from P are usually needed. However, in most applications the exact distribution P is unknown, we only have an estimate of the distribution, and thus have to run the chart with estimated parameters. In the example in the previous section, the parameters needed to run the chart are $\xi = (\mu, \sigma)$ and the unknown in-control distribution P is $N(\mu, \sigma)$.

Common performance measures for control charts are the ARL and the hitting probability of the chart within a certain number of steps. These depend both on the unknown P as well as on the parameters ξ . Indeed, let τ denote the time (observation number) at which a chart gives a signal, e.g. the first time a CUSUM chart has a value above c . The distribution of this stopping time τ depends on P and ξ . We can express the ARL as $ARL(P; \xi) = \mathbb{E}(\tau(\xi))$, where the expectation is with respect to P . The probability of signaling within m time steps (for some finite $m > 0$) can be expressed as $hit(P; \xi) = P(\tau(\xi) \leq m)$.

The signal limit c is chosen to achieve a certain in-control performance of the chart. For charts which signal when a threshold c is crossed we can express this as follows.

- $c_{ARL}(P; \xi) = \inf\{c > 0 : ARL(P; \xi) \geq \gamma\}$ for some $\gamma > 0$, i.e. the threshold needed to give an in-control ARL of γ .
- $c_{hit}(P; \xi) = \inf\{c > 0 : hit(P; \xi) \leq \beta\}$ for some $0 < \beta < 1$, which is the threshold needed to give a false alarm probability of β .

Again, both c_{ARL} and c_{hit} depend on P and ξ .

Let \hat{P} and $\hat{\xi} = \xi(\hat{P})$ denote the estimated distribution and estimated parameters, estimated from past in-control data X_{-n}, \dots, X_{-1} . Since we have to run the chart with $\hat{\xi}$ while future in-control data follow P ideally we should use the threshold $c_{ARL}(P; \hat{\xi})$ or $c_{hit}(P; \hat{\xi})$. These are unknown since P is unknown. Instead in practice $c_{ARL}(\hat{P}; \hat{\xi})$ or $c_{hit}(\hat{P}; \hat{\xi})$ is often used, but this may lead to performance substantially off from the nominal performance.

The suggestion in Gandy and Kvaløy (2013) is to calculate, by bootstrapping, an adjusted threshold which with high probability will guarantee that the in-control performance is not worse than the nominal value. For instance to calculate an adjustment to $c_{ARL}(\hat{P}; \hat{\xi})$ which with a probability $1 - \alpha$

guarantees that the true in-control ARL is at least as large as the nominal value.

To make a unified presentation of the bootstrap we let q be a common notation for quantities of interest like c_{ARL} and c_{hit} , or simple transformations such as $\log(c_{ARL})$ and $\log(c_{hit})$. Further let p_α be a constant such that

$$P(q(\hat{P}; \hat{\xi}) - q(P; \xi) > p_\alpha) = 1 - \alpha,$$

which implies the following bound on the quantity of interest

$$P(q(P; \xi) < q(\hat{P}; \hat{\xi}) - p_\alpha) = 1 - \alpha.$$

Since P is unknown we cannot calculate p_α , but an approximation can be obtain by bootstrapping. Let \hat{P}^* denote a parametric or non-parametric bootstrap replicate of the estimated in-control distribution \hat{P} , based on the same sample size n as \hat{P} , and let $\hat{\xi}^* = \xi(\hat{P}^*)$. Then we can approximate p_α by p_α^* where

$$P(q(\hat{P}^*; \hat{\xi}^*) - q(\hat{P}; \hat{\xi}) > p_\alpha^* | \hat{P}) = 1 - \alpha.$$

Then an approximate upper bound which guarantees a certain performance with an approximate probability of $1 - \alpha$ is $q(\hat{P}; \hat{\xi}) - p_\alpha^*$. We can also think of

$$(-\infty, q(\hat{P}; \hat{\xi}) - p_\alpha^*)$$

as a sort of one-sided (approximate) confidence interval for $q(P; \xi)$. The bootstrap distribution has to be approximated by simulations, easily performed by generating B bootstrap samples from \hat{P} and calculating $q(\hat{P}^*_b; \hat{\xi}^*_b) - q(\hat{P}; \hat{\xi})$ for each bootstrap sample. For further details, including theoretical properties, we refer to Gandy and Kvaløy (2013).

If we return to the example in the previous section and consider the threshold needed to get a certain in-control ARL value γ , we adapt the above procedure with either $q = c_{ARL}$ or $q = \log(c_{ARL})$. The latter is recommended as the log-transform usually improves the precision (Gandy and Kvaløy, 2013). Then B bootstrap samples of size n are generated from \hat{P} and from these we calculate $\hat{P}^*_1, \dots, \hat{P}^*_B$ and $\hat{\xi}^*_1, \dots, \hat{\xi}^*_B$. With $q = \log(c_{ARL})$ we then calculate p_α^* as the $1 - \alpha$ empirical quantile of $\log c_{ARL}(\hat{P}^*_b; \hat{\xi}^*_b) - \log c_{ARL}(\hat{P}; \hat{\xi})$, $b = 1, \dots, B$, and obtain the adjusted threshold $\exp(\log c_{ARL}(\hat{P}; \hat{\xi}) - p_\alpha^*) = c_{ARL}(\hat{P}; \hat{\xi}) \exp(-p_\alpha^*)$. With this adjusted threshold there is an approximate $1 - \alpha$ probability that the actual ARL of the charts is at least γ .

Risk-adjusted charts

In many applications of control charts the units being monitored are heterogeneous, for instance when monitoring data from human beings. To make reasonable monitoring systems in such situations the explainable part of the difference between units should be accounted for by regression models. Charts based on regression models are often called risk-adjusted, an overview of some such charts is found in Grigg and Farewell (2004).

For risk-adjusted charts the regression model has to be estimated based on past data, and the impact of estimation error thus has to be taken into account. The bootstrap procedure outlined in the previous section also applies to risk adjusted charts. Let the stream of observations now be denoted $(Y_1, X_1), (Y_2, X_2), \dots$, where Y_i is a response variable and X_i a corresponding vector of covariates. Further let P denote the joint distribution of (Y_i, X_i) . For regression models we recommend to use a non-parametric bootstrap. Let \hat{P} be the empirical distribution which puts weight $1/n$ on each of the n past observations $(Y_{-n}, X_{-n}), \dots, (Y_{-1}, X_{-1})$. Then by resampling from this \hat{P} the bootstrap procedure also applies to a wide variety of risk adjusted charts (Gandy and Kvaløy, 2013).

As an example consider a CUSUM chart for a linear regression model. Suppose that in-control $E(Y_i|X_i) = X_i\beta$ (where the first component of X_i is 1) and we want to detect a change in the mean response to $E(Y_i|X_i) = \Delta + X_i\beta$ for some $\Delta > 0$. For linear regression models it is natural to base the monitoring on the residuals of the model (Horváth et al., 2004). A CUSUM to monitor changes in the mean can for instance be defined by

$$S_t = \max(0, S_{t-1} + Y_t - X_t\beta - \Delta/2), \quad S_0 = 0,$$

which signals when $S_t \geq c$. In practice since β and the distribution of the residuals, P_ϵ , are estimated the CUSUM is run with $\hat{\beta}$ and a threshold calculated e.g. as $c_{ARL}(\hat{P}_\epsilon; \hat{\beta})$. This might lead to an ARL far off from the nominal (Gandy and Kvaløy, 2013).

To account for the estimation error we can use the bootstrap procedure from the previous section with e.g. $q = \log(c_{ARL})$ and non-parametric bootstrapping as outlined above to calculate the adjusted threshold $\exp(\log c_{ARL}(\hat{P}_\epsilon; \hat{\beta}) - p_\alpha^*) = c_{ARL}(\hat{P}_\epsilon; \hat{\beta}) \exp(-p_\alpha^*)$. Using this adjusted threshold there

class name	data model
"SPCModelNormal"	normally distributed updates of the form $(X_t - \mu - \Delta/2)/\sigma$ (Sections <i>CUSUM chart with estimated in-control state</i> and <i>Shewhart chart with estimated in-control state</i>)
"SPCModelNonparCenterScale"	updates $(X_t - \mu - \Delta/2)/\sigma$, no distributional assumptions (Section <i>CUSUM chart with estimated in-control state</i>)
"SPCModelNonpar"	user defined updates, no distributional assumptions
"SPCModellm"	linear regression model with updates $Y_t - X_t\beta - \Delta/2$ (Sections <i>CUSUM chart with linear regression model</i> and <i>EWMA chart with linear regression model</i>)
"SPCModellogregLikRatio"	logistic regression model, likelihood ratio updates (Section <i>Application to cardiac surgery data</i>)
"SPCModellogregOE"	logistic regression model, observed minus expected updates

Table 1: Overview of pre-implemented data models.

is an approximate $1 - \alpha$ probability that the actual ARL of the charts is at least as large as desired. Further risk-adjusted charts are discussed in Section *EWMA chart with linear regression model* and Section *Application to cardiac surgery data*.

Other adjustments

So far we have focused on how to adjust the signal limit of the control chart to achieve a certain performance with a high probability. In practice that will be a typical application, but it is easy to change focus to other quantities. For instance, instead of adjusting the threshold to obtain a certain ARL we could instead fix the threshold and calculate which ARL we with high probability at least will achieve. For instance with Shewhart charts it is very common to use $c = 3$ standard deviations as signal limit, and if we with estimated parameters still choose to stick with this limit we can use the bootstrap approach to calculate a lower limit of the achieved ARL. In practice this is done by defining the appropriate q -function and then run the general bootstrap procedure as before. For the Shewhart example with fixed c and focus on ARL the q would simply be the ARL or $\log(ARL)$. See Gandy and Kvaløy (2013) for details and Section *Shewhart chart with estimated in-control state* for an example.

Another variant which we consider further in Section *Implementing a new type of chart* is Shewhart charts with non-symmetric signal limits for skew distributions. For such charts the signal limit can be defined in terms of the quantiles corresponding to a certain tail probability α . Then we can via the bootstrap find the appropriate adjustment of this α to achieve a certain performance with high probability.

Basic usage of the package

In this section we discuss how the package can be used for pre-implemented chart types and data models. The framework provided in the package can also easily be extended to work with other charts, data models and/or estimation procedures as will be explained in Section *Details of the package and advanced usage*.

An important basic structure of the package is that chart types and data models are implemented in separate objects and in such a way that they can be flexibly combined. The implemented chart types are the Shewhart chart (SPCShew), the CUSUM chart (SPCCUSUM) and the EWMA chart (SPCEWMA). Table 1 lists the implemented data models.

Calculation of charts and chart properties like thresholds, ARLs and hitting probabilities are defined with the chart type object. Estimation of chart parameters, the form and cdf of updates and the bootstrap procedure are defined in the data model object. With *updates* we mean the quantity added to the chart in each step. Parametric bootstrapping is used in the normal model, nonparametric bootstrapping in all the other pre-implemented models.

The parameter Δ in the updates in some of the data models listed in Table 1 specifies the out-of-control situation for CUSUM charts, but should be set to the default value $\Delta = 0$ for Shewhart and EWMA charts.

The main steps for basic usage of the package is to define a chart object with the `new()` function and to calculate the properties of interest with the `SPCproperty()` function. A generic description of

these functions is given below.

To define a chart object we need to specify the combination of chart type and data model as follows:

```
chartgeneric <- new("Charttype", model = Datamodel(Delta = x),...)
```

Here "Charttype" should be one of "SPCShew", "SPCCUSUM" or "SPCEWMA" and "Datamodel" should be one of the class names listed in Table 1. Finally x should be 0 for Shewhart and EWMA chart and set to the desired value for CUSUM charts.

The following function invokes the bootstrap procedure and calculate the property of interest for the chart:

```
SPCproperty(data, nrep, property = "specifyproperty", chart = chartgeneric,
            params = list(specifyparameters), covprob=0.9, parallel=1,...)
```

Here data are the past observations (usually a vector) and nrep is the number of bootstrap replications. Further, specifyproperty specifies the property of interest, with choices calARL, calhitprob, ARL and hitprob. The two first choices calculate a calibrated threshold to achieve, with high probability, a desired ARL or a desired hitting probability. Based on a specified choice of threshold, the two last choices calculate the smallest ARL or the largest hitting probability that is attained with high probability. Necessary parameters are given in specifyparameters (depending on the property this includes the desired ARL, the desired hitting probability or the threshold). Finally, covprob, gives the desired coverage probability $1 - \alpha$ with a default of 90%. The bootstrap can be sped up by parallel processing by specifying the number of cores to be used via the 'parallel' option.

Further functions and details are explained in the examples below. In the two first subsections below we consider situations with homogeneous observations, in the next subsections, we consider situations with risk-adjusted charts.

CUSUM chart with estimated in-control state

We now return to the motivating example in Section *Motivating example*. Recall that we want to run a CUSUM chart of the form (1) to monitor for a change in the mean in a stream of normally distributed data. We first define the chart object by specifying chart type and data model.

```
> library(spcadjust)
> chart <- new("SPCCUSUM", model = SPCModelNormal(Delta = 1))
```

Here "SPCCUSUM" specifies that it should be a CUSUM chart, and SPCModelNormal(Delta = 1) specifies that it is a model with normally distributed updates of the form $(X_t - \mu - \Delta/2)/\sigma$ with $\Delta = 1$.

Next we generate $n = 100$ past observations and use the function xiofdata to calculate the estimated chart parameters $\hat{\xi}$.

```
> X <- rnorm(100)
> xihat <- xiofdata(chart, X)
> str(xihat)
```

```
List of 3
 $ mu: num -0.0284
 $ sd: num 0.921
 $ m : int 100
```

Now we can use the function SPCproperty() to calculate the naive and adjusted threshold:

```
> cal <- SPCproperty(data = X, nrep = 50, property = "calARL",
+                   chart = chart, params = list(target = 500), covprob = 0.9, quiet = TRUE)
> cal
```

90 % CI: A threshold of 5.513 gives an in-control ARL of at least 500.

Unadjusted result: 4.101

Based on 50 bootstrap repetitions.

Here property = "calARL" specifies that the chart should be calibrated to achieve a certain ARL, this ARL is specified to be 500 (target = 500) and the probability of attaining at least this ARL specified to be 90% (covprob = 0.9). The adjusted threshold of 5.5 is calculated using parametric bootstrapping with nrep replications assuming normality of the observations. For real applications nrep should of course be more than 50. If nonparametric bootstrapping is preferred the model specification in the definition of the chart should be replaced by SPCModelNonparCenterScale().

If we rather would like to calibrate the chart according to a certain hitting probability, for instance a hitting probability of 0.05 within 100 steps, this is achieved by specifying property = "calhitprob" and params = list(target = 0.05, nsteps = 100):

```
> SPCproperty(data = X, nrep = 50, property = "calhitprob",
+             chart = chart, params = list(target = 0.05, nsteps=100), covprob = 0.9,
+             quiet = TRUE)
90 % CI: A threshold of 7.137 gives an in-control false alarm probability
of at most 0.05 within 100 steps.
Unadjusted result: 5.285
Based on 50 bootstrap repetitions.
```

The function `runchart` is used to run the chart on future data, and the option `xi` specifies which parameters to use when running the chart:

```
> newX <- rnorm(100)
> S <- runchart(chart, newdata = newX, xi = xihat)
```

The following code produces the plots in the first row of Figure 1, using the ARL calibrated threshold calculated above:

```
> par(mfrow = c(1, 2), mar = c(4, 5, 0, 0))
> plot(newX, xlab = "t")
> plot(S, ylab = expression(S[t]), xlab = "t", type = "b",
+      ylim = range(S, cal@res+2, cal@raw))
> lines(c(0,100), rep(cal@res, 2), col = "red")
> lines(c(0,100), rep(cal@raw, 2), col = "blue", lty = 2)
> legend("topleft", c("Adjusted threshold", "Unadjusted threshold"),
+       col = c("red", "blue"), lty = 1:2)
```

Shewhart chart with estimated in-control state

Next we consider a two-sided Shewhart chart, assuming that all observations are normally distributed. The in-control mean and standard deviation are estimated from n past in-control observations X_{-n}, \dots, X_{-1} . For new observations X_1, X_2, \dots a two-sided Shewhart chart is defined by

$$S_t = \frac{X_t - \hat{\mu}}{\hat{\sigma}},$$

which signals when $|S_t| > c$ for some threshold c . A common choice for Shewhart charts is to set $c = 3$, corresponding to three standard deviations if the chart is run with the correct in-control mean and standard deviation.

