The R Journal

Volume 11/1, June 2019

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

Contents

Contributed Research Articles

News and Notes

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

Email:
r-journal@R-project.org

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

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

Editorial

by Norm Matloff

The editorial board and I are pleased to present the latst issue of the *R Journal.*

We apologize that this issue has been so late in publication. As this is my first issue as Editor-in-Chief, I must personally thank Roger Bivand and John Verzani, the two previous EiCs, for their guidance in the technical aspects of putting an issue together.

The good news, though, is that publication should be much more timely in the future, due to improved internal technical documentation and the hiring of the journal's first-ever editorial assistants, Stephanie Kobakian and Mitchell O'Hara-Wild. We are thankful to the R Consortium for a grant supporting the assistants (https://rjpilot.netlify.com).

This issue is chock full of interesting papers, many of them on intriguing, unusual topics. For those of us whose connection to R goes back to the old S days, it is quite gratifying to see the wide diversity of application areas in which R has been found productive.

Regular readers of this journal are aware of a change in policy that began January 2017, under which we are moving away from a paradigm in which a typical article is merely an extended user's manual for the author's R package.

To be sure, most articles will continue to be tied to specific packages. But we hope for broader coverage, and even the package-specific articles should emphasize aspects such as technical challenges the package needed to overcome, how it compares in features and performance to similar packages, and so on. As described in the announcement:

> Short introductions to contributed R packages that are already available on CRAN or Bioconductor, and going beyond package vignettes in aiming to provide broader context and to attract a wider readership than package users. Authors need to make a strong case for such introductions, based for example on novelty in implementation and use of R, or the introduction of new data structures representing general architectures that invite re-use.

Clearly, there is some subjectivity in assessing these criteria, and views will vary from one handling editor to the next. But this is the current aim of the journal, so please keep it in mind in your submissions.

We wish the journal to further evolve in two more senses:

- In 2016, the American Statistical Assocation released a dramatic policy statement, seriously questioning the general usefulness and propriety of p-values. Though the statement did not call for a ban on the practice, it did have a strong theme that p-values should be used more carefully and less often. Many of us, of course, had been advocating a move away from p-values for years. We wish authors of future submissions to the journal to be mindful of the ASA policy statement. We hope for reduced emphasis on hypothesis testing, and in articles that do include testing, proper consideration of power calculation.

- In the interest of reproducibility—a requirement already imposed by the journal on article submissions—we will require that any real datasets used as examples in an article must be provided. Note that this will mean that datasets with privacy issues or datasets of extremely large size should not be used in an article.

Finally, we note our deep appreciation for the anonymous reviewers. A journal is only as good as its reviewers, and most reviews are quite thoughtful and useful. If a handling editor solicits your review for a paper, please make some time for it. And if you must decline the request, a reply to that effect would be quite helpful; don't just discard the editor's e-mail message. The handling editors are quite busy, and it is unfair to both them and the authors to have the editors wait until they must conclude you will not reply, causing unnecessary delay.

Bibliography

R. Wasserstein and N. Lazar. The ASA's Statement on p-Values: context, process, and purpose. *The American Statistician*, 85(15):129–133, 2016. [p165]

Norm Matloff
Dept. of Computer Science
University of California, Davis
Davis, CA 95616
USA matloff@cs.ucdavis.edu

Matching with Clustered Data: the CMatching Package in R

by Massimo Cannas and Bruno Arpino

Abstract Matching is a well known technique to balance covariates distribution between treated and control units in non-experimental studies. In many fields, clustered data are a very common occurrence in the analysis of observational data and the clustering can add potentially interesting information. Matching algorithms should be adapted to properly exploit the hierarchical structure. In this article we present the CMatching package implementing matching algorithms for clustered data. The package provides functions for obtaining a matched dataset along with estimates of most common parameters of interest and model-based standard errors. A propensity score matching analysis, relating math proficiency with homework completion for students belonging to different schools (based on the NELS-88 data), illustrates in detail the use of the algorithms.

Background

Causal inference with observational data usually requires a preliminary stage of analysis corresponding to the *design* stage of an experimental study. The aim of this preliminary stage is to reduce the imbalance in covariates distribution across treated and untreated units due the non-random assignment of treatment before estimating the parameters of interest. Matching estimators are widely used for this task (Stuart, 2010). Matching can be done directly on the covariates (multivariate matching) or on the propensity score (Rosenbaum and Rubin, 1983). The latter is defined as the probability of the treatment given the covariates value and it has a central role for the estimation of causal effects. In fact, the propensity score is a one dimensional summary of the covariates and thus it mitigates the difficulty of matching observations in high dimensional spaces. Propensity score methods have flourished and several techniques are now well established both in theory and in practice, including stratification on the propensity score, propensity score weighting (PSW), and propensity score matching (PSM).

Whilst the implementation of matching techniques with unstructured data has became a standard tool for researchers in several fields (Imbens and Rubin, 2016), the increasing availability of clustered (nested, hierarchical) observational data poses new challenges. In a clustered observational study individuals are partitioned into clusters and the treatment is non-randomly assigned in each cluster so that confounders may exist both at the individual and at the cluster level. Note that this framework is different from clustered observational data where a treatment is non-randomly assigned for all units in the cluster, for which an optimal matching strategy has been suggested by Zubizarreta and Keele (2017). Such nested data structures are ubiquitous in the health and social sciences where patients are naturally clustered in hospitals and students in schools, just to make two notable examples. If relevant confounders are observed at both levels then a standard analysis, adjusting for all confounders, seems reasonable. However, when only the cluster label — but not the cluster level variables — is observed there is not a straightforward strategy to exploit the information on the clustering. Intuitively, the researcher having a strong belief on the importance of the cluster level confounders may adopt a '*within-cluster*' matching strategy. On the other extreme, a researcher may decide to ignore the clustering by using only the *pooled* data. It is important to note that this pooling strategy implicitly assumes that cluster level variables are not important confounders. Indeed, there have been a few proposals to adapt PSW and PSM to clustered data, see Cafri et al. (2018) for a review. Li et al. (2013) proposed several propensity score weighting algorithms for clustered data showing, both analytically and by simulation, that they reduce the bias of causal effects estimators when "clusters matter," that is, when cluster level covariates are important confounders. In the PSM context, Arpino and Mealli (2011) proposed to account for the clustering in the estimation of the propensity score via multilevel models. Recently, Rickles and Seltzer (2014) and Arpino and Cannas (2016) proposed caliper matching algorithms to perform PSM with clustered data. As we will discuss shortly, these algorithms can be used not only for PSM but also in the more general context of multivariate matching.

In the remaining of this paper, after reviewing the basic ideas underlying matching estimators, we briefly describe the available packages for matching in the R environment. Then, we describe the algorithms for matching with clustered data proposed by Arpino and Cannas (2016) and we present the package **CMatching** implementing these algorithms. The applicability of these algorithms is very broad and refers to all situations where cluster-level data are present (in medicine, epidemiology, economics, etc.). A section is devoted to illustrate the use of the package on data about students and schools, which is a common significant occurrence of clustered data.

Packages for matching unstructured data in R

A list of the most important packages for matching available for R users is shown in Table 1. The **Matching** package, which is required to run **CMatching**, is a remarkably complete package offering several matching options. `Matching` implements many greedy matching algorithms including genetic matching (Diamond and Sekhon, 2013). It also contains a general `MatchBalance` function to measure pre- and post-matching balance with a large suite of diagnostics. As for optimal matching, there are dedicated packages like **designmatch** and **optmatch**. The latter can also be called from **MatchIT**, a general purpose package implementing also the Coarsened Exact Matching approach of Iacus et al. (2011). Full matching is a particular form of optimal matching implemented by **quickmatch** with several custom options.