We first define the chart by

```
> chartShew <- new("SPCShew", model = SPCModelNormal(), twosided = TRUE)
```

and then generate $n = 250$ past observations and estimate the chart parameters:

```
> X <- rnorm(250)
> xihat <- xiofdata(chartShew, X)
> str(xihat)
List of 3
 $ mu: num 0.0251
 $ sd: num 1.05
 $ m : int 250
```

If the Shewhart chart is run with the standard threshold $c = 3$, we can use the bootstrap method to calculate a lower limit for the actual ARL of the chart by specifying `property = "ARL"` and `params = list(threshold = 3)`:

```
> SPCproperty(data = X, nrep = 50, property = "ARL", chart = chartShew,
+             params = list(threshold = 3), quiet = TRUE)
90 % CI: A threshold of 3 gives an in-control ARL of at least 213.1.
Unadjusted result: 370.4
Based on 50 bootstrap repetitions.
```

A two-sided Shewhart chart for normally distributed data with true parameters and a threshold of $c = 3$ will correspond to an ARL of roughly 370. We can thus compute an adjusted threshold that with roughly 90% probability results in an average run length of at least 370 in control:

```
> cal <- SPCproperty(data = X, nrep = 50, property = "calARL", chart = chartShew,
+                   params = list(target = 370), quiet = TRUE)
```

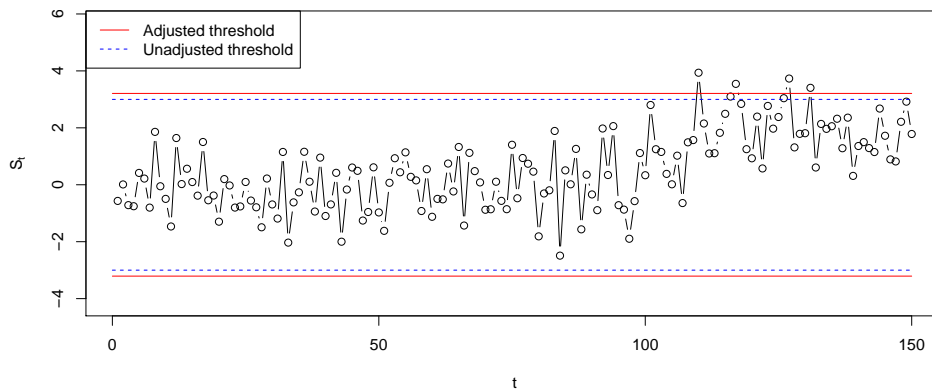



Figure 2: Shewhart charts with estimated parameters run on data which are in-control until observation 100 and with a shift in the mean from observation 101 an onwards.

```
> cal
90 % CI: A threshold of 3.209 gives an in-control ARL of at least 370.
Unadjusted result: 3
Based on 50 bootstrap repetitions.
```

Finally we run the chart with new observations. The simulated new observations are in-control for the first 100 observations, and then there is a shift in the mean from observations 101 an onwards. The corresponding plot is given in Figure 2.

```
> newX <- rnorm(150, mean = c(rep(0, 100), rep(2, 50)))
> S <- runchart(chartShew, newdata = newX, xi = xihat)
```

CUSUM chart with linear regression model

The set up is as described in Section *Risk-adjusted charts*, and with estimated regression coefficients $\hat{\beta}$ the CUSUM is

$$S_t = \max(0, S_{t-1} + Y_t - X_t\hat{\beta} - \Delta/2), \quad S_0 = 0.$$

The following generates a data set of past observations from the model $\mathbb{E}(Y) = 2 + x_1 + x_2 + x_3$ with standard normal noise and distribution of the covariate values as specified below.

```
> n <- 500
> Xlinreg <- data.frame(x1 = rbinom(n, 1, 0.4), x2 = runif(n, 0, 1), x3 = rnorm(n))
> Xlinreg$y <- 2 + Xlinreg$x1 + Xlinreg$x2 + Xlinreg$x3 + rnorm(n)
```

Next, we initialize the chart

```
> chartlinregCUSUM <-
+ new("SPCCUSUM", model = SPCModellm(Delta = 1, formula = "y~x1+x2+x3"))
```

where SPCModellm() uses non-parametric bootstrapping as explained in Section *Risk-adjusted charts*. The estimated parameters for running the chart, $\hat{\beta}$, are:

```
> xihat <- xiofdata(chartlinregCUSUM, Xlinreg)
> xihat
Call:
lm(formula = formula, data = P)
```

Coefficients:
(Intercept) x1 x2 x3
2.0222 1.0360 1.0350 0.9711

Next we find the threshold that with roughly 90% probability results in an average run length of at least 100 in control.

```
> cal <- SPCproperty(data = Xlinreg, nrep = 50, property = "calARL",
+ chart = chartlinregCUSUM, params = list(target = 100), quiet = TRUE)
> cal
90 % CI: A threshold of 3.138 gives an in-control ARL of at least 100.
```

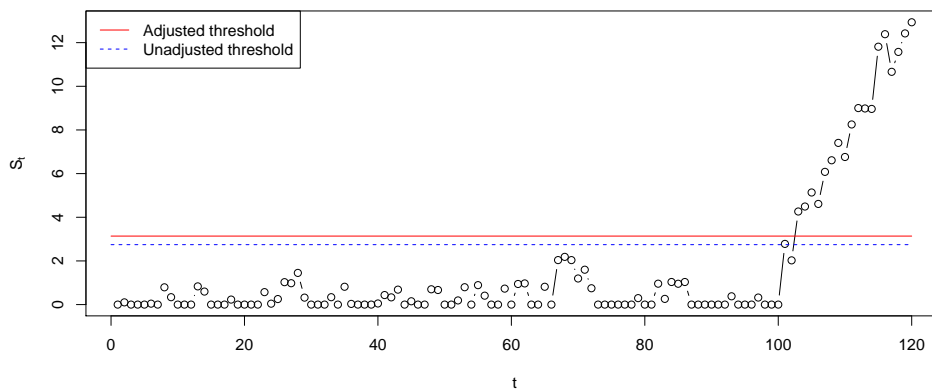


Figure 3: CUSUM chart for a linear regression model with estimated parameters. The data shift from in-control to out-of-control from observation 101 onwards.

Unadjusted result: 2.745
Based on 50 bootstrap repetitions.

Finally, we run the chart with new observations that are in-control for the first 100 observations and then switches to out-of-control. A plot of the resulting CUSUM is given in Figure 3.

```
> n <- 120
> newXlinreg <- data.frame(x1 = rbinom(n, 1, 0.4), x2 = runif(n, 0, 1),
+                          x3 = rnorm(n))
> outind <- c(rep(0, 100), rep(1, n-100))
> newXlinreg$y <-
+ 2 + newXlinreg$x1 + newXlinreg$x2 + newXlinreg$x3 + rnorm(n) + outind
> S <- runchart(chartlinregCUSUM, newdata = newXlinreg, xi = xihat)
```

EWMA chart with linear regression model

An EWMA chart based on the residuals of a linear regression model can be defined by

$$M_t = \lambda(Y_t - X_t\beta) + (1 - \lambda)M_{t-1}, \quad M_0 = 0,$$

where λ is a smoothing parameter determining how to weight the most recent observation versus the past data. We can now set up the chart, calculate adjusted thresholds and run the chart on the new data with estimated parameters in the same manner as for the CUSUM chart. The only differences are that we have to specify "SPCEWMA", $\Delta = 0$ and a value of λ when the chart is initialized.

```
> chartlinregEWMA <- new("SPCEWMA", model = SPCModelm(Delta = 0,
+ formula = "y~x1+x2+x3"), lambda = 0.1)
> calEWMA <- SPCproperty(data = Xlinreg, nrep = 50, property = "calARL",
+ chart = chartlinregEWMA, params = list(target = 100), quiet = TRUE)
> calEWMA
90 % CI: A threshold of +/- 0.5337 gives an in-control ARL of at least 100.
Unadjusted result: 0.496
Based on 50 bootstrap repetitions.
> xihat <- xiofdata(chartlinregEWMA, Xlinreg)
> M <- runchart(chartlinregEWMA, newdata = newXlinreg, xi = xihat)
```

A plot of the resulting EWMA chart is given in Figure 4.

Further usage of risk-adjusted charts will be demonstrated in Section [Application to cardiac surgery data](#) where a CUSUM for logistic regression will be explained and used.

Details of the package and advanced usage

A basic structure of the package is, as explained in the introduction of Section [Basic usage of the package](#), a definition of two types of objects. We will now look further into the details of these two objects and how they can be used to add new charts, new data models and other estimation procedures.

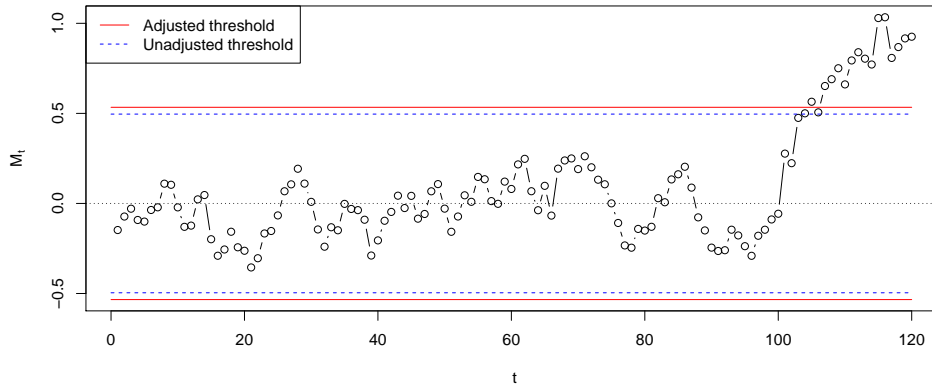


Figure 4: EWMA chart for a linear regression model with estimated parameters. The data shift from in-control to out-of-control from observation 101 onwards.

One object is an S3 class of type "SPCDataModel" that implements how observed data are used to fit the model and how updates for the chart are being computed. The second object is an S4 class of type "SPCchart" which implements how these updates are converted into charts and how the charts are being calibrated. The main advantage of this separation into two different objects is that it reduces the amount of redundancy in the code.

The package was originally developed with S4 classes only, to take advantage of the more flexible method dispatch. However, to improve performance, the data model classes, whose methods are called very frequently, were switched to S3 classes.

Bespoke data model

We first focus on how to generate a bespoke data model. For this one needs to implement a class of type "SPCDataModel". Every element of the class has to consist of a list of the following functions: updates, getcdfupdates, Pofdata, resample, xiofP, which have to be of a specific form. The arguments generally have the following meaning: xi denotes the parameter vector needed to create updates for running the chart from observed data, data is observed data, P is a data model.

- updates(xi, data): Returns updates for the chart using the parameter xi and the observed data data.
- Pofdata(data): Estimates a probability model from the data.
- xiofP(P): Computes the parameter xi needed to compute updates from an (estimated) probability model P.
- resample(P): Generates a new data set from the probability model P.
- getcdfupdates(P, xi): Returns the cumulative distribution function (CDF) of updates with data generated from the probability model P and updates computed using the parameter xi.

In the following we give some examples.

Robust estimation

Consider again the example discussed in Sections *Motivating example* and *CUSUM chart with estimated in-control state* of a CUSUM chart which assumes a normal distribution of the observations with unknown mean and standard deviation. We now demonstrate how to change the estimators to use the median the mean absolute deviation (MAD) instead of using the mean and the sample standard deviation. Using these robust estimators could be desirable if there could be outliers present in the past in-control data.

For this we only need to override one function of the existing data model "SPCModelNormal", the method Pofdata that estimates the parameters. The format of the input and output for the new Pofdata function needs to be unchanged. The following code first list the old function and then overrides it with the new.

```
> model <- SPCModelNormal(Delta = 1)
> model$Pofdata
function (data)
{
```

```

      list(mu = mean(data), sd = sd(data), m = length(data))
    }
  }
  > model$Pofdata <- function(data){
+   list(mu = median(data), sd = mad(data), m = length(data))
+ }

```

Properties of this chart can then be computed as before:

```

> X <- rnorm(100)
> chartrobust <- new("SPCCUSUM", model = model)
> SPCproperty(data = X, nrep = 50, property = "calARL",
+   chart = chartrobust, params = list(target = 100), quiet = TRUE)
90 % CI: A threshold of 4.162 gives an in-control ARL of at least 100.
Unadjusted result: 2.987
Based on 50 bootstrap repetitions.

```

Parametric exponential CUSUM chart

In this example we illustrate how to construct a CUSUM chart that assumes that the observations are coming from an exponential distribution with unknown rate λ in control. Again, only the data model needs to be defined, but now all functions are needed from scratch. I.e. we need to define all the functions updates, Pofdata, xiofP, resample and getcdfupdates. The basic CUSUM chart class "SPCCUSUM" will be used without changes.

The updates for a CUSUM chart can in general situations be based on the log likelihood ratio between an out-of-control model and the in-control model (Hawkins and Olwell, 1998). I.e. the CUSUM can be written

$$S_t = \max(0, S_{t-1} + R_t), \quad S_0 = 0,$$

where the update R_t is the log likelihood ratio for observation t . Suppose that we want to detect a change of the rate to $\lambda\Delta$ for some given $\Delta > 0$, $\Delta \neq 1$. To define the updates, we need to compute the log likelihood ratio between the out-of-control and the in-control model for an observation X_t , which gives

$$R_t = \log \left(\frac{\lambda\Delta \exp(-\lambda\Delta X_t)}{\lambda \exp(-\lambda X_t)} \right) = \log(\Delta) - \lambda(\Delta - 1)X_t,$$

defining the function updates.

To define the data model the CDF of these updates must also be computed. This can be done in closed form, but requires distinguishing the case $\Delta > 1$ and $\Delta < 1$ and taking into account that the rate parameter used in the updates typically differs from the true rate parameter. Let $\hat{\lambda}$ be the rate parameter used in the updates (typically an estimated parameter) and λ be the true parameter. Then the cdf (conditional on the value of $\hat{\lambda}$) is

$$P(\log(\Delta) - \hat{\lambda}(\Delta - 1)X_i \leq x) = \begin{cases} 1 - \exp(-\lambda(x - \log(\Delta))/(\hat{\lambda}(1 - \Delta))) & \text{for } \Delta < 1 \\ \exp(-\lambda(\log(\Delta) - x)/(\hat{\lambda}(\Delta - 1))) & \text{for } \Delta > 1, \end{cases}$$

which needs to be implemented in getcdfupdates. We decide to use parametric resampling of the data (resample) and we decide to estimate the parameter λ based on the past observations X_{-n}, \dots, X_{-1} using the maximum likelihood estimator $\hat{\lambda} = n / \sum_{i=1}^n X_{-i}$.

The following code implements this.

```

> SPCModelExponential = function(Delta = 1.25){
+   structure(list(
+     Pofdata = function(data){
+       list(lambda = 1/mean(data), n = length(data))
+     },
+     xiofP = function(P) P,
+     resample = function(P) rexp(P$n, rate = P$lambda),
+     getcdfupdates = function(P, xi) {
+       if (Delta<1)
+         function(x)
+           pmax(0, 1-exp(-P$lambda*(x-log(Delta))/(xi$lambda*(1-Delta))))
+       else
+         function(x)
+           pmin(1, exp(-P$lambda*(log(Delta)-x)/(xi$lambda*(Delta-1))))
+     },
+     updates = function(xi, data) log(Delta)-xi$lambda*(Delta-1)*data

```

```
+   ), class = "SPCDataModel")
+ }
```

Next, we put this into practice. First we initiate the chart.

```
> ExpCUSUMchart <- new("SPCCUSUM", model = SPCModelExponential(Delta = 1.25))
```

The following creates some past observations and compute the threshold needed to achieve an ARL of 1000.

```
> X <- rexp(500)
> cal <- SPCproperty(data = X, nrep = 50, property = "calARL", chart = ExpCUSUMchart,
+                   params = list(target = 1000), covprob = 0.9, quiet = TRUE)
> cal
90 % CI: A threshold of 4.054 gives an in-control ARL of at least 1000.
Unadjusted result: 3.165
Based on 50 bootstrap repetitions.
```

Finally, we generate some new data and make a CUSUM plot with thresholds which is displayed in Figure 5.

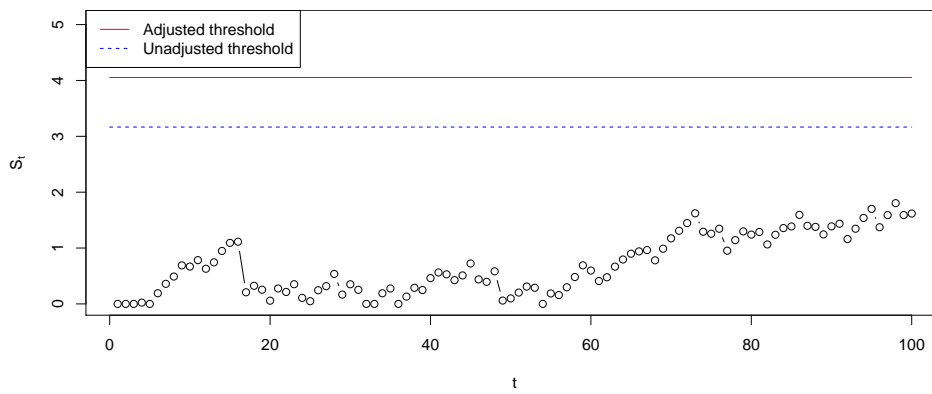


Figure 5: CUSUM chart for the rate of an exponential distribution with estimated parameters. The data are in control.

Implementing a new type of chart

We now discuss the chart model and how to implement new charts. Every chart is an S4 class derived from the class "SPCchart". It has one slot, model, which contains the data model to be used with the chart. The main method that needs to be implemented is the method getq, which computes desired properties of a given control chart. It receives two arguments: which property to report (a string, e.g. ARL, hitprob, calARL, calhitprob) and additional parameters for this property, e.g. a threshold when computing the ARL (property ARL), a threshold and a number of steps when computing hitting probabilities (property hitprob), a desired ARL when calibrating the threshold (property calARL).

We now give one example of how to implement a new chart. Suppose the in-control distribution is assumed to have a distribution with continuous cdf F , which does not need to be symmetric. Then we can define a Shewhart type chart which signals if an observation X is in the upper $\alpha/2$ or lower $\alpha/2$ quantile for a given threshold $\alpha > 0$, i.e. if

$$X \leq F^{-1}(\alpha/2) = \hat{f}_{\alpha/2} \text{ or } X \geq F^{-1}(1 - \alpha/2) = \hat{f}_{1-\alpha/2}.$$

This can be termed a Shewhart chart with asymmetric control limits (Chen and Kuo, 2010).

The main work in implementing a chart is implementing functions that compute properties of the chart (e.g. ARL in control, threshold needed to give a certain ARL or hitting probabilities within certain steps). These properties need to be computed given the parameter that is used for running the chart (x_i) and given the distribution of the observations (P).

In this example we implement two methods: one for computing the ARL (ARL) and one for computing the α needed to give a certain ARL (calARL). The ARL for the Shewhart chart with asymmetric control limits run with estimated parameters will be

$$\frac{1}{P(X \leq \hat{f}_{\alpha/2}) + P(X \geq \hat{f}_{1-\alpha/2})} = \frac{1}{F(\hat{f}_{\alpha/2}) + 1 - F(\hat{f}_{1-\alpha/2})}$$

and in the implementation below we make a log transform of this to increase the accuracy of the bootstrap. To implement the property `calARL`, the α needed to achieve a certain ARL is found by a numerical search using the above expression (we have implemented this with ad hoc choices for the boundaries of the numeric search). A logit transform is here used in the implementation to improve the bootstrap.

```
> setClass("SPCShewAsym", contains = c("SPCchart"))
> setMethod("getq", signature = "SPCShewAsym", function(chart, property, params){
+   if (property == "calARL"){
+     list(
+       q = function(P, xi){
+         pobs <- function(alpha){
+           getcdfupdates(chart, xi = xi, P = P)(xi$quant(alpha/2))
+           +(1-getcdfupdates(chart, xi = xi, P = P)(xi$quant(1-alpha/2)))
+         }
+         res <- uniroot(function(x) params$target-(1/pobs(x)),
+           lower = 1e-7, upper = 0.4)$root
+         as.double(log(res/(1-res)))
+       },
+       trafo = function(x) exp(x)/(1+exp(x)),
+       lowerconf = FALSE,
+       format = function(res)
+         paste("A threshold of alpha=", format(res, digits = 4),
+           " gives an in-control ARL of at least ",
+           params$target, ".", sep = "", collapse = "")
+     )
+   }else if (property == "ARL"){
+     list(
+       q = function(P, xi){
+         -log(getcdfupdates(chart, xi = xi, P = P)(xi$quant(params$alpha/2))
+           +(1-getcdfupdates(chart, xi = xi, P = P)(xi$quant(1-params$alpha/2)))
+         },
+       trafo = function(x) exp(x),
+       lowerconf = FALSE,
+       format = function(res)
+         paste("A threshold defined by alpha=", params$alpha,
+           " gives an in-control ARL of at least ",
+           format(res, digits = 4), ".", sep = "", collapse = "")
+     )
+   }else stop("property ", property, " not implemented.")
+ })
[1] "getq"
```

Now we want to use this chart for the example of a gamma distribution. For this we need to implement a basic data model, which uses the observations directly as updates. We estimate the parameter of the gamma distribution via the method of moments (`Pofdata`). To run the chart we need the quantile function to calculate the estimates of the quantiles $f_{\alpha/2}$ and $f_{1-\alpha/2}$ (this appears in `xiofP`). Resampling is again parametric resampling under the assumed Gamma distribution (`resample`).

```
> X <- rgamma(100, scale = 3, shape = 2)
> modGammaBasic = structure(
+   list(
+     Pofdata = function(data){
+       list(scale = var(data)/mean(data),
+         shape = mean(data)^2/var(data),
+         n = length(data))
+     },
+     xiofP = function(P){
+       res <- P;
+       res$quant <- function(alpha)
+         qgamma(alpha, shape = P$shape, scale = P$scale);
+       res
+     },
+     resample = function(P) {
+       rgamma(P$n, shape = P$shape, scale = P$scale)
+     }
+   )
+ )
```

```

+         getcdfupdates = function(P, xi) {
+             function(x) pgamma(x, shape = P$shape, scale = P$scale)
+         },
+         updates = function(xi, data) data
+     ),
+     class = "SPCDataModel")
> chartAsym <- new("SPCShewAsym", model = modGammaBasic)
> SPCproperty(data = X, nrep = 50, chart = chartAsym,
+             property = "ARL", params = list(alpha = 0.01), quiet = TRUE)
90 % CI: A threshold defined by alpha=0.01 gives an in-control ARL of at
least 34.54.
Unadjusted result: 100
Based on 50 bootstrap repetitions.
> SPCproperty(data = X, nrep = 50,
+             property = "calARL", chart = chartAsym,
+             params = list(target = 100), quiet = TRUE)
90 % CI: A threshold of alpha=0.002869 gives an in-control ARL of at least
100.
Unadjusted result: 0.009998
Based on 50 bootstrap repetitions.

```

To show the advantage of the modular setup we now modify the data model to assume that the data is coming from an exponential distribution, in other words that the shape parameter of the gamma distribution is 1. We just need to redefine the function `PofData` to accomplish this.

```

> modExp = modGammaBasic
> modExp$Pofdata <- function(data){
+     list(scale = mean(data),
+          shape = 1,
+          n = length(data))
+ }
> chartAsymExp <- new("SPCShewAsym", model = modExp)
> X <- rexp(100)
> SPCproperty(data = X, nrep = 50, chart = chartAsymExp,
+             property = "ARL", params = list(alpha = 0.01), quiet = TRUE)
90 % CI: A threshold defined by alpha=0.01 gives an in-control ARL of at
least 84.08.
Unadjusted result: 100
Based on 50 bootstrap repetitions.
> SPCproperty(data = X, nrep = 50,
+             property = "calARL", chart = chartAsymExp,
+             params = list(target = 100), quiet = TRUE)
90 % CI: A threshold of alpha=0.007553 gives an in-control ARL of at least
100.
Unadjusted result: 0.009998
Based on 50 bootstrap repetitions.

```

Application to cardiac surgery data

In this section we illustrate use of the package with an application to a data set on the outcome of cardiac surgery from a UK centre for cardiac surgery over the period 1992-1998. These data were first analysed by [Steiner et al. \(2000\)](#) and have later been used for illustration by several authors (e.g. [Sego and Woodall, 2009](#); [Jones and Steiner, 2012](#); [Zhang et al., 2016](#)). A random subset of these data with some random noise added is available in the data frame `cardiacsurgery` in `spcadjust`. In this data frame the date of surgery, a surgeon number, the time until death if the patient died during the follow up time and the Parsonnet score of the patient is given. The Parsonnet score is a well established scoring system in cardiac surgery which combines a number of risk factors into a risk score for the patient. The data frame contains 5595 cases.

```
> data(cardiacsurgery)
```

Like [Steiner et al. \(2000\)](#) we will focus on the 30-day post-operative mortality rate and use a logistic regression model with Parsonnet score as covariate for taking into account the differences in risk between patients, and use a CUSUM for monitoring.

CUSUM for logistic regression models

We first describe the general set up for CUSUM monitoring with logistic regression models and then return to the cardiac surgery example. Assume we have n past in-control data $(Y_{-n}, X_{-n}), \dots, (Y_{-1}, X_{-1})$, where Y_i is a binary response variable and X_i is a corresponding vector of covariates. Suppose that in control $\text{logit}(P(Y_i = 1|X_i)) = X_i\beta$. A maximum likelihood estimate $\hat{\beta}$ is obtained based on the past data.

For detecting a change to $\text{logit}(P(Y_i = 1|X_i)) = \Delta + X_i\beta$, a CUSUM chart based on the cumulative sum of log likelihood ratios of the out-of-control versus in-control model can be defined by (Steiner et al., 2000)

$$S_t = \max(0, S_{t-1} + R_t), \quad S_0 = 0,$$

where

$$\exp(R_t) = \frac{\exp(\Delta + X_t\beta)^{Y_t} / (1 + \exp(\Delta + X_t\beta))}{\exp(X_t\beta)^{Y_t} / (1 + \exp(X_t\beta))} = \exp(Y_t\Delta) \frac{1 + \exp(X_t\beta)}{1 + \exp(\Delta + X_t\beta)}.$$

Like in the linear regression case we apply non-parametric bootstrap as described in Section *Risk-adjusted charts*.

Cardiac surgery data

The two first years of data, containing 1769 cases, are used for estimating the parameters of the logistic regression model. The effect of the Parsonnet score turns out to be non-linear on the logit scale, applying a square root transform of the score sorts out this. We thus set up data for estimating the chart parameters (phase I sample) and for running the chart (phase II sample) as follows:

```
> #Use dead within 30 days as response
> dead30 <- as.numeric(cardiacsurgery$time <= 30)
> #Use the two first years of data as phase I sample
> phaseone <- cardiacsurgery$date <= 730
> estdata <- data.frame(y = dead30[phaseone],
+                       x = sqrt(cardiacsurgery$Parsonnet[phaseone]))
> #Use the five last years of data as phase II sample
> phasetwo <- !phaseone
> rundata <- data.frame(y = dead30[phasetwo],
+                       x = sqrt(cardiacsurgery$Parsonnet[phasetwo]),
+                       z = cardiacsurgery$surgeon[phasetwo],
+                       year = (cardiacsurgery$date[phasetwo]-730)/365)
```

Next we set up charts for monitoring against roughly a doubling and a halving of the mortality rate, respectively. With a baseline rate of 6.1% this corresponds to $\Delta = 0.75$ and $\Delta = -0.75$.

```
> chartlogregd <-
+ new("SPCCUSUM", model = SPCModellogregLikRatio(Delta = 0.75, formula = "y~x"))
> chartlogregh <-
+ new("SPCCUSUM", model = SPCModellogregLikRatio(Delta = -0.75, formula = "y~x"))
```

For calculating the thresholds we specify an in control ARL of 10000, i.e. a false alarm should on average only occur once per 10000 procedures.

```
> cald <- SPCproperty(data = estdata, chart = chartlogregd, property = "calARL",
+                    nrep = 50, params = list(target = 10000, gridpoints = 250),
+                    parallel = Inf)
> cald
90 % CI: A threshold of 6.157 gives an in-control ARL of at least 10000.
Unadjusted result: 5.065
Based on 50 bootstrap repetitions.
> calh <- SPCproperty(data = estdata, chart = chartlogregh, property = "calARL",
+                    nrep = 50, params = list(target = 10000, gridpoints = 250),
+                    parallel = Inf)
> calh
90 % CI: A threshold of 6.271 gives an in-control ARL of at least 10000.
Unadjusted result: 4.469
Based on 50 bootstrap repetitions.
```

The option `parallel = Inf` speeds up the bootstrap, but this option must be skipped if the code is run in an environment which does not support parallel processing.

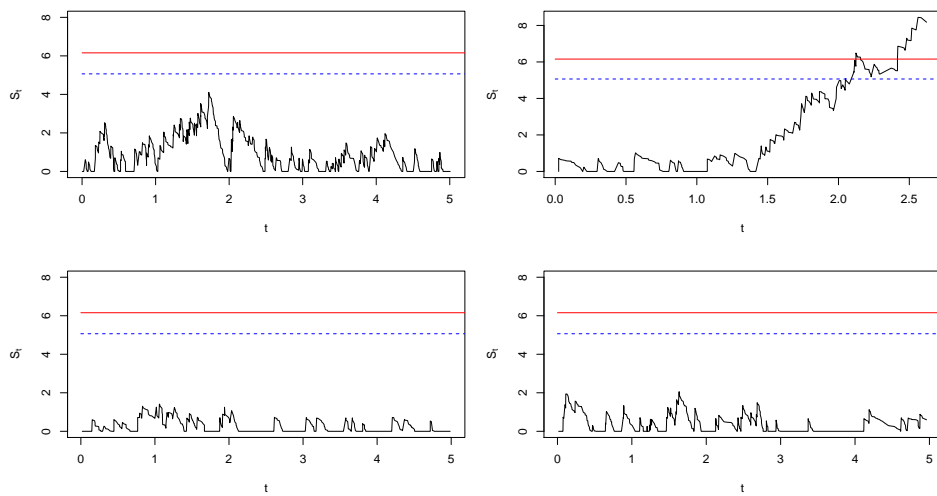


Figure 6: CUSUM charts based on logistic regression model with estimated parameters monitoring against increased mortality. Individual charts for four surgeons.

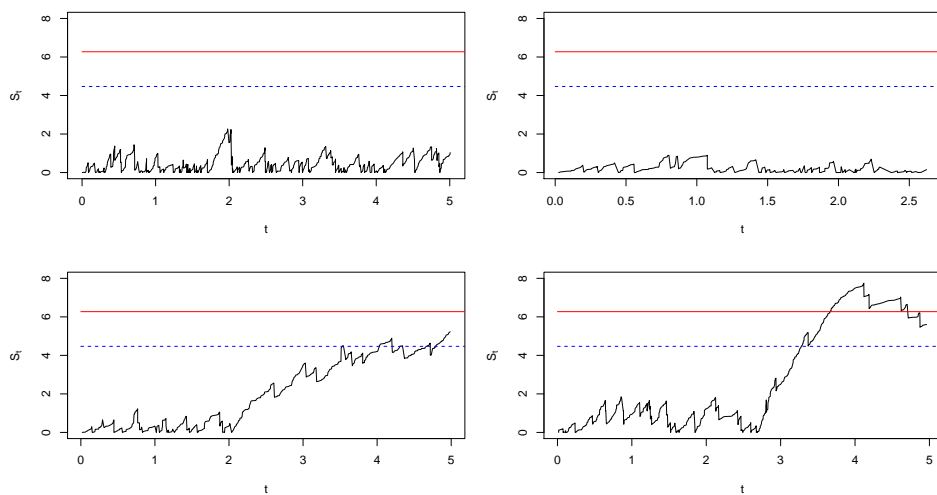


Figure 7: CUSUM charts based on logistic regression model with estimated parameters monitoring against decreased mortality. Individual charts for four surgeons.

Assuming that the distribution of Parsonnet scores is roughly the same for the patients each surgeon receives, and that the distribution remains approximately the same in the remainder of the period as in the first two years, the thresholds calculated above can be used for running individual charts for each of the surgeons. Notice that in such a setting where several charts are run with the same estimated parameters a threshold adjustment which achieves a guaranteed conditional performance is particularly relevant (Gandy and Kvaløy, 2013).

The resulting CUSUM plots for four of the surgeons are shown in Figures 6 and 7. In Figures 6 the CUSUM for the second surgeon starts to increase after a while and passes both the unadjusted and the adjusted threshold. This could e.g. be due to this surgeon starting to receive more difficult cases, not sufficiently accounted for by the adjustment for Parsonnet score.

For the monitoring against decreased mortality in Figures 7 there is a signal for one of the surgeons, indicating better survival than explained by the adjustment for Parsonnet score. The CUSUM for the third surgeon crosses the unadjusted threshold, but not the adjusted and is thus not regarded as a true signal.

Bibliography

- W. Albers and W. C. M. Kallenberg. Are estimated control charts in control? *Statistics*, 38:67–79, 2004. URL <https://doi.org/10.1080/02669760310001619369>. [p458]
- W. Albers and W. C. M. Kallenberg. New corrections for old control charts. *Quality Engineering*, 17:467–473, 2005. URL <https://doi.org/10.1081/QEN-200063498>. [p458]
- W. Albers, W. C. M. Kallenberg, and S. Nurdianti. Exceedance probabilities for parametric control charts. *Statistics*, 39:429–443, 2005. URL <https://doi.org/10.1080/02331880500310181>. [p458]
- P. Biswas and J. D. Kalbfleisch. A risk-adjusted CUSUM in continuous time based on the Cox model. *Statistics in Medicine*, 27:3382–3406, 2008. URL <https://doi.org/10.1002/sim.3216>. [p458]
- A. Bottle and P. Aylin. Intelligent information: a national system for monitoring clinical performance. *Health Services Research*, 43:10–31, 2008. URL <https://doi.org/10.1111/j.1475-6773.2007.00742.x>. [p458]
- G. Capizzi and G. Masarotto. Bootstrap-based design of residual control charts. *IIE Transactions*, 41:275–286, 2009. URL <https://doi.org/10.1080/07408170802120059>. [p458]
- C. W. Champ and L. A. Jones-Farmer. Properties of multivariate control charts with estimated parameters. *Sequential Analysis*, 26:153–169, 2007. URL <https://doi.org/10.1080/07474940701247040>. [p458]
- S. Chatterjee and P. Qiu. Distribution-free cumulative sum control charts using bootstrap-based control limits. *Annals of Applied Statistics*, 3:349–369, 2009. URL <https://doi.org/10.1214/08-AOAS197>. [p458]
- H. Chen and W.-L. Kuo. Comparison of the symmetric and asymmetric control limits for \bar{X} and R charts. *Computers and Industrial Engineering*, 59:903–910, 2010. URL <https://doi.org/10.1016/j.cie.2010.08.021>. [p469]
- M. Flores. *Qcr: Quality Control Review*, 2016. URL <https://CRAN.R-project.org/web/package=qcr>. R package version 1.0. [p458]
- M. Fouladirad, A. Grall, and L. Dieulle. On the use of on-line detection for maintenance of gradually deteriorating systems. *Reliability Engineering and System Safety*, 93:1814–1820, 2008. URL <https://doi.org/10.1016/j.ress.2008.03.020>. [p458]
- M. Frisén, editor. *Financial Surveillance*. John Wiley & Sons, 2008. [p458]
- A. Gandy and J. T. Kvaløy. Guaranteed conditional performance of control charts via bootstrap methods. *Scandinavian Journal of Statistics*, 40:647–668, 2013. URL <https://doi.org/10.1002/sjos.12006>. [p458, 460, 461, 462, 474]
- A. Gandy and J. T. Kvaløy. *Spcadjust: Functions for Calibrating Control Charts*, 2016. URL <https://CRAN.R-project.org/web/package=spcadjust>. R package version 1.1. [p458]
- A. Gandy, J. T. Kvaløy, A. Bottle, and F. Zhou. Risk-adjusted monitoring of time to event. *Biometrika*, 97:375–388, 2010. URL <https://doi.org/10.1093/biomet/asq004>. [p458]

- O. Grigg and V. Farewell. An overview of risk-adjusted charts. *Journal of the Royal Statistical Society A*, 167:523–539, 2004. URL <https://doi.org/10.1111/j.1467-985X.2004.0apm2.x>. [p458, 461]
- D. M. Hawkins and D. H. Olwell. *Cumulative Sum Charts and Charting for Quality Improvement*. Springer-Verlag, 1998. [p468]
- L. Horváth, M. H. a, P. Kokoszka, and J. Steinebach. Monitoring changes in linear models. *Journal of Statistical Planning and Inference*, 126:225–251, 2004. URL <https://doi.org/10.1016/j.jspi.2003.07.014>. [p461]
- W. A. Jensen, L. A. Jones-Farmer, C. W. Champ, and W. H. Woodall. Effects of parameter estimation on control chart properties: A literature review. *Journal of Quality Technology*, 38:349–364, 2006. [p458]
- L. A. Jones. The statistical design of EWMA control charts with estimated parameters. *Journal of Quality Technology*, 34:277–288, 2002. [p458]
- L. A. Jones, C. W. Champ, and S. E. Rigdon. The run length distribution of the CUSUM with estimated parameters. *Journal of Quality Technology*, 36:95–108, 2004. [p458]
- M. A. Jones and S. H. Steiner. Assessing the effect of estimation error on risk-adjusted CUSUM chart performance. *International Journal for Quality in Health Care*, 24:176–181, 2012. URL <https://doi.org/10.1093/intqhc/mzr082>. [p458, 471]
- S. Knot. *Spc: Statistical Process Control – Collection of Some Useful Functions*, 2016. URL <https://CRAN.R-project.org/web/package=spc>. R package version 0.5.3. [p458]
- N. A. Saleh, M. A. Mahmoud, M. J. Keefe, and W. H. Woodall. The difficulty in designing Shewhart \bar{X} and X control charts with estimated parameters. *Journal of Quality Technology*, 47:127–138, 2015. [p458]
- M. Salmon, D. Schumacher, and M. Höhle. Monitoring count time series in R: Aberration detection in public health surveillance. *Journal of Statistical Software*, 70(10):1–35, 2016. URL <https://doi.org/10.18637/jss.v070.i10>. [p458]
- E. Santos-Fernández. *Multivariate Statistical Quality Control Using R*, volume 14. Springer-Verlag, 2013. ISBN 9781461454533. URL <https://doi.org/10.1007/978-1-4614-5453-3>. [p458]
- W. Schmid, editor. *80 Years of the Control Chart*. *Sequential Analysis*, 26(2), 2007a. [p458]
- W. Schmid, editor. *80 Years of the Control Chart*. *Sequential Analysis*, 26(3), 2007b. [p458]
- L. Scrucca. Qcc: An R package for quality control charting and statistical process control. *R News*, 4: 11–17, 2004. [p458]
- L. Scrucca. *IQCC: Improved Quality Control Charts*, 2014. URL <https://CRAN.R-project.org/web/package=IQCC>. R package version 0.6. [p458]
- L. H. Sego and W. H. Woodall. Risk-adjusted monitoring of survival times. *Statistics in Medicine*, 28: 1386–1401, 2009. URL <https://doi.org/10.1002/sim.3546>. [p471]
- S. H. Steiner, R. J. Cook, V. T. Farewell, and T. Treasure. Monitoring surgical performance using risk-adjusted cumulative sum charts. *Biostatistics*, 1:441–452, 2000. URL <https://doi.org/10.1093/biostatistics/1.4.441>. [p471, 472]
- Z. G. Stoumbos, M. R. Reynolds Jr., Ryan, T. P., and W. H. Woodall. The state of statistical process control as we proceed into the 21st century. *Journal of the American Statistical Association*, 95:992–998, 2000. URL <https://doi.org/10.1080/01621459.2000.10474292>. [p458]
- W. H. Woodall. The use of control charts in health-care and public-health surveillance. *Journal of Quality Technology*, 38:89–134, 2006. With discussion. [p458]
- M. Zhang, Y. Xu, Z. He, and X. Hou. The effect of estimation error on risk-adjusted survival time CUSUM chart performance. *Quality and Reliability Engineering International*, 32:1445–1452, 2016. URL <https://doi.org/10.1002/qre.1849>. [p458, 471]
- Y. Zhang, P. Castagliola, Z. Wu, and M. B. C. Khoo. The synthetic \bar{X} chart with estimated parameters. *IIE Transactions*, 43:676–687, 2011. URL <https://doi.org/10.1080/0740817X.2010.549547>. [p458]
- W. Zhu and C. Park. edcc: An R package for the economic design of the control chart. *Journal of Statistical Software*, 52(9):1–24, 2013. URL <https://doi.org/10.18637/jss.v052.i09>. [p458]

Axel Gandy
Department of Mathematics
Imperial College London
London, SW7 2AZ, UK
a.gandy@imperial.ac.uk

Jan Terje Kvaløy
Department of Mathematics and Natural Sciences
University of Stavanger
4036 Stavanger, Norway
jan.t.kvaloy@uis.no

Weighted Effect Coding for Observational Data with `wec`

by Rense Nieuwenhuis, Manfred te Grotenhuis, Ben Pelzer

Abstract Weighted effect coding refers to a specific coding matrix to include factor variables in generalised linear regression models. With weighted effect coding, the effect for each category represents the deviation of that category from the weighted mean (which corresponds to the sample mean). This technique has particularly attractive properties when analysing observational data, that commonly are unbalanced. The `wec` package is introduced, that provides functions to apply weighted effect coding to factor variables, and to interactions between (a.) a factor variable and a continuous variable and between (b.) two factor variables.