Package	Description	Reference
Matching	Greedy matching and balance analysis	Sekhon (2011)
MatchIT	Greedy matching and balance analysis	Iacus et al. (2011)
optmatch	Optimal matching	Hansen and Klopfer (2006)
quickmatch	Generalized full matching	Savje et al. (2018)
designmatch	Optimal matching and designs	Zubizarreta et al. (2018)

Table 1: General purpose packages available from CRAN implementing matching algorithms. The list is not exhaustive as there are several packages covering specialized matching routines: a list can be found at http://www.biostat.jhsph.edu/ estuart/propensityscore-software.html.

At the time of writing none of the packages described above offers specific routines for clustered data. The **CMatching** package fills this important gap implementing the algorithms for matching clustered data described in the next section.

Matching clustered data

Let us consider a clustered data structure $D = \{y_{ij}, x_{ij}, t_{ij}\}, \quad i = 1, \cdots n_j, \quad j = 1, \cdots J$. For observation i in cluster j we observe a vector of covariates X and a binary variable T specifying the treatment status of each observation. Here $n = \sum n_j$ is the total number of observations and J is the number of clusters in the data. We observe also a response variable Y whose average value we are willing to compare across treated and untreated units. A matching algorithm assigns a (possibly empty) subset of control units to each treated unit. The assignment is made with the aim of minimizing a loss function, typically expressed in terms of covariates distance between treated and untreated units. Matching algorithms can be classified as greedy or optimal depending whether the cost function is minimized locally or globally, respectively. Optimal matching algorithms are not affected by the order of the units being matched so they can reach the global optimum, but they are typically more computer-intensive than greedy algorithms proceedings step by step. To bound the possibility of bad matches in greedy matching, it is customary to define a maximum distance for two units to be matched, i.e., a *caliper*, which is usually expressed in standard deviation units of the covariates (or of the propensity score). one covariate.

A greedy matching procedure can then be articulated in the following steps:

1. fix the caliper;
2. match each treated with control unit(s) at minimum covariates distance (provided that distance $<$ caliper);

3. measure the residual covariates' imbalance in the matched dataset and count the number of unmatched units (drops);
4. carefully consider both the balance and the number of drops: if they are satisfactory then proceed to the outcome analysis; otherwise stop or revise previous steps.

If matching proves successful in adjusting covariates, the researcher can proceed to outcome analysis where the causal estimand of interest is estimated from the matched data using a variety of techniques (Ho et al., 2007). On the other hand, if the procedure gives either an unsatisfactory balance or an excessive number of unmatched units, the investigator may try to modify some aspects of the procedure (e.g., the caliper, the way the distance is calculated).

Conceptually, the same procedure can be used also for hierarchical data. Indeed, it is not atypical to find analysis ignoring the clustering and pooling together all units. A pooling strategy implicitly assumes that the clustering is not relevant. However, in several cases the clustering does matter, that is, the researcher can hypothesize or suspect that some important cluster-level confounders are unobserved. In this case, the information on the cluster labels can be exploited in at least two ways: i) forcing the matching to be implemented *within-cluster* only; ii) performing a *preferential* within-cluster matching, an intermediate approach between the two extremes of pooled and within-cluster matching (Arpino and Cannas, 2016). A within-cluster matching can be obtained by modifying step 2 above in the following way:

2' match each treated with the control unit(s) *in group j* at minimum covariate distance (provided that distance $<$ caliper).

This procedure may result in a large number of unmatched units (drops) so it increases the risk of substantial bias due to incomplete matching (Rosenbaum and Rubin, 1985), in particular when the clusters are small. This particular bias arises when a matched subset is not representative of the original population of treated units because of several *non random* drops. Even in the absence of bias due to incomplete matching, a high number of drops reduces the original sample size with possible negative consequences in terms of higher standard errors.