Introduction

Weighted effect coding is a type of dummy coding to facilitate the inclusion of categorical variables in generalised linear models (GLM). The resulting estimates for each category represent the deviation from the weighted mean. The weighted mean equals the arithmetic mean or sample mean, that is the sum of all scores divided by the number of observations. As we will show, weighted effect coding has important advantages over traditional effect coding if unbalanced data are used (i.e. with categories holding different numbers of observations), which is common in the analysis of observational data. We describe weighted effect coding for categorical variables and their interactions with other variables. Basic weighted effect coding was first described in 1972 (Sweeney and Ulveling, 1972) and recently updated to include weighted effect interactions between categorical variables (Grotenhuis et al., 2017b,a). In this paper we develop the interaction between weighted effect coded categorical variables and continuous variables. All software is available in the `wec` package.

Treatment, effect, and weighted effect coding

Weighted effect coding is one of various ways to include categorical (i.e., nominal and ordinal) variables in generalised linear models. As this type of variables is not continuous, so-called dummy variables have to be created first which represent the categories of the categorical variable. In R, categorical variables are handled by factors, to which different contrasts can be assigned. For unordered factors, the default is dummy or treatment coding. With treatment coding, each category in the factor variable is tested against a preselected reference category. This coding can be specified with `contr.treatment`. Several alternatives are available, including orthogonal polynomials (default for ordered factors and set with `contr.poly`), helmert coding to compare each category to the mean of all subsequent categories (`contr.helmert`), and effect coding (`contr.sum`).

In effect coding (also known as deviation contrast or ANOVA coding), parameters represent the deviation of each category from the grand mean across all categories (i.e., the sum of arithmetic means in all categories divided by the number of categories). To achieve this, the sum of all parameters is constrained to 0. This implies that the possibly different numbers of observations in categories is not taken into account. In *weighted* effect coding, the parameters represent the deviation of each category from the *sample mean*, corresponding to a constraint in which the *weighted* sum of all parameters is equal to zero. The weights are equal to the number of observations per category.

The differences between treatment coding, effect coding, and weighted effect coding are illustrated in Figure 1, showing the mean wage score for 4 race categories in the USA. The grey circles represent the numbers of observations per category, with whites being the largest category. In treatment coding, the parameters for the Black, Hispanic and Asian populations reflect the mean wage differences from the mean wage in the white population that serves here as the reference category. The dotted double-headed arrow in Figure 1 represents the effect for Blacks based on treatment coding, with whites as the reference category. In effect coding, the reference is none of the four racial categories, but the grand mean. This mean is the sum of all four (arithmetic) mean wages divided by 4, and amounts to 49,762 and is shown as the dashed horizontal line in Figure 1. The effect for Blacks then is the difference between their mean wage score (37,664) and the grand mean, represented by the dashed double-headed arrow and amounts to $(37,664 - 49,762) = -12,096$. Weighted effect coding accounts for the number of observations per category, and thus weighs the mean wages of all categories first by the different number of observations per category. Because whites outnumber all other other categories the weighted (sample) mean ($= 52,320$) is much larger than the unweighted (grand) mean, and is represented by the horizontal continuous line in Figure 1. As a consequence, the effect for

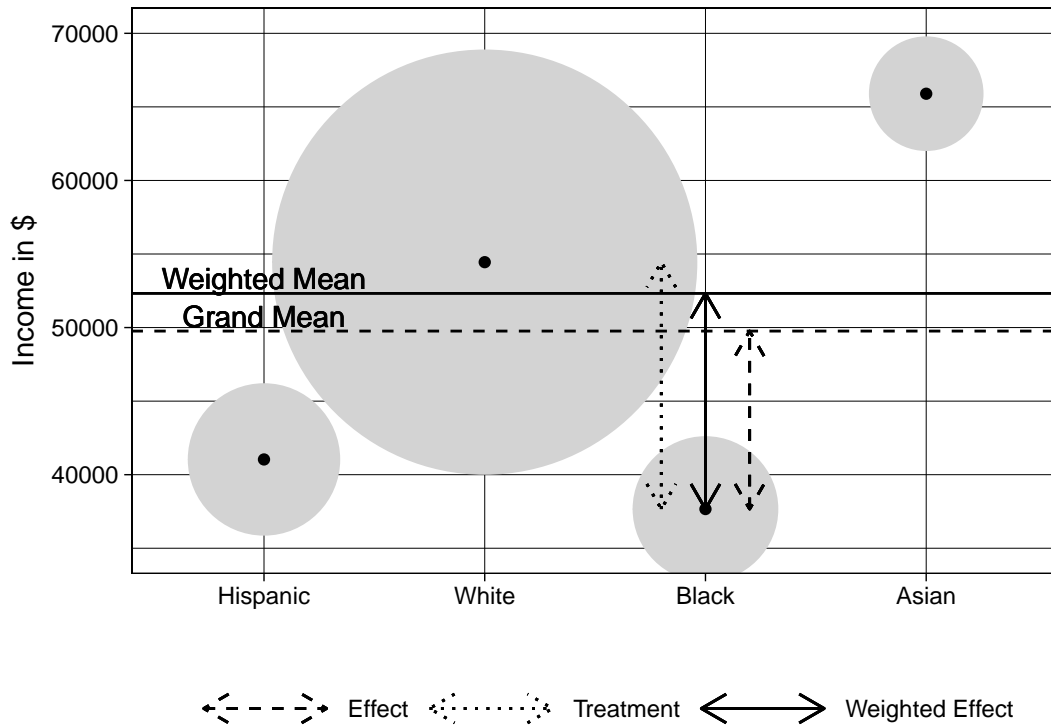


Figure 1: Illustration of treatment coding, effect coding, and weighted effect coding

Blacks now is much stronger ($37,664 - 52,320 = -14,654$) as represented by the (vertical) continuous double-headed arrow.

If the data are balanced, meaning that all categories have the same number of observations, the results for effect coding and weighted effect coding are identical. With unbalanced data, such as typically is the case in observational studies, weighted effect coding offers a number of interesting features that are quite different from those obtained by unweighted effect coding. First of all, in observational data the sample mean provides a natural point of reference. Secondly, the results of weighted effect coding are not sensitive to decisions on how observations were assigned to categories: when categories are split or combined, the grand mean is likely to shift as it depends on the means within categories. In weighted effect coding the sample mean of course remains unchanged. Therefore, combining or splitting *other* categories does not change the effects of categories that were not combined or split. Finally, weighted effect coding allows for an interpretation that is complementary to treatment coding, and seems particularly relevant when comparing datasets from different populations (e.g., from different countries, or time-periods): the effects represent how deviant a specific category is from the sample mean, while accounting for differences in the composition between populations. Looking at Figure 1, this would allow for the finding that the Black population would have become more deviant over time in a situation where the whites grew in numbers (thus shifting the weighted mean upwards) while the wage gap between Blacks and whites remained constant (the dotted line, as would be estimated with treatment coding).

The coding matrix for weighted effect coding is shown in Table 1. In effect coding, the columns of the coding matrix would all have summed to 0. This can be seen in the first example of the next section. The coding matrix for *weighted* effect coding is based on the restriction that the columns multiplied by the proportions of the respective categories sum to 0. In other words, if the values in each cell of the

	Hispanic	Black	Asian
Hispanic	1	0	0
Black	0	1	0
Asian	0	0	1
White	$-(n_{hispanic}/n_{white})$	$-(n_{black}/n_{white})$	$-(n_{asian}/n_{white})$

Table 1: Coding Matrix Weighted Effect Coding

coding matrix in Table 1 are weighted by the relative number (or proportion) in each category, each column sums to 0.

Examples

This article introduces the `wec` package and provides functions to obtain coding matrices based on weighted effect coding. The examples in this article are based on the PUMS data.frame, which has data on wages, education, and race in the United States in 2013. It is a subset of 10,000 randomly sampled observations, all aged 25 or over and with a wage larger than zero, originating from the PUMS 2013 dataset (Census, 2013). Because the calculation of weighted effect coded variables involves numbers of observations, it is important to first remove any relevant missing values (i.e., list-wise deletion), before defining the weighted effect coded variables.

```
> library(wec)
> data(PUMS)
```

We first demonstrate the use of standard effect coding, which is built into base R, to estimate the effects of race on wages. To ensure that the original race variable remains unaltered, we create a new variable 'race.effect'. This is a factor variable with 4 categories ('Hispanic', 'Black', 'Asian', and 'White'). Effect coding is applied using the `contr.sum()` function. 'White' is selected as the omitted category by default. Then, we use this new variable in a simple, OLS regression model. This is shown below:

```
> PUMS$race.effect <- factor(PUMS$race)
> contrasts(PUMS$race.effect) <- contr.sum(4)
> contrasts(PUMS$race.effect)
```

	[,1]	[,2]	[,3]
Hispanic	1	0	0
Black	0	1	0
Asian	0	0	1
White	-1	-1	-1

```
> m.effect <- lm(wage ~ race.effect, data=PUMS)
> summary(m.effect)$coefficients
      Estimate Std. Error t value Pr(>|t|)
(Intercept)   49762      954    52.2 0.0e+00
race.effect1  -8724     1649    -5.3 1.3e-07
race.effect2 -12096     1702    -7.1 1.3e-12
race.effect3  16135     2042     7.9 3.0e-15
```

The results of regressing wages on the effect coded race variable (only the fixed effects are shown above) indicate that the grand mean of wages is 49,762. In Figure 1 this grand mean was shown as the horizontal, dashed line. This is the grand (unweighted) mean of the average (arithmetic) wages among Hispanics, Blacks, Asians, and white Americans. The mean wage among Blacks ('race.effect2', refer to the coding matrix to see which category received which label) is, on average, 12,096 dollar lower than this grand mean. This was shown as the dashed double-headed arrow in Figure 1. The wages of Asians ('race.effect3'), on the other hand, are on average 16,135 dollar higher than the grand mean.

We already saw in Figure 1 that not only the average wages vary across races, but also that the number of Hispanics, Blacks, Asians, and whites are substantially different. As these observational data are so unbalanced, the grand mean is not necessarily the most appropriate point of reference. Instead, the sample (arithmetic) mean may be preferred as a point of reference. To compare and test the deviations of all four mean wages from the sample mean, weighted effect coding has to be used:

```
> PUMS$race.wec <- factor(PUMS$race)
> contrasts(PUMS$race.wec) <- contr.wec(PUMS$race.wec, "White")
> contrasts(PUMS$race.wec)
```

	Hispanic	Black	Asian
Hispanic	1.00	0.00	0.000
Black	0.00	1.00	0.000
Asian	0.00	0.00	1.000
White	-0.12	-0.11	-0.069

```
> m.wec <- lm(wage ~ race.wec, data=PUMS)
```

```
> summary(m.wec)$coefficients
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    52320         587    89.1  0.0e+00
race.wecHispanic -11282        1810    -6.2  4.8e-10
race.wecBlack   -14654        1905    -7.7  1.6e-14
race.wecAsian   13577         2484     5.5  4.7e-08
```

The example above again creates a new variable ('race.wec') and uses the new `contr.wec()` function to assign a *weighted* effect coding matrix. Unlike many other functions for contrasts in R, `contr.wec()` requires that not only the omitted category is specified, but also the specification of the factor variable for which the coding matrix is computed. The reason for this is that, as seen in Table 1, to calculate a weighted effect coded matrix, information on the number of observations within each category is required. The coding matrix now shows a (negative) weight (which is the ratio between the number of observations in category *x* and the omitted category 'Whites') for the omitted category, which was -1 in the case of effect coding.

In the regression analysis, the intercept now represents the sample mean and all other effects represent deviations from that sample mean. This corresponds to the continuous double-headed arrow and line in Figure 1. For instance, Blacks earn on average 14,654 dollars less compared to the sample mean. To be able to test how much whites' mean wage differs from the sample mean, the omitted category must be changed and subsequently a new variable is to be used in an updated regression analysis:

```
> PUMS$race.wec.b <- PUMS$race.wec
> contrasts(PUMS$race.wec.b) <- contr.wec(PUMS$race.wec, "Black")
> m.wec.b <- lm(wage ~ race.wec.b, data=PUMS)
> summary(m.wec.b)$coefficients
```

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    52320         587    89.1  0.0e+00
race.wec.bHispanic -11282        1810    -6.2  4.8e-10
race.wec.bAsian   13577         2484     5.5  4.7e-08
race.wec.bWhite   2128          325     6.5  6.5e-11
```

Here, the omitted category was changed to Blacks. Note that the intercept as well as the estimates for Hispanics and for Asians did not change. This is unlike treatment coding, where each estimate represents the deviation from the omitted category (in treatment coding: the reference category). The new estimate shows that whites earn 2,128 dollar more than the mean wage in the sample. In the remainder of this article we use 'White' as the omitted category by default, but in all analyses the omitted category can be changed.

Next, we control the results for respondents' level of education using a continuous variable (which is mean centred to keep the intercept at 52,320).

```
> m.wec2 <- lm(wage ~ race.wec + education.int, data=PUMS)
> summary(m.wec2)$coefficients
```

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    52320         560    93.4  0.0e+00
race.wecHispanic -4955        1738    -2.9  4.4e-03
race.wecBlack   -11276        1820    -6.2  6.0e-10
race.wecAsian   5151         2385     2.2  3.1e-02
education.int   9048          287    31.5  7.9e-208
```

The results show that one additional point of education is associated with an increase in wages of 9,048. This represents the average increase of wages due to education in the sample while controlling for race. The estimates for the categories of race again represent the deviation from the sample mean *controlled* for education. When more control variables are added, the weighted effect coded estimates still represent the deviation from the sample mean, but now controlled for all other variables as well. Comparing these estimates to those from the model without the control for education suggests that educational differences partially account for racial wage differences. In the next section, we discuss how weighted effect coded factor variables can be combined with interactions, to test whether the wage returns of educational attainment vary across race.

Interactions

Weighted effect coding can also be used in generalised linear models with interaction effects. The weighted effect coded interactions represent the additional effects over and above the main effects

obtained from the model without these interactions. This was recently shown for an interaction between two weighted effect coded categorical variables (Grotenhuis et al., 2017a). In this paper we address the novel interaction between weighted effect coded categorical variables and a continuous variable. In the previous section a positive effect (9,048) of education on wage was found. The question is whether this effect is equally strong for all four racial categories.

With treatment coding, an interaction would represent how much the effect of (for instance) education for one category differs from the educational effect in another category that was chosen as reference. With effect coding, the interaction terms represent how much the effect of (for instance) education for a specific category differs from the unweighted main effect (which here happens to be 8,405). Because the data are unbalanced, weighted effect coding is considered here an appropriate parameterisation. In weighted effect coded interactions the point of reference is the main effect in the same model but without the interactions.

In our case this educational main effect on wage is 9,048 (see Figure 2), which we calculated in the example above. Let's assume we already know, as will be confirmed in later examples, that the estimates for the effect of education on wages among whites is 9,461, among Hispanics 5,782, among Asians 12,623, and finally among Blacks the effect is 5,755. The weighted effect coded interactions then are, respectively, $9,461 - 9,048 = 413$; $5,782 - 9,048 = -3,266$; $12,623 - 9,048 = 3,575$; and $5,755 - 9,048 = -3,293$. These estimates represent how much the education effect for each group differs from the main effect of education in the sample.

With weighted effect coded interactions, one can obtain these estimates simultaneously with the mean effect of education. To do so, a coding matrix has to be calculated. This coding matrix is based on the restriction that if the above-mentioned effects are multiplied by the sum of squares of education within each category, the sum of these multiplications is zero. This is the weighted effect coded restriction for interactions.

The sum of squares (SS) of the continuous variable x (education) for level j of the categorical variable (race) is calculated as:

$$SS_j = \sum_{i=1}^I (x_{ij} - \bar{x}_j)^2 \tag{1}$$

where, for the example, x_{ij} denotes the education of a person i in race j , I denotes the total number of people in race j and \bar{x}_j denotes the mean of education for people in race j .

To impose this restriction we replaced the weights in Table 1 by the ratio between two sums of squares to obtain a new coding matrix (see Table 2) (Lammers, 1991). The denominator of this ratio is the sum of squares of education among the omitted category. If we multiply this coding matrix with the mean centred education variable, then we get three interaction variables, and the estimates for these variables reflect the correct deviations from the main education effect together with the correct statistical tests. To have the intercept unchanged, we finally mean centred the new interaction variables within each category of race.

In previous approaches to interactions with weighted effect coding (West et al., 1996; Aguinis, 2004), it was not possible to have the effects of the first order model unchanged. This is because a restriction to the coding matrix was used based on the number of observations rather than on the sum of squares used here.

An attractive interpretation of interaction terms is provided: as the (multiplicative) interaction terms are orthogonal to the *main* effects of each category, these main effects remain unchanged upon adding the interaction terms to the model. The interaction terms represent, and test the significance of, the additional effect to the main effects.

The logic of interactions between weighted effect coded dummies and a continuous variable is demonstrated in Figure 2. The dashed blue and red lines represent the effects of education for Blacks and whites, respectively (Hispanics and Asians not shown here). The dashed black line represents the effect of education that is the average of the effects among the four racial categories. This is the effect of education one would estimate if effect coding was used to estimate the interaction, and the differences in slopes between this reference and each racial categories would be the interaction parameters. However, the observations of whites influence the height of the average effect of education in the sample to a larger extent than the Blacks, due to their larger sum of squares. Therefore, the weighted effect of education, shown as the continuous line, is a more useful reference. The sum of squares are represented in Figure 2 by grey squares, and are distinct from the grey circles representing frequencies in Figure 1. The sum of squares pertain to the complete regression slope, and therefore the position of the grey squares was chosen arbitrarily at the center of the x-axis.

Finally, we briefly address the interaction between two weighted effect coded categorical variables. Unlike dummy coding and effect coding, the interaction variables are not simply the multiplication of the two weighted effect coded variables. Instead, partial weights are assigned to the interaction

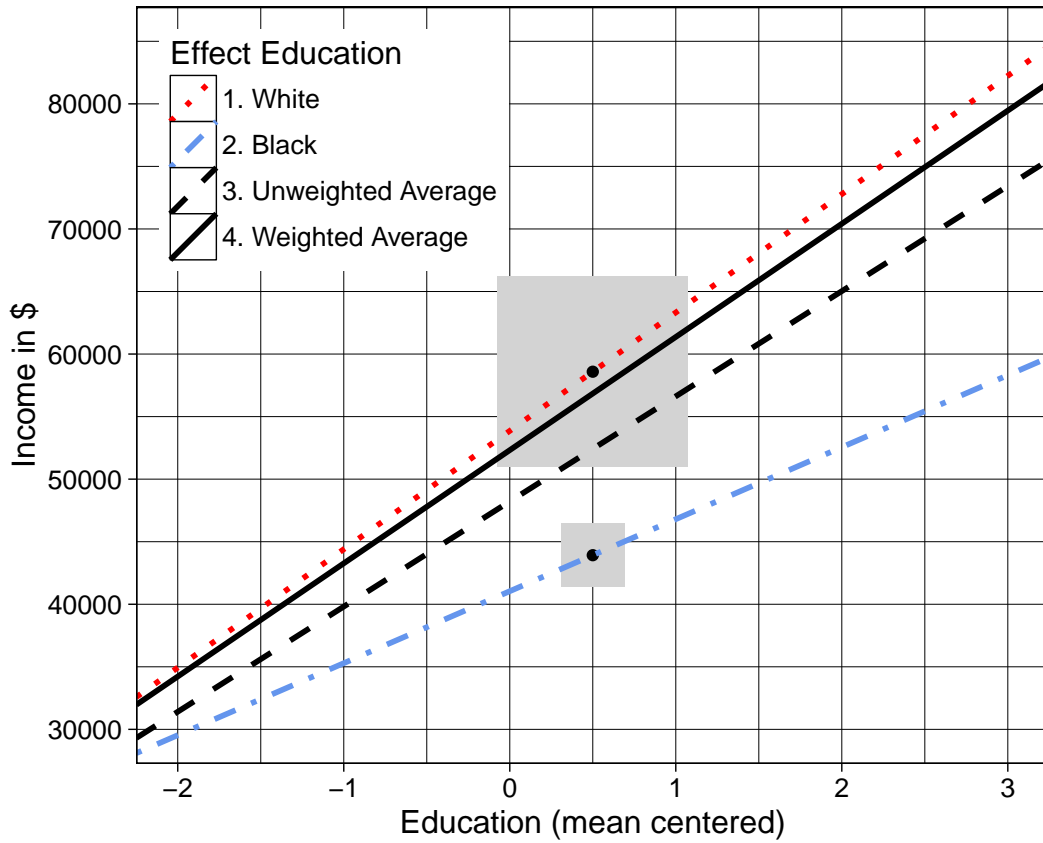


Figure 2: Illustration of interacting weighted effect coding and continuous variable.
 Note: Grey squares represent sum of squares (position on x-axis was chosen arbitrarily).

	Hispanic	Black	Asian
Hispanic	1	0	0
Black	0	1	0
Asian	0	0	1
White	$-(SS_{hispanic}/SS_{white})$	$-(SS_{black}/SS_{white})$	$-(SS_{asian}/SS_{white})$

Table 2: Coding Matrix for interaction Factor with Weighted Effect Coding and Continuous variable

	Degree	Main Effects			Interaction Effects		
		Hispanic	Black	Asian	Degree × Hispanic	Degree × Black	Degree × Asian
HS & H	$-(n_d/n_{hs})$	1	0	0	$-(n_{d,h}/n_{hs,h})$	0	0
HS & B	$-(n_d/n_{hs})$	0	1	0	0	$-(n_{d,b}/n_{hs,b})$	0
HS & A	$-(n_d/n_{hs})$	0	0	1	0	0	$-(n_{d,a}/n_{hs,a})$
HS & W	$-(n_d/n_{hs})$	$-(n_h/n_w)$	$-(n_b/n_w)$	$-(n_a/n_w)$	$(n_{d,h}/n_{hs,w})$	$(n_{d,b}/n_{hs,w})$	$(n_{d,a}/n_{hs,w})$
D & H	1	1	0	0	1	0	0
D & B	1	0	1	0	0	1	0
D & A	1	0	0	1	0	0	1
D & W	1	$-(n_h/n_w)$	$-(n_b/n_w)$	$-(n_a/n_w)$	$-(n_{d,h}/n_{d,w})$	$-(n_{d,b}/n_{d,w})$	$-(n_{d,a}/n_{d,w})$

Table 3: Coding Matrix for interaction two factor variables with Weighted Effect Coding

variables to obtain main effects that equal the effects from the model without these interactions (see Table 3 for the weights, for in-depth matrix information about how to create these partial weights please visit <http://ru.nl/sociology/mt/wec/downloads/>). The orthogonal interaction effects in our example denote the extra wage over and above the mean wages found in the model without these interactions, no matter whether the data are unbalanced or not. In case the data are completely balanced, the estimates from weighted effect coding are equal to those from effect coding, but they can

be quite different in effect size and associated t-values when the data are unbalanced.

Examples of interactions

To demonstrate interactions that include weighted effect coded factor variables, we continue our previous example. For these interactions, the functions in the `wec` package deviate a little from standard R conventions. This is a direct result of how weighted effect coding works. With many forms of dummy coding, interaction variables can be created by simply multiplying the values of the two variables that make up the interaction. This is not true for weighted effect coding, as the coding matrix for the interaction is a function of the numbers of observations of the two variables that interact. So, instead of multiplying two variables in the specification of the regression model in typical R-fashion, a new, third, variable is created prior to specifying the regression model and then added. Here, we refer to these additional variables as the 'interaction' variable.

Interaction variables for interacting weighted effect coded factor variables are produced by the `wec.interact()` function. The first variable entered ('x1') must be a weighted effect coded factor variable. The second ('x2') can either be a continuous variable or another weighted effect coded factor variable. By default, this function returns an object containing one column for each of the interaction variables required. However, by specifying `output.contrasts = TRUE`, the coding matrix (see Table 2) is returned:

```
> wec.interact(PUMS$race.wec, PUMS$education.int, output.contrasts = TRUE)
```

```
  [,1] [,2] [,3]
1  1.0  0.000  0.000
2  0.0  1.000  0.000
3  0.0  0.000  1.000
4 -0.1 -0.098 -0.066
```

The example above shows the coding matrix for interacting the (weighted effect coded) race variable with the continuous education variable. The omitted category is, again, 'Whites' (category 4), and the coding matrix shows the ratio of sum of squares as was defined in Table 2. To include this in the regression analysis, a new factor variable is created:

```
> PUMS$race.educint <- wec.interact(PUMS$race.wec, PUMS$education.int)
> m.wec.educ <- lm(wage ~ race.wec + education.int + race.educint, data=PUMS)
> summary(m.wec.educ)$coefficients
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	52320	559	93.5	0.0e+00
race.wecHispanic	-4955	1736	-2.9	4.3e-03
race.wecBlack	-11276	1817	-6.2	5.7e-10
race.wecAsian	5151	2381	2.2	3.1e-02
education.int	9048	287	31.6	2.3e-208
race.educintinteractHispanic	-3266	977	-3.3	8.3e-04
race.educintinteractBlack	-3293	990	-3.3	8.8e-04
race.educintinteractAsian	3575	1217	2.9	3.3e-03

The `wec.interact` function is now called without the `output.contrasts = TRUE` option. The first specification is the factor variable and the second term is the continuous variable. The results are stored in a new variable. This new interaction variable is entered into the regression model *in addition* to the variables with the main effects for race and education.

The results show that the returns to education, in terms of wages, for Hispanics and Blacks are lower than the average returns to education in the sample, and the returns to education are higher among Asians than it is in the sample as a whole. Note that without additional control variables, all effects for race and for education, as well as the estimate for the intercept, remained unchanged compared to previous examples after including the (weighted effect coded) interaction variable.

Note that if one wants to estimate the effects and standard errors for the omitted category, in this case 'Whites', not only the contrasts for the categorical variable need to be changed (as demonstrated above), but also the interaction variable needs to be updated.

Below, we specify the interaction between the race variable with a factor variable differentiating respondents who have a high school diploma and those who have a higher degree. Of course, both are weighted effect coded:

```
> PUMS$education.wec <- PUMS$education.cat
> contrasts(PUMS$education.wec) <- contr.wec(PUMS$education.cat, "High school")
```

```
> PUMS$race.educat <- wec.interact(PUMS$race.wec, PUMS$education.wec)
> m.wec.educwec <- lm(wage ~ race.wec + education.wec + race.educat, data=PUMS)
> summary(m.wec.educwec)$coefficients
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	52320	569	92.0	0.0e+00
race.wecHispanic	-6645	1764	-3.8	1.7e-04
race.wecBlack	-11738	1849	-6.3	2.3e-10
race.wecAsian	7528	2419	3.1	1.9e-03
education.wecDegree	14343	572	25.1	1.6e-134
race.educatx1Hispanic:x2Degree	-7674	2441	-3.1	1.7e-03
race.educatx1Black:x2Degree	-6682	2252	-3.0	3.0e-03
race.educatx1Asian:x2Degree	4022	1536	2.6	8.8e-03

We created a new categorical variable ‘education.wec’ and assigned a coding matrix based on weighted effect coding, with ‘High school’ as the omitted category. The results show that respondents with a degree on average earn 14,343 dollar more than the sample average (52,320). Hispanics benefit 7,674 dollar less from having a degree compared to the average benefit of a degree, while Asians benefit 4,022 dollar more. All in all, the results are very similar to those in the previous model with the continuous variable for education. It should be noted that in the model with interactions between weighted effect coded factor variables, the intercept again shows the same value, representing the average wage in the sample. Just like with the previous examples, the omitted estimates and standard errors (for instance the income effect of Hispanics without a degree) can be obtained by changing the omitted categories in the weighted effect coded factor variables, and by re-calculating the interaction variable(s).

Conclusion

This article discussed benefits and applications of weighted effect coding. It covered weighted effect coding as such, interactions between two weighted effect coded variables, and interactions with a weighted effect coded variable and an continuous variable. The **wec** package to apply these techniques in R was introduced. The examples shown in this article were based on OLS regression, but weighted effect coding (also) applies to all generalised linear models.

The benefits of using weighted effect coding are apparent when analysing observational data that, unlike experimental data, typically do not have an equal number of observations across groups or categories. When this is the case, the grand mean is not necessarily the appropriate point of reference. Consequently estimates of effects and standard errors based on weighted effect coding are not sensitive to how *other* observations are categorised.

With weighted effect coding, compared to treatment coding, no arbitrary reference category has to be selected. Instead, the sample mean serves as a point of reference. With treatment coding, selecting as a reference a category with a small number of observations and a deviant score can lead to significant results while this reference category has little contribution to the overall sample mean.

When weighted effect coded variables are used in interactions, the main effects remain unchanged after the introduction of the interaction terms. In previous, related, approaches this was not possible (West et al., 1996; Aguinis, 2004). This allows for the straightforward interpretation that the interaction terms represent how much the effect is weaker / stronger in each category. That is, when interacting with treatment coded categorical variables, the so-called ‘main’ effect refers to the reference category, whereas with weighted effect coding the unconditional main (/mean) effect is shown. As such, it can be used to test the assumption that estimated effects do not vary across groups.

It should be noted that the R-square of regression models does not depend on which type of dummy coding is selected. This means that the predicted values based on models using treatment coding, effect coding, or weighted effect coding, will be exactly the same. Yet, as each type of dummy coding selects a different point of reference, the interpretation of the estimates differs and a different statistical test is performed.

To conclude, the **wec** package contributes functionality to apply weighted effect coding to factor variables and interactions between (a.) a factor variable and a continuous variable and between (b.) two factor variables. These techniques are particularly relevant with unbalanced data, as is often the case when analysing observational data.

Bibliography

- H. Aguinis. *Regression Analysis for Categorical Moderators*. The Guilford Press, New York / London, 2004. [p481, 484]
- Census. *Public Use Microdata Sample (PUMS)*. prepared by the U.S. Census Bureau, United States of America, 2013. URL <https://doi.org/10.3886/icpsr33802.v1>. [p479]
- M. Grotenhuis, B. Pelzer, R. Eisinga, R. Nieuwenhuis, A. Schmidt-Catran, and R. Konig. A Novel Method for Modelling Interaction between Categorical Variables. *International Journal of Public Health*, 62(3):427–431, 2017a. URL <https://doi.org/10.1007/s00038-016-0901-1>. [p477, 481]
- M. Grotenhuis, B. Pelzer, R. Eisinga, R. Nieuwenhuis, A. Schmidt-Catran, and R. Konig. When Size Matters: Advantages of Weighted Effect Coding in Observational Studies. *International Journal of Public Health*, 62(1):263–167, 2017b. URL <https://doi.org/10.1007/s00038-016-0902-0>. [p477]
- J. Lammers. *Grootschalig Veldonderzoek. Inleiding in De Data-Analyse (An Introduction to the Analysis of Observational Data)*. Coutinho, Muiderberg, 1991. [p481]
- R. E. Sweeney and E. F. Ulveling. A Transformation for Simplifying the Interpretation of Coefficients of Binary Variables in Regression Analysis. *The American Statistician*, 1972. URL <https://doi.org/10.1080/00031305.1972.10478949>. [p477]
- S. G. West, L. S. Aiken, and J. L. Krull. Experimental Personality Designs: Analyzing Categorical by Continuous Variable Interactions. *Journal of personality*, 64(1):1–48, 1996. URL <https://doi.org/10.1111/j.1467-6494.1996.tb00813.x>. [p481, 484]

Rense Nieuwenhuis

Swedish Institute for Social Research (SOFI), Stockholm University

Affiliate of The Linnaeus Center on Social Policy and Family Dynamics in Europe (SPaDE), Stockholm University

106 91 Stockholm

Sweden

rense.nieuwenhuis@sofi.su.se

Manfred te Grotenhuis

Radboud University

Box 9104, 6500 HE, Nijmegen

The Netherlands

m.tegrotenhuis@maw.ru.nl

Ben Pelzer

Radboud University

Box 9104, 6500 HE, Nijmegen

The Netherlands

b.pelzer@maw.ru.nl

Hosting Data Packages via drat: A Case Study with Hurricane Exposure Data

by G. Brooke Anderson and Dirk Eddelbuettel

Abstract Data-only packages offer a way to provide extended functionality for other R users. However, such packages can be large enough to exceed the package size limit (5 megabytes) for the Comprehensive R Archive Network (CRAN). As an alternative, large data packages can be posted to additional repositories beyond CRAN itself in a way that allows smaller code packages on CRAN to access and use the data. The `drat` package facilitates creation and use of such alternative repositories and makes it particularly simple to host them via GitHub. CRAN packages can draw on packages posted to `drat` repositories through the use of the ‘Additional_repositories’ field in the DESCRIPTION file. This paper describes how R users can create a suite of coordinated packages, in which larger data packages are hosted in an alternative repository created with `drat`, while a smaller code package that interacts with this data is created that can be submitted to CRAN.

Motivation

“Big data”, apart from being a buzzword, also accurately describes the current scale of many scientific data sets. While the R language and environment (R Core Team, 2017a) enables the creation, use, and sharing of data packages to support methodology or application packages, the size of these data packages can be very large. The Bioconductor project has addressed the potentially large size requirements of data packages through the use of Git Large File Storage, with the package contributor covering costs for extremely large data packages (over 1 gigabyte) (Bioconductor Core Team, 2017). The Bioconductor repository, though, is restricted to topic-specific packages related to bioinformatics. The Comprehensive R Archive Network (CRAN), which archives R packages on any topic, has a recommended size limit (reasonable for a widely-mirrored repository) of 5 megabytes (MB) for package data and documentation (R Core Team, 2017b). A twofold need therefore arises for package maintainers seeking to share large R data packages that are outside the scope of the Bioconductor project. First, there is a need to create and share such a data package for integration into and extensions of a given methodology or application package, and, second, there is a need to integrate use of such a package in a way that makes it seamlessly integrated with smaller CRAN packages that use the data package. Here, we outline one possible approach to satisfy these needs by creating a suite of coordinated packages, in which larger data packages are hosted outside of CRAN but can still be accessed by smaller code packages that are submitted to CRAN.

The problem of creating CRAN packages that interface with large datasets is not new, and various approaches have been taken in the past to allow for such an interface. For example, the `NMMAPlite` package (currently available only from the CRAN archive) was built to allow users to interact with daily data on weather, air pollution, and mortality outcomes for over 100 US communities over 14 years (Peng and Dominici, 2008). To enable interaction with this large dataset through a CRAN package, the package maintainer posted the data on a server and included functions in the `NMMAPlite` package to create an empty database on the user’s computer that would be filled with community-specific datasets as the user queried different communities. This data interaction was enabled with the `stashR` package by the same maintainer (Eckel and Peng, 2012).

More recent packages similarly allow interaction between a web-hosted database and R, in some cases for a database maintained by an outside entity. For example, the `rnoaa` package allows access to weather data posted by the National Oceanic and Atmospheric Administration (NOAA) (Chamberlain et al., 2016), while the `tigris` package allows access to spatial data from the United States Census Bureau (Walker and Rudis, 2016). Both packages are posted on CRAN and allow R users to work with a large collection of available online data by creating and sending HTTP requests from R using the conventions defined in the respective online databases’ application program interfaces (APIs). This approach is a good one for data that is already available through an online database, especially if the database is outside the control of the R package maintainer or if the potential set of data is extremely large and it is unlikely that an R user would want to download all of it, since an API allows an R user to selectively download parts of the data. However, if the data is not already available through an online database, this approach would require the R package maintainer to create and maintain an online database, including covering the costs and managing the security of that database.

Another approach is to create a suite of packages, in which smaller (‘code’) packages are submitted to CRAN while larger (‘data’) packages are posted elsewhere. With this approach, the R user downloads all the data in the data package when he or she installs the data package, effectively

caching this data locally so that data can be used in different R sessions without reloading or when the computer is offline. This can be a good approach in cases where the data is not otherwise available through an online database and when R users are likely to want to download the full set of data. In this case, this second approach offers several advantages, including: (1) the data can be documented through package helpfiles that are easily accessible from the R console; (2) if a user would like to delete all downloaded data from their computer, he or she can easily do so by removing the entire data package with `remove.packages` (as compared to other caching solutions, in which case the user might need to do some work to determine where a package cached data on his or her computer); (3) versioning can be used for the data package, which can improve reproducibility of research using the data (Gentleman et al., 2004); and (4) the data package can include the R scripts used to clean the original data (for example, in a 'data-raw' directory, with the directory's name excluded from the R package build through a listing in the '.Rbuildignore' file), which will make the package more reproducible for the package author and, if the full package directory is posted publicly (e.g., through a public GitHub repository), for other users.

The **UScensus2000** suite of packages (currently available from the CRAN archive) used this approach to allow access to U.S. Census Bureau data from the 2000 Decennial Census (Almquist, 2010). This suite of packages included data at a more aggregated spatial level (e.g., state- and county-level data) through data packages submitted to CRAN, but included the largest dataset (block-level data) in an R package that was instead posted to the research lab's website (Almquist, 2010). A convenience function was included in one of the CRAN packages in the suite to facilitate installing this data package from the lab's website (Almquist, 2010).

This approach can be facilitated by posting the data package through an online package repository rather than a non-repository website. While support for repositories outside of CRAN, Bioconductor, and OmegaHat has existed within R for years, few users appear to have deployed this mechanism to host additional repositories. The **drat** package facilitates the creation and use of a user-controlled package repository. Once a package maintainer has created a repository, he or she can use it to host larger packages like a data package (and of course also any number of code packages). Use of a **drat** repository allows R users to install and update the data package using traditional R functions for managing packages (e.g., `install.packages`, `update.packages`) after the user has added the **drat** repository through a call to `addRepo` (Eddelbuettel et al., 2016). A **drat** repository can be published online through GitHub Pages, and GitHub repositories have a recommended maximum size of 1 GB, much larger than the size limit for a CRAN package, with a cap of 100 MB on any single file (<https://help.github.com/articles/what-is-my-disk-quota/>). Even for data packages below the CRAN size limit, this approach to hosting data packages can help remove some of the burden of hosting and archiving large data packages from CRAN. Although the **drat** package is relatively new, some package maintainers are already taking this approach—for example, the **grattan** package facilitates research in R on Australian tax policy, with relevant data available through the large **taxstats** data package, posted in a **drat** repository.

When taking the approach of creating a suite of packages in which the smaller code package or packages are submitted to CRAN while larger data packages are hosted in **drat** repositories, it is necessary to add some infrastructure to the smaller code packages. CRAN policies do not allow the submission of a package with mandatory dependencies on packages hosted outside of a mainstream repository, which means that the smaller package could not be submitted to CRAN if the data package is included in the smaller package through 'Imports:' or 'Depends:'. The R package ecosystem offers a solution: a package maintainer can create a weaker relationship between code and data packages via a 'Suggests:', which makes the data package optional, rather than mandatory, as either 'Imports:' or 'Depends:' would. Being optional, one then has to condition any code in the code package that accesses data in the data package on whether that package is present on the user's system. This approach offers the possibility of posting a data package that is too large for CRAN on a non-mainstream repository, like a **drat** repository, while submitting a package that interacts with the data to CRAN.

This paper outlines in sufficient detail the steps one has to take to use **drat** to host a large data package for R and how to properly integrate conditional access to such an optional data package into a smaller code package. The packaging standard aimed for is the CRAN Repository Policy and the passing of 'R CMD check --as-cran'. Broadly, these steps are:

1. Create a **drat** repository;
2. Create the data package(s), build it, and post it in the **drat** repository; and
3. Create / alter the code package(s) to use the data package(s) in a way that complies with CRAN checks.

As a case study, we discuss and illustrate the interaction between the packages **hurricaneexposure** (Anderson et al., 2017b) and **hurricaneexposedata** (Anderson et al., 2017a). The latter package contains data for eastern U.S. counties on rain, wind, and other hurricane exposures, covering all historical Atlantic-basin tropical storms over a few decades. The total size of the pack-

age's source code is approximately 25 MB, easily exceeding CRAN's package size limit. This package includes only data, but the companion package, **hurricaneexposure**, provides functions to map and interact with this data. The **hurricaneexposure** package is of standard size and available from CRAN since the initial version 0.0.1, but to fully utilize all its capabilities requires access to the data in package **hurricaneexposedata**. Here, we highlight specific elements of code in these packages that allow coordination between the two. The full code for both packages is available through GitHub repositories (<https://github.com/geanders/hurricaneexposure> and <https://github.com/geanders/hurricaneexposedata>); this article references code in version 0.0.2 of both packages.

Posting a data package to a drat repository

Creating a drat repository

A package maintainer must first create a **drat** repository if he or she wishes to host packages through one. Essentially, this repository is a way to store R packages such that it is easy for R users to download and update the packages; the repository can be shared, among other ways, through a GitHub-hosted website. Because a **drat** repository is controlled by the package maintainer, it allows increased flexibility to package maintainers compared to repositories like CRAN. A single **drat** repository can host multiple packages, so a maintainer likely only needs a single **drat** repository, regardless of how many packages he or she wishes to host on it.

A **drat** repository is essentially a network-accessible directory structure. The **drat** repository's directory must include index files (e.g., 'PACKAGES' and 'PACKAGES.gz'; Figure 1), which have metadata describing the packages available in the repository. The directory structure should also store files with the source code of one or more packages (e.g., 'hurricaneexposedata_0.0.2.tar.gz') and can also include operating system-specific package code (e.g., 'hurricaneexposedata_0.0.2.zip'). Multiple versions of a package can be included (e.g., 'hurricaneexposedata_0.0.1.tar.gz' and 'hurricaneexposedata_0.0.2.tar.gz' in Figure 1), allowing for archiving of old packages.

A user can create a **drat** directory with the required structure either by hand, via functions in the **drat** package, or by copying an existing **drat** repository (e.g., forking one on GitHub, like the original **drat** repository available at <https://github.com/eddelbuettel/drat>). While this directory can have any name, we suggest the user name the directory 'drat', as this allows the easy use of default variable names (which can of course be overridden as needed) for functions in the **drat** package, as shown in later code examples.

Second, for other users to be able to install packages from a **drat** repository, the repository must be available online. This can be easily achieved via the https protocol using GitHub's GitHub Pages, which allows GitHub users to create project webpages by posting content to an 'gh-pages' branch of the project's repository. (While there are now ways to publish content from a directory 'docs/' in the 'master' branch, functions in the **drat** package currently only support use of the older 'gh-pages' publishing option). Once this 'gh-pages' branch is pushed, the content in that branch will be available as part of the GitHub user's GitHub Pages website. For example, if the project is in the GitHub repository <https://github.com/username/projectname>, the content from the 'gh-pages' branch of that repository will be published at <https://username.github.io/projectname>.

Once this **drat** repository is created and published online through GitHub Pages, any source code or binaries for R packages within the repository can be installed and updated by R users through functions in the **drat** package. An R user can install a package from a **drat** repository by first adding the **drat** repository using the `addRepo` function from **drat**, with the appropriate GitHub username, and then using `install.packages`, as one would for a package on CRAN. The **drat** package documentation, including several vignettes, as well as supplementary webpages, have more detail on this process; see [Eddelbuettel et al. \(2016\)](#).

Creating and building a data package

The next step is to create an R data package that contains the large dataset; this data package will be posted in the **drat** repository to be accessible to other R users. In the case study example, this package is called **hurricaneexposedata** and includes data on rain, wind, flood, and tornado exposures for all eastern US counties for Atlantic-basin tropical storms between 1988 and 2015. For full details on creating R packages, including data-only packages, see the canonical reference by [R Core Team \(2017b\)](#); another popular reference is [Wickham \(2015\)](#).

If a package is hosted in a **drat** repository rather than posted to CRAN, it does not have to pass all tests executed by 'R CMD check'. However, it is good practice to resolve as many ERRORS, WARNINGS,

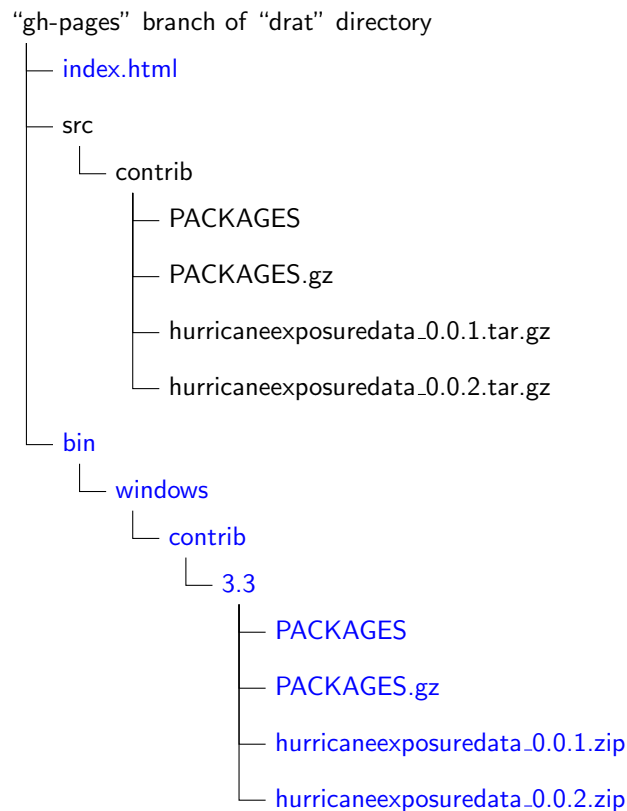


Figure 1: Example of the structure of the directory for a **drat** repository for a repository with two versions (0.0.1 and 0.0.2) of **hurricaneexposedata**. Directories and files shown in black are required while those in blue are optional. This example **drat** depository has source code available for the **hurricaneexposedata**, as well as the binaries for Windows (the binaries for Mac OS X could also be included but are not shown in this example), through the optional "bin" subdirectory. The top-level "index.html" file can be used to customize the appearance of the webpage a user would see at <https://username.github.io/drat>. Functions from the **drat** package automate the insertion of compressed package files ('.tar.gz' for source code files, '.tgz' for Mac OS X binaries, and '.zip' for Windows binaries) within this directory structure. The 'PACKAGES' and 'PACKAGES.gz' files serve as index files, with metadata about packages available in the repository, and are created by `drat::insertPackage` via a call to the R function `tools::write_PACKAGES`.

and NOTES from CRAN checks as possible for any R package that will be shared with other users, regardless of how it is shared. Several possibilities exist to build and check a package so these ERRORS, WARNINGS, and NOTES can be identified and resolved. The standard approach is to execute 'R CMD build' from one directory above the source directory (as discussed below), followed by 'R CMD check' with the resulting tar archive (e.g., `hurricaneexposedata.tar.gz`) as first argument. The optional switch '--as-cran' is recommended in order to run a wider variety of tests. Other alternatives for checking the package are to use the check function from the `devtools` package (Wickham et al., 2016), the `rcmdcheck` function of the eponymous `rcmdcheck` package (Csárdi, 2016), the 'Check' button in the Build pane of the RStudio GUI, or the RStudio keyboard shortcut Ctrl-Shift-E. For a large data package, it is desirable to resolve all issues except the NOTE on the package size being large.

Once the code in the data package is finalized, the package can be posted in a **drat** repository to be shared with others. Packages are inserted into a **drat** repository as source code tarballs (e.g., '.tar.gz' files); if desired, package binaries for specific operating systems can also be inserted (e.g., '.zip' or '.tgz' files), but this is not required for the application described here. To build a package into a '.tar.gz' file, there are again several possible approaches. The most convenient one may be to build the package in a temporary directory created with the `tmpdir` function, as this directory will be cleaned up when the current R session is closed. If the current working directory is the package directory, the package can be built to a temporary directory with the `build` function from the `devtools` package:

```
tmp <- tmpdir()
devtools::build(path = tmp)
```

While this function call assumes that the user is currently using the directory of the data package as the working directory, the `pkg` option of the `build` function can be used to run this call successfully from a different directory. If the build is successful, a '.tar.gz' file will be created containing the package's source code in the directory specified by the `path` option; this can be checked from R with the call `list.files(tmp)`.

Adding the data package to your drat repository

Once the source code of the package has been built, the data package can be inserted into the **drat** repository using the `insertPackage` function from the **drat** package. This function identifies the package file type (e.g., '.tar.gz', '.zip', or '.tgz'), adds the package to the appropriate spot in the **drat** directory structure (Figure 1), and adds metadata on the package to the 'PACKAGES' and 'PACKAGES.GZ' index files (created by the R function `tools::write_PACKAGES`) in the appropriate subdirectory. Once this updated version of the **drat** repository is pushed to GitHub, the package will be available for other users to install from the repository.

For example, the following code can be used to add version 0.0.2 of **hurricaneexposedata** to the **drat** repository. This code assumes that the '.tar.gz' file for the package was built into a temporary directory with a path given by the R object `tmp`, as would be the case if the user built the package tarball using the code suggested in the previous subsection, and that the user is in the same R session as when the package was built (as any temporary directories created with `tmpdir` are deleted when an R session is closed). Further, this code assumes that the user has the `git2r` package installed and has stored their **drat** directory within the parent directory `~/git`; if this is not the case, the correct path to the **drat** directory should be specified in the `repor_dir` argument of `insertPackage`.

```
pkg_path <- file.path(tmp, "hurricaneexposedata_0.0.2.tar.gz", sep = "/")
drat::insertPackage(pkg_path, commit = TRUE)
```

As mentioned before, only material in the 'gh-pages' branch of the GitHub repository is published through GitHub Pages, so it is important that the package be inserted in that branch of the user's **drat** repository. The `insertPackage` function checks out that branch of the repository and so ensures that the package file is inserted in the correct branch. However, it is important that the user be sure to push that specific branch to GitHub to update the online repository. If unsure, the `commit` option can be left at its default value of `FALSE`, permitting an inspection of the repository followed by a possible manual commit.

For users who prefer working from the command line, an alternative pipeline for building the data package and inserting it into the **drat** repository is to run, from the command line:

```
R CMD build sourcedir/
dratInsert.r pkg_1.2.3.tar.gz
```

Note that this pipeline requires having the `littler` (Eddelbuettel and Horner, 2016) package installed, as well as the 'dratInsert.r' helper script for that package.

If desired, operating system-specific binaries of the data package can be built with tools like `win-builder` (<http://win-builder.r-project.org/>) and `rhub` (<https://builder.r-hub.io>) and then inserted into the `drat` repository. However, this step is not necessary, as ‘R CMD check --as-cran’ will be satisfied for the code package as long as a source package is available for any suggested packages stored in repositories listed in ‘Additional_repositories’ (R Core Team, 2017b).

Setting up a smaller code package to use the data package

So far, the process described is the same one would use to create and add any R package to a `drat` repository. However, if a package maintainer would like to coordinate a code package that will be submitted to CRAN with a data package posted in a `drat` repository, it is necessary to add some infrastructure to the code package (in our example, `hurricaneexposure`, which has functions for exploring and mapping the data in `hurricaneexposedata`). These additions ensure that the code package will pass CRAN checks and also appropriately load and access the data in the data package posted in the `drat` repository.

Add infrastructure to the DESCRIPTION file

First, two additions are needed and a third is suggested in the DESCRIPTION file (Figure 2) of the code package that will be submitted to CRAN:

1. We suggest the ‘Description’ field of the code package’s ‘DESCRIPTION’ file be modified to let users know how to install the data package and how large it is (this tip is inspired by the `grattan` package; Parsonage et al. (2017)). Figure 2 (#1) shows an example of this added information for the `hurricaneexposure` ‘DESCRIPTION’ file. For a CRAN package, this ‘Description’ field will be posted on the package’s CRAN webpage, so this field offers an opportunity to inform users about the data package before they install the CRAN package. This addition is not required, but is particularly helpful in cases where the data package is very large, in which case it would take up a lot of room on a user’s computer and take a long time to install and load.
2. The ‘Suggests’ field for the code package must specify the suggested dependency on the data package (Figure 2, #2). Because the data package is in a non-mainstream repository, this dependency must be specified in the ‘Suggests’ field rather than the ‘Depends’ or ‘Imports’ field if the code package is to be submitted to CRAN. The ‘Suggests’ field allows version requirements, so if the code package requires either a minimum version or an exact version of the data package in the `drat` repository, this requirement can be included in this field.
3. The ‘Additional_repositories’ field of the code package must give the address of the `drat` repository that stores the data package (Figure 2, #3). This field is necessary if a package depends on a package in a non-mainstream repository. Repositories listed here are checked by CRAN to confirm their availability (R Core Team, 2017b), but packages from these repositories are not installed prior to CRAN checks. This repository address should be listed using `https:` rather than `http:`.

Customize behavior when the package is loaded

When a package is installed, any packages listed as ‘Imports’ or ‘Depends’ in the package ‘DESCRIPTION’ file are guaranteed to be previously installed (R Core Team, 2017b); the same is not true for packages in ‘Suggests’. It is therefore important that the maintainer of any package that suggests a data package from a `drat` repository take steps to ensure that the package does not fail if it is installed without the data package being previously installed. Such steps includes adding code that will be run when the package is loaded (described in this subsection), as well as ensuring that code in all functions, examples, vignettes, and tests be conditional on whether the data package installed if the code requires data from the data package (described in later subsections).

First, the code package should have code to check whether the data package is installed when the code package is loaded. This can be achieved through load hooks (`.onLoad` and `.onAttach`), saved to a file named, for example, ‘zzz.R’ in the ‘R’ directory of the code package (the name ‘zzz.R’ dates back to a time when R required this; now any file name can be chosen). For a concrete example, such a ‘zzz.R’ file in the code package might look like (numbers in comments of this code are used within specific comments later in this section): 2

```

Package: hurricaneexposure
Type: Package
Title: Explore and Map County-Level Hurricane Exposure in the United States
Version: 0.0.2
Date: 2017-01-31
Authors@R: c(person("Brooke", "Anderson",
  email = "brooke.anderson@colostate.edu", role = c("aut", "cre")),
  person("Meilin", "Yan",
  email = "meilin.yan@colostate.edu", role = "aut"),
  person("Joshua", "Ferreri",
  email = "joshua.m.ferreri@gmail.com", role = "aut"),
  person("William", "Crosson",
  email = "bill.crosson@nasa.gov", role = "ctb"),
  person("Mohammad", "Al-Hamdan",
  email = "mohammad.alhamdan@nasa.gov", role = "ctb"),
  person("Andrea", "Schumacher",
  email = "andrea.schumacher@colostate.edu", role = "ctb"),
  person("Dirk", "Eddelbuettel",
  email = "edd@debian.org", role = "ctb")
)

```

```

Description: Allows users to create time series of tropical storm
  exposure histories for chosen counties for a number of hazard metrics (wind,
  rain, distance from the storm, etc.). This package interacts with data available
  through the 'hurricaneexposedata' package, which is available in a 'drat'
  repository. To access this data package, run
  'install.packages("hurricaneexposedata",
  repos = "https://geanders.github.io/drat/", type = "source")'. The size of the
  'hurricaneexposedata' package is approximately 25 MB. This work was supported in
  part by grants from the National Institute of Environmental Health Sciences
  (R00ES022631), the National Science Foundation (1331399), and a NASA Applied
  Sciences Program/Public Health Program Grant (NNX09AV81G).

```

```

URL: https://github.com/geanders/hurricaneexposure
BugReports: https://github.com/geanders/hurricaneexposure/issues
License: GPL (>= 2)
LazyData: TRUE
Imports:

```

```

  data.table (>= 1.9.6),
  dplyr (>= 0.4.3),
  ggmap (>= 2.6.1),
  ggplot2 (>= 2.1.0),
  lazyeval (>= 0.1.10),
  lubridate (>= 1.5.6),
  maps (>= 3.1.1),
  purrr (>= 0.2.2),
  RColorBrewer (>= 1.1.2),
  splines,
  stats,
  stringr (>= 1.1.0),
  tidyr (>= 0.6.0)
RoxygenNote: 5.0.1
Depends:

```

```

  R (>= 2.10)

```

```

Suggests:
  hurricaneexposedata (>= 0.0.2),

```

```

  knitr,
  pander,
  rmarkdown,
  weathermetrics
VignetteBuilder: knitr

```

```

Additional_repositories: https://geanders.github.io/drat

```

Figure 2: Example of the elements that should be added to the DESCRIPTION file of the code package planned to be submitted to CRAN to coordinate it with a data package posted to a **drat** repository, showing the DESCRIPTION file for **hurricaneexposure**. Elements are: (1) added details on installing the data package in the 'Description' field (suggested but not required); (2) suggested dependency on the data package in the 'Suggests' field; and (3) reference to the drat repository in the 'Additional_repositories' field.

```

.pkgenv <- new.env(parent=emptyenv()) #1

.onLoad <- function(libname, pkgname) { #2
  has_data <- requireNamespace("hurricaneexposedata", quietly = TRUE) #3
  .pkgenv[["has_data"]] <- has_data #4
}

.onAttach <- function(libname, pkgname) { #5
  if (!.pkgenv$has_data) { #6
    msg <- paste("To use this package, you must install the",
                 "hurricaneexposedata package. To install that ",
                 "package, run `install.packages('hurricaneexposedata',",
                 "repos='https://geanders.github.io/drat/', type='source')`.",
                 "See the `hurricaneexposure` vignette for more details.")
    msg <- paste(strwrap(msg), collapse="\n")
    packageStartupMessage(msg)
  }
}

hasData <- function(has_data = .pkgenv$has_data) { #7
  if (!has_data) {
    msg <- paste("To use this function, you must have the",
                 "`hurricaneexposedata` package installed. See the",
                 "`hurricaneexposure` package vignette for more details.")
    msg <- paste(strwrap(msg), collapse="\n")
    stop(msg)
  }
}

```

First, this ‘zzz.R’ file creates an environment called `.pkgenv` (#1 in example code). This environment will be used to pass a Boolean variable indicating whether the data package is available (#4) to other code in the package.

Next, this ‘zzz.R’ file defines two functions called `.onLoad` and `.onAttach`. Both functions have arguments `libname` and `pkgname` (#2, #5). The `.onLoad()` function, triggered when code from the package is first accessed, should use `requireNamespace` to test if the user has the data package available (#3). This value is stored in the package environment (#4). When the package is loaded and attached to the search path, the `.onAttach` function is called (#5). This function references the stored Boolean value and uses this to print a start-up message (using the `packageStartupMessage` function, whose output can be suppressed via `suppressPackageStartupMessages`) for users who lack the data package (#6).

Finally, any functions in the package that use data from the `drat` data package should check that the data package is available before running the rest of the code in the function. If the data package is not available, the function should stop with a useful error message. One way to achieve this is to define a function in an R script file in the package, like the `hasData` function defined in the example code above, that checks for availability of the data package and errors with a helpful message if that package is not installed (#7). This function should then be added within all of the package’s functions that require data from the data package.

Condition code in the code package on availability of the data package

Next, it is important to ensure that code in the vignettes, examples, and tests of the package will run without error on CRAN, even without the data package installed. When a package is submitted to CRAN, the user first creates a tarball of the package source on his or her own computer. In this local build, the vignette is rendered and the resulting PDF or HTML file is stored within the ‘`inst/doc`’ directory of the source code (R Core Team, 2017b). This is the version of the rendered vignette that is available to users when they install the package, and so the vignette is rendered using the packages and other resources available on the package maintainer’s computer during the package build. However, CRAN runs initial checks on a package submission, and continues to run regular checks on posted packages, that include testing any executable code within the package vignette or any examples or tests in the source code that are not explicitly marked to not run on CRAN (e.g., with `\donttest{}` within example code). CRAN does not install suggested packages from non-mainstream repositories before doing these checks. Therefore, code that requires data from a data package in a `drat` repository

would cause errors in these tests unless it is conditioned to only run when the data package is available, as is recommended for any example or test code that uses suggested packages (R Core Team, 2017b).

Therefore, if the code package contains a vignette, the vignette must be coded so that its code will run without an error on systems that do not have the data package installed. This can be done by adding a code chunk to the start of the vignette. This code chunk will check if the data package is installed on the system. If it is, the vignette will be rendered as usual, and so it will be rendered correctly on the package maintainer's computer when the package is built for CRAN submission. However, if the optional data package is not installed, a message about installing the data package from the **drat** repository will be printed in the vignette, and all the following code chunks will be set to not be evaluated using the `opts_chunk` function from **knitr** (Xie, 2016).

The following code is an example of the code chunk that was added to the beginning of the vignette in the **hurricaneexposure** package, for which all code examples require the **hurricaneexposedata** package:

```
```${r echo = FALSE, message = FALSE}
hasData <- requireNamespace("hurricaneexposedata", quietly = TRUE) #1
if (!hasData) { #2
 knitr::opts_chunk$set(eval = FALSE) #3
 msg <- paste("Note: Examples in this vignette require that the",
 "`hurricaneexposedata` package be installed. The system",
 "currently running this vignette does not have that package",
 "installed, so code examples will not be evaluated.")
 msg <- paste(strwrap(msg), collapse="\n")
 message(msg) #4
}
```
```

In this code, the function `requireNamespace` is used to check if **hurricaneexposedata** is installed on the system (#1). It is necessary to run this function in the vignette with the `quietly = TRUE` option; otherwise, this call will cause an error if **hurricaneexposedata** is unavailable. If **hurricaneexposedata** is not available, the result of this call is `FALSE`, in which case (#2) the chunk option `eval` is set to `FALSE` for all following chunks in the vignette (#3) and a message is printed in the vignette explaining why code chunks are not evaluated (#4).

Similarly, the code for examples in help files of the CRAN package should be adjusted so they only run if the data package is available. It may also be helpful to users to include a commented message on why the example is wrapped in a conditional statement. For example, for a function that requires the data package the `\examples{}` field (or `@examples` tag if **roxygen2** is used) might look like:

```
\examples{
  # Ensure that data package is available before running the example.
  # If it is not, see the `hurricaneexposure` package vignette for details
  # on installing the required data package.
  if (requireNamespace("hurricaneexposedata", quietly = TRUE)) {
    map_counties("Beryl-1988", metric = "wind")
  }
}
```

As alternatives, the code in the example could also either check the `.pkgenv[["has_data"]]` object created by code in the `'zzz.R'` file in the conditional statement, or the package maintainer could create a helper function to in the `'zzz.R'` file to use for conditional running of examples in the example code. However, the use of the `requireNamespace` call, as shown in the above code example, may be the most transparent for package users to understand when working through package examples. Code in package tests can similarly be conditioned to run if and only if the data package is available.

Maintaining a suite of drat and CRAN R packages

Once a suite of coordinated CRAN and **drat** packages have been created, there are a few considerations a package maintainer should keep in mind for maintaining the packages.

First, unlike a single package that combines data and code, a suite of packages will require thoughtful coordination between versions of packages in the suite, especially as the packages are updated. The `'Suggests'` field allows the package maintainer to specify an exact version of the data package required by a code package (e.g., using `'(= 0.0.1)'` with the data package name in the

Suggests field) or a minimum version of the data package (e.g., ' $>= 0.0.1$ '). If the data package is expected to only change infrequently, the maintainer may want to use an exact version dependency in 'Suggests' and plan to submit an updated version of the code package to CRAN any time the data package is updated. However, CRAN policy recommends that package versions not be submitted more than once every one to two months. Therefore, if the data package will be updated more frequently, it may make more sense to use a minimum version dependency. However, the maintainer should be aware that, in this case, packages users may find that any changes in the structure of the data in the data package that breaks functions in older versions of the code package may cause errors, without notifying the user that the error can be resolved by updating the code package.

Second, while the maximum size of a GitHub repository is much larger than the recommended maximum size of an R package, there are still limits on GitHub repository size. If a package maintainer uses a **drat** repository to store multiple large data packages, with multiple versions of each, there may be some cases where the repository approaches or exceeds the maximum allowable GitHub repository size. In this case, the package maintainer may want to consider, when inserting new versions of the data package into the **drat** repository, changing the `action` option in the `insertPackage` function to free repository space by removing older versions of the package or packages. This, however, reduces reproducibility of research using the data package, as older versions would no longer be available through an archive.

Further, while the CRAN maintainers regularly run code from examples and vignettes within packages posted to CRAN, they do not install any suggested packages from non-mainstream repositories before doing this. Since much of the code in the examples and vignette are not run if the data package is not available under the approach we suggest, regular CRAN checks will provide less code coverage than is typical for a CRAN package. Package maintainers should keep this in mind, and they may want to consider alternatives for ensuring regular testing of all examples in the code package. One option may be through use of a regularly scheduled cron job through Travis CI to check the latest stable version of the code package.

It is important to note that, under the suggested approach, proper versioning of any data packages hosted in a **drat** repository is entirely the responsibility of the owner of that repository. By contrast, R users who install packages from CRAN can be confident that a version number of a package is tied to a unique version of the source code. While versioning of R data packages can improve research reproducibility (Gentleman et al., 2004), if the owner of the **drat** repository is not vigilant about changing the version number for every change in the source code of packages posted in the repository, the advantages of packaging the data in terms of facilitating reproducible research are lost. Similarly, the repository owner is solely responsible for archiving older versions of the package, unlike a CRAN package, for which archiving is typically ensured by CRAN. In particular, for very large packages that are updated often, the size limitations of a GitHub repository may force a repository owner to remove older versions from the archive.

Conclusion

R packages offer the chance to distribute large datasets while also providing functions for exploring and working with that data. However, data packages often exceed the suggested size of CRAN packages, which is a challenge for package maintainers who would like to share their code through this central and popular repository. Here, we suggest an approach in which the maintainer creates a smaller code package with the code to interact with the data, which can be submitted to CRAN, and a separate data package, which can be hosted by the package maintainer through a personal **drat** repository. Although **drat** repositories are not mainstream, and so cannot be listed with an 'Imports' or 'Depends' dependency for a package submitted to CRAN, we suggest a way of including the data package as a suggested package and incorporating conditional code in the executable code within vignettes, examples, and tests, as well as conditioning functions in the code package to check for the availability of the data package. This approach may prove useful for a number of R package maintainers, especially with the growing trend to the sharing and use of open data in many of the fields in which R is popular.

Acknowledgements

We thank Martin Morgan and Roger Peng for comments and suggestion on an earlier draft of this manuscript. This work was supported in part by a grant from the National Institute of Environmental Health Sciences (R00ES022631).

Bibliography

- Z. W. Almquist. US Census spatial and demographic data in R: The UScensus2000 suite of packages. *Journal of Statistical Software*, 37(6):1–31, 2010. URL <https://doi.org/10.18637/jss.v037.i06>. [p487]
- B. Anderson, A. Schumacher, W. Crosson, M. Al-Hamdan, M. Yan, J. Ferreri, Z. Chen, S. Quiring, and S. Guikema. *hurricaneexposedata: Data Characterizing Exposure to Hurricanes in United States Counties*, 2017a. URL <https://github.com/geanders/hurricaneexposedata>. Github repository containing R package. [p487]
- B. Anderson, M. Yan, J. Ferreri, W. Crosson, M. Al-Hamdan, A. Schumacher, and D. Eddelbuettel. *hurricaneexposure: Explore and Map County-Level Hurricane Exposure in the United States*, 2017b. URL <https://CRAN.R-project.org/package=hurricaneexposure>. R package version 0.0.1. [p487]
- Bioconductor Core Team. Bioconductor– Package guidelines. <http://bioconductor.org/developers/package-guidelines/>, 2017. Accessed: 2017-01-31. [p486]
- S. Chamberlain, B. Anderson, M. Salmon, A. Erickson, N. Potter, J. Stachelek, A. Simmons, K. Ram, and H. Edmund. *rnoaa: 'NOAA' Weather Data from R*, 2016. URL <https://CRAN.R-project.org/package=rnoaa>. R package version 0.6.6. [p486]
- G. Csárdi. *rcmdcheck: Run 'R CMD Check' from 'R' and Capture Results*, 2016. URL <https://CRAN.R-project.org/package=rcmdcheck>. R package version 1.2.1. [p490]
- S. Eckel and R. D. Peng. *stashR: A Set of Tools for Administering SHared Repositories*, 2012. URL <https://CRAN.R-project.org/package=stashR>. R package version 0.3.5. [p486]
- D. Eddelbuettel and J. Horner. *littler: R at the Command-Line via 'r'*, 2016. URL <https://CRAN.R-project.org/package=littler>. R package version 0.3.1. [p490]
- D. Eddelbuettel, C. Boettiger, S. Gibb, C. Gillespie, J. Górecki, M. Jones, T. Leeper, S. Pav, and J. Schulz. *drat: Drat R Archive Template*, 2016. URL <https://CRAN.R-project.org/package=drat>. R package version 0.1.2. [p487, 488]
- R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, and others. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, 2004. URL <https://doi.org/10.1186/gb-2004-5-10-r80>. [p487, 495]
- H. Parsonage, T. Cameron, B. Coates, W. Young, and I. Cherastidtham. *grattan: Perform Common Quantitative Tasks for Australian Analysts and to Support Grattan Institute Analysis*, 2017. URL <https://CRAN.R-project.org/package=grattan>. R package version 1.3.0. [p491]
- R. D. Peng and F. Dominici. *Statistical Methods for Environmental Epidemiology with R: A Case Study in Air Pollution and Health*. Springer-Verlag, New York, NY, USA, 2008. [p486]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017a. URL <https://www.R-project.org/>. [p486]
- R Core Team. *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2017b. URL <http://CRAN.R-Project.org/doc/manuals/R-exts.html>. [p486, 488, 491, 493, 494]
- K. Walker and B. Rudis. *Tigris: Load Census TIGER/Line Shapefiles into R*, 2016. URL <https://CRAN.R-project.org/package=tigris>. R package version 0.3.3. [p486]
- H. Wickham. *R Packages*. O'Reilly Media, Inc., Sebastopol, California, 2015. [p488]
- H. Wickham, W. Chang, and RCoreTeam. *devtools: Tools to Make Developing R Packages Easier*, 2016. URL <https://CRAN.R-project.org/package=devtools>. R package version 1.12.0. [p490]
- Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2016. URL <https://CRAN.R-project.org/package=knitr>. R package version 1.15.1. [p494]

G. Brooke Anderson
Colorado State University
Lake Street

Fort Collins, CO
brooke.anderson@colostate.edu

Dirk Eddelbuettel
Debian and R Projects; Ketchum Trading
Chicago, IL, USA
edd@debian.org

R Foundation News

by *Torsten Hothorn*

Donations and members

Membership fees and donations received between 2017-03-15 and 2017-06-15.

Donations

Emilio Ciccone (Italy)
Guillaume Coqueret (France)
Yves Deville (France)
Ken Ikeda (Japan)
Kuniaki Kawahara (Japan)
Kem Phillips (United States)
Joshua Rosenstein (United States)
Stefan Wyder (Switzerland)

Supporting benefactors

Brigham Young University, Provo (United States)
Displayr, Glebe (Australia)

Supporting institutions

BC Cancer Agency, Vancouver (Canada)
EMBL, Heidelberg (Germany)
Institute of Mathematical Statistics, Beachwood (United States)

Supporting members

Abdulrahman Al-Qasem (Saudi Arabia)
Philippe Baril Lecavalier (Canada)
Marcel Baumgartner (Switzerland)
Morten Braüner (Denmark)
Andrew Brown (United States)
Ian Cook (United States)
Robin Crockett (United Kingdom)
Gábor Csárdi (United Kingdom)
Gergely Daróczi (Hungary)
James Davis (United States)
Ajit de Silva (United States)
Dubravko Dolic (Germany)
Sandrine Dudoit (United States)
Joran Elias (United States)
Daniel Emaasit (United States)
Soo-Heang Eo (South Korea)
Arturo Erdely (Mexico)
Hubert Eser (Austria)
John Fox (Canada)
Carl Ganz (United States)
Jan Marvin Garbuszus (Germany)

J. Antonio García (Mexico)
Matthieu Gousseff (France)
Arthur W. Green (United States)
Philippe Grosjean (Belgium)
Hlynur Hallgrímsson (Iceland)
Bela Hausmann (Austria)
Kieran Healy (United States)
Jim Hester (United States)
Hans Hlynsson (United Kingdom)
Rick Hubbard (United States)
Michael Johansson (Puerto Rico)
Stephen Kaluzny (United States)
Christian Keller (Switzerland)
Sebastian Koehler (Germany)
Katarzyna Kopczewska (Poland)
Alexander Kowarik (Austria)
Diego Kuonen (Switzerland)
Caleb Lareau (United States)
Andy Liaw (United States)
Ian Lyttle (United States)
Ben Marwick (United States)
Shigeru Mase (Japan)
Dieter Menne (Germany)
David Monterde (Spain)
Guido Möser (Germany)
Bob Muenchen (United States)
Hannes Mühleisen (Netherlands)
Joris Muller (France)
Michihiro Nakamura (Japan)
Anthony OFarrell (United States)
Ludvig Renbo Olsen (Denmark)
George Ostrouchov (United States)
Matthew Pancia (United States)
Marc Pelath (United States)
John Pellman (United States)
Lauf Peter (Germany)
Erik Petrovski (Denmark)
Thomas Petzoldt (Germany)
Jonas Ranstam (Sweden)
Paulo Justiniano Ribeiro Jr (Brazil)
Marty Rose (United States)
Peter Ruckdeschel (Germany)
Bob Rudis (United States)
Manel Salamero (Spain)
Kenneth Spriggs (United States)
Julian Stander (United Kingdom)
Berthold Stegemann (Germany)
Nicholas Tierney (Australia)
Caoimhin Ua Buachalla (Ireland)
Shinya Uryu (Japan)
Mauricio Vargas (United States)
Steven Vasquez-Grinnell (United States)
Boris Veytsman (United States)
Vincent Vinh-Hung (Martinique)
Daehler Werner (Switzerland)
Jordan White (United States)

Andy Wills (United Kingdom)
Nan Xiao (United States)
Hiroaki Yutani (Japan)
Tomas Zelinsky (Slovakia)

Torsten Hothorn
Universität Zürich, Switzerland
Torsten.Hothorn@R-project.org

Conference Report: European R Users Meeting 2016

by Maciej Beręsewicz, Adolfo Alvarez, Przemysław Biecek, Marcin K. Dyderski, Marcin Kosinski, Jakub Nowosad, Kamil Rotter, Alicja Szabelska-Beręsewicz, Marcin Szymkowiak, Łukasz Wawrowski, Joanna Zyprych-Walczak

Introduction

The European R Users Meeting (eRum) 2016 was an international conference aimed at integrating users of the R language. eRum 2016 was held between October 12 and 14, 2016, in Poznań, Poland at Poznań University of Economics and Business (<http://erum.ue.poznan.pl/>).

The main purpose of eRum was to integrate R users from Europe and provide a platform for sharing experiences between academics and practitioners. We wanted to give the participants the possibility to present various applications of R, get to know different R packages and get involved in a broader collaboration. In addition, we wanted to create an opportunity for R users who were not able to participate in UseR 2016 that was held in Stanford, CA, USA.

The conference was organized by the Students Scientific Association 'Estymator', the Department of Statistics of Poznań University of Economics and Business and the Department of Mathematical and Statistical Methods of Poznań University of Life Sciences. The supporting organizers were two local groups of R enthusiasts - PAZUR - Poznań R Users Group and SER - Warsaw R Users Group. eRum hosted 82 speakers from over 20 countries. Over 250 participants took part in 12 parallel sessions devoted to methodology, business, R packages, data workflow, bioR, lightning talks, and education learning. Beside the parallel sessions the conference featured 10 invited talks.

We are looking forward to the next European R Users Meeting in 2018 and we encourage anyone interested in organizing an eRum event in their country to get in touch with us.

The Organizing Committee

To ensure the Organizing Committee was as diverse as possible, its chair, Maciej Beręsewicz from Poznań University of Economics and Business and Statistical Office in Poznań had invited people from different fields of science and business:

- Adolfo Alvarez, Analyx,
- Przemysław Biecek, MIM University of Warsaw, MiNI Warsaw University of Technology,
- Marcin Dyderski, Institute of Dendrology of the Polish Academy of Sciences, Poznań University of Life Sciences,
- Marcin Kosiński, Warsaw R Enthusiasts,
- Jakub Nowosad, Adam Mickiewicz University in Poznań, University of Cincinnati,
- Kamil Rotter, Poznań University of Economics and Business,
- Alicja Szabelska-Beręsewicz, Poznań University of Life Sciences,
- Marcin Szymkowiak, Poznań University of Economics and Business,
- Łukasz Wawrowski, Poznań University of Economics and Business,
- Joanna Zyprych-Walczak, Poznań University of Life Sciences.

Pre-conference workshops

Ten workshops were held at Poznań University of Life Sciences and in the Statistical Office in Poznań during the first day of eRum 2016 (October 12th 2016). They were divided into two sessions.

Morning session:

- An introduction to R (in Polish) held by Adam Dąbrowski
- Predictive modeling with R held by Artur Suchwałko
- Data Visualization using R held by Matthias Templ
- Time series forecasting with R held by Adam Zagdański
- R for expression profiling by Next Generation Sequencing held by Paweł Łabaj

Afternoon session:

- Introduction to Bayesian Statistics with R and Stan held by Rasmus Bååth
- Small Area Estimation with R held by Virgilio Gómez-Rubio
- R for industry and business: Statistical tools for quality control and improvement held by Emilio L. Cano
- An introduction to changepoint models using R held by Rebecca Killick
- Visualising spatial data with R: from 'base' to 'shiny' held by Robin Lovelace

Altogether the workshops were attended by 150 people. The workshops dedicated to Bayesian Statistics with R and Stan and Predictive modeling with R turned out to be the most popular ones - each attracted as many as 40 attendees.

Invited speakers

There were five plenary sessions, each featuring two invited speakers. The speaker roster was balanced in terms of nationality (5 from Poland, 5 from abroad), gender (6 males, 4 females), institution (7 from academia, 3 from business) and topic (5 methodology, 5 tools or misc).

We were honored to host the following invited speakers: **Rasmus Bååth** (Lund University), **Przemysław Biecek** (University of Warsaw), **Romain Francois** (Consulting Datactive), **Marek Gagolewski** (Polish Academy of Sciences), **Jakub Glinka** (GfK Data Lab), **Ulrike Grömping** (Beuth University of Applied Sciences), **Katarzyna Kopczewska** (University of Warsaw), **Katarzyna Stapor** (Silesian University of Technology), **Matthias Templ** (Vienna University of Technology), **Heather Turner** (University of Warwick).

All invited talks were recorded and are available as a youtube playlist at <http://www.youtube.com/playlist?list=PLCsJUtCRSFbejqCqAURNV0FFpoDCMeu05>. The book of abstracts, presentations from all sessions, posters and other documents associated with eRum 2016 are available at <https://github.com/eRum2016/>.

Trivia

The conference speakers cited 179 R packages. The most popular included: **ggplot2**, **shiny**, **dplyr**, **caret**, **leaflet**, **sp**, **cluster**, **e1071**, **knitr**, **magrittr**, **randomForest**, **Rcpp**, **rgdal**, **rgeos**, **rmarkdown**, **rstan**, **SparkR**.

Thanks to our platinum sponsor McKinsey it was possible to organize a social event which included a conference dinner at Poznań's Municipal Stadium together with a tour of the Stadium.

Group photos taken on the first day of the conference can be found here <https://www.google.gl/X0QfZ5>.

After the conference some of the participants took part in a sightseeing tour of Poznań, which was organized on 15th of October.

Acknowledgements

We would like to thank the Faculty of Informatics and Electronic Economy Poznan University of Economics and Business and our sponsors (McKinsey, Microsoft, eoda, Analyx, tidk, DataCamp, WLOG solutions, Quantide and Rstudio) for their financial support, which was used to organize daily lunches, coffee breaks and a social program featuring a conference dinner at the Municipal Stadium in Poznań.

Special thanks to [Klaudia Korniluk](#) for her outstanding work on the logo of eRum 2016 and coming up with the idea of "R heros". We also thank R-Bloggers and SmartedPoland for their support in spreading the word about eRum 2016 in the Internet.

In addition we would like to thank Prof. Elżbieta Gołata, Prof. Grażyna Dehnel, Prof. Idzi Siatkowski, Prof. Anita Dobek and directors of the Statistical Office in Poznań (dr Jacek Kowalewski and dr Tomasz Klimanek) for their support in organizing the conference.

Maciej Beręsewicz
Poznań University of Economics and Business
Niepodległości 10 av., 61-875 Poznań
Poland
maciej.berensewicz@ue.poznan.pl

Adolfo Alvarez
Analyx
Święty Marcin 24, 60-101 Poznań
Poland
adalvarez@gmail.com

Przemysław Biecek
MIM University of Warsaw, MiNI Warsaw University of Technology
Banacha 2, 02-097 Warszawa
Poland
przemyslaw.biecek@gmail.com

Marcin K. Dyderski
Institute of Dendrology of the Polish Academy of Sciences
Parkowa 5, 62-035 Kórnik
Poland
Marcin.Dyderski@gmail.com

Marcin Kosiński
Warsaw R Enthusiasts
Koszykowa 75, Warszawa
Poland
m.p.kosinski@gmail.com

Jakub Nowosad
University of Cincinnati
219 Braunstein Hall, Cincinnati, OH 45221
USA

nowosad.jakub@gmail.com

Kamil Rotter
SKN Estymator
Niepodległości 10 av., 61-875 Poznań
Poland
limak665@gmail.com

Alicja Szabelska-Beręsewicz
Poznań University of Life Sciences
Wojska Polskiego 28, 60-637 Poznań
Poland
aszab@up.poznan.pl

Marcin Szymkowiak
Poznań University of Economics and Business
Niepodległości 10 av., 61-875 Poznań
Poland
marcin.szymkowiak@ue.poznan.pl

Łukasz Wawrowski
Poznań University of Economics and Business
Niepodległości 10 av., 61-875 Poznań
Poland
lukasz.wawrowski@ue.poznan.pl

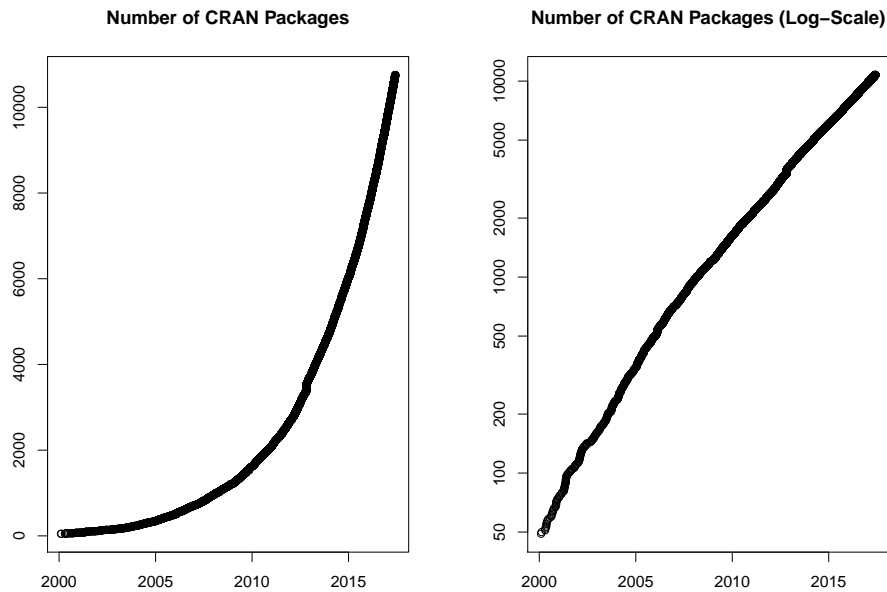
Joanna Zyprych-Walczak
Poznań University of Life Sciences
Wojska Polskiego 28, 60-637 Poznań
Poland
zjanna@up.poznan.pl

Changes on CRAN

2017-02-01 to 2017-05-31

by Kurt Hornik, Uwe Ligges and Achim Zeileis

In the past 4 months, 794 new packages were added to the CRAN package repository. 16 packages were unarchived, 98 archived and 1 removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2017-05-31, the number of active packages was around 10727.

Changes in the CRAN checks

In addition to the results for the regular check runs and the valgrind, ASAN and UBSAN tests of memory access errors provided by Brian Ripley, the package check pages now also show additional issues found by tests without long double (also provided by Brian Ripley) and checks of native code (C/C++) based on static code analysis, currently reporting potential errors in the use of PROTECT (provided by Tomáš Kalibera).

Changes in the CRAN submission pipeline

In the light of the many submissions of new and updated packages received every day, CRAN is in transition to a more and more automated submission system. Package maintainers may experience that their packages are auto-accepted in case a well established package without other packages depending on it passes the checks without problems. Some packages will undergo a manual inspection as before, but it may also happen a package is auto-rejected in case problems occur. In case you strongly believe the auto-rejection is a false positive, the procedure of contacting the CRAN team is explained in the rejection message.

Changes in the CRAN Repository Policy

The following items were added to the [Policy](#):

- CRAN versions of packages should work with the current CRAN and Bioconductor

releases of dependent packages and not anticipate nor recommend development versions of such packages on other repositories.

- Downloads of additional software or data as part of package installation or startup should only use secure download mechanisms (e.g., 'https' or 'ftps').

CRAN mirror security

Currently, there are 97 official CRAN mirrors, 51 of which provide both secure downloads via 'https' and use secure mirroring from the CRAN master (via rsync through ssh tunnels). Since the R 3.4.0 release, `chooseCRANmirror()` offers these mirrors in preference to the others which are not fully secured (yet).

CRAN tools

Since R 3.4.0, package **tools** 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. See `?tools::CRAN_package_db` for more information.

Hyperlinks in package DESCRIPTION files on CRAN

The CRAN package web page shows important information about the package and gives the package's Description. Package maintainers are now encouraged to insert web links and links to relevant publications they want to cite in order to explain the package's content by using the following formats (all enclosed in `<...>`) that will cause automatical insertions of the corresponding links:

Classical web hyperlinks should be given as the URL enclosed in `<...>`, e.g., write `<https://www.R-project.org>`.

Digital Object Identifier System (DOI) entries to link to a publication should be given in the form `<DOI:10.xxxx...>`.

arXiv.org e-prints without a DOI (yet) should be referred to using their arXiv identifier enclosed in `<...>`, e.g., `<arXiv:1501.00001>` or `<arXiv:0706.0001v2>`.

New CRAN task views

FunctionalData Topic: Functional Data Analysis. Maintainer: Fabian Scheipl. Packages: **FDboost***, **Funclustering**, **GPFDA**, **MFPCA**, **RFgroove**, **classiFunc**, **dbstats**, **fda***, **fda.usc***, **fdaPDE**, **fdakma**, **fdapace***, **fdasrvf***, **fdatest**, **fdcov**, **fds**, **flars**, **fpca**, **freqdom**, **fts***, **funData**, **funFEM**, **funHDDC**, **funcy***, **geofd**, **growfunctions**, **pcdpca**, **rainbow**, **refund***, **refund.shiny**, **refund.wave**, **roahd**, **sparseFLMM**, **switchnpreg**, **warpMix**.

(* = core package)

New packages in CRAN task views

Bayesian **BayesVarSel**, **NetworkChange**, **bridgesampling**, **deBInfer**, **gRain**, **revdbayes**.

Cluster **idendr0**, **prcr**.

Distributions **Renext**, **kernelboot**, **revdbayes**.

Econometrics **REndo**, **margins**, **pco**, **rdlocrand**.

ExtremeValue **Renext**, **revdbayes**.

Finance BCC1997, BLModel, PortfolioOptim, RcppQuantuccia, Sim.DiffProc, rpatrec, rpgm.

MachineLearning LTRCtrees, MXM, RLT, RcppDL, darch, deepnet, gradDescent, opusminer, wsrf, xgboost.

MetaAnalysis CPBayes, metafuse, metacov, metaviz, surrosurv.

OfficialStatistics haven, micEconIndex, missForest.

Pharmacokinetics NonCompart, PKNCA, PKgraph, PKreport, cpk, dfpk, mrgsolve, ncappc, ncar, nmw, pkr, scaRabee.

Phylogenetics idendr0, nLTT, phylocanvas.

Psychometrics AnalyzeFMRI, BTLLasso, ThreeWay, eegkit, ica, multiway, munfold.

Spatial FRK, sperrorest, spind, spmoran, starma.

TimeSeries dataseries, mafs, prophet, robustarima.

WebTechnologies RMixpanel.

gR DiagrammeR.

(* = 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 3.5 was released on 25 April, 2017. It is compatible with R 3.4 and consists of 1383 software packages, 316 experiment data packages, and 911 up-to-date annotation packages. The [release announcement](#) includes descriptions of 88 new packages, and updated NEWS files for many additional packages. Start using Bioconductor by installing the most recent version of R and evaluating the commands

```
source("https://bioconductor.org/biocLite.R")
biocLite()
```

Install additional packages and dependencies, e.g., [AnnotationHub](#), with

```
BiocInstaller::biocLite("AnnotationHub")
```

[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:

- [bioconductor.org](#) to install, learn, use, and develop Bioconductor packages.
- A listing of [available software](#), linked to pages describing each package.
- A question-and-answer style [user support site](#) and developer-oriented [mailing list](#).
- The [F1000Research Bioconductor channel](#) for peer-reviewed Bioconductor work flows.
- Our [package submission](#) repository for open technical review of new packages.

Our annual conference, [BioC 2017: Where Software and Biology Connect](#), will be on June 26 ('developer day'), 27 and 28, in Boston, MA.

Bioconductor Core Team
Biostatistics and Bioinformatics
Roswell Park Cancer Institute, Buffalo, NY
USA maintainer@bioconductor.org

Changes in R

From version 3.3.3 to version 3.4.1

by R Core Team

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 NAs, also in its result no longer setting them to `0`. Further, a new logical option `addNA` allows to count NAs 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(<loooooooooong_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 NAs.
- `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 ≥ 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_DEFS` 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 NAs or NaNs, 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 \emptyset . (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 NAs, 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 NAs 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 `f "{"`.
- (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).