It is possible to profit as much as possible of the the exact balance of (unobserved) cluster-level covariates by first matching within clusters and then recovering some unmatched treated units in a second stage. This leads to the preferential within-cluster matching, which can be obtained by modifying step 2 above in the following way:

2" *a)* match each treated with the control units(s) *in group j* at minimum covariate distance (distance $<$ caliper);

2" *b)* match each unmatched treated unit from previous step with the control unit(s) at minimum covariate distance in some group different from j (provided that distance $<$ caliper).

Now consider the outcome variable Y. We can define for each unit potential outcomes $Y1$, $Y0$ as the outcome we would observe under assignment to the treatment and control group, respectively (Holland, 1986). Causal estimands of interest are the Average Treatment effect: $ATE = E[Y1 - Y0]$ or, more often, the Average Treatment effect on the treated: $ATT = TE[Y1 - Y0]$. Given that a unit is either assigned to the treatment or control group it is not possible to directly observe the individual causal effect on each unit; we have $Y = T \cdot Y1 + (1 - T) \cdot Y0$. In a randomized study T is independent of $(Y0, Y1)$ so, for $k = 0, 1$, we have

$$E(Yk) = E(Yk \,|\, T = k) = E(Y \,|\, T = k)$$

which can be estimated from the observed data. In a observational study, matching can be used to balance covariates across treated and control units and then the previous relation can be exploited to impute the unobserved potential outcomes from the *matched dataset.* In our clustered data context, after the matched dataset has been created using one of the algorithms above, the ATT and its standard error can be estimated using a simple regression model:

$$Y_{ij} = \alpha_j + T_{ij}\beta \tag{1}$$

that is, a linear regression model with clustered standard errors to take into account within-cluster dependence in the outcome (Arpino and Cannas, 2016). The resulting ATT estimate is the difference of outcome means across treated and controls, i.e., $\widehat{ATT} := mean(Y \,|\, T = 1) - mean(Y \,|\, T = 0)$, computed on the matched data. Standard errors are calculated using the cluster bootstrapping method for estimating variance-covariance matrices proposed by Cameron et al. (2011) and implemented in the package **multiwayvcov**. In general, calculating standard errors for PSM in clustered observational studies is a difficult problem requiring prudence from the researcher. While close formulae exist for weighting estimators (Li et al., 2013), standard error estimation after PSM matching relies upon approximation (Cafri et al., 2018), modelling assumption (Arpino and Cannas, 2016), or simulation (Rickles and Seltzer, 2014).

Multisets and matching output

In this section we briefly detail the two routines in the **CMatching** package. Multisets are useful to compactly describe pseudo code so we recall some definitions and basic properties herein. A multiset is a pair $\{U, m\}$ where U is a given set, usually called universe, and $m : x \to \mathbb{N} \cup \{0\}$ is the multiplicity function assigning each $x \in U$ its frequency in U. Both the summation symbol and union symbols are used to manipulate multisets and they have different meanings: if A and B are multisets then $C \equiv A \cup B$ is defined by $m_C(x) = max(m_A(x), m_B(x))$, while $C \equiv A + B$ is defined by $m_C(x) = m_A(x) + m_B(x)$. For example if $A \equiv \{1, 2, 2\}$ and $B \equiv \{1, 2, 3\}$ then $A \cup B \equiv \{1, 2, 2, 3\}$, $A + B \equiv \{1, 1, 2, 2, 2, 3\}$. In our framework U is the set of observations indexes and thus m gives information about the number of times a given observation occurred in the matched dataset. Multisets then allow to naturally represent multiple matches arising from matching with replacement. When using multiset notation to describe the output of a matching algorithm, we are implicitly overlooking the fact that the output of a matching algorithm is richer as it also brings out the pairings, i.e., the associations between matched treated and untreated observations. However, it can be noted that this pairing is not relevant for calculating common estimates or common balance measures (e.g., the ATT), as they are invariant to permutations of the labels of the matched observations.

The routines `MatchW` and `MatchPW`

We denote with W_i and $\overline{W}_i$ the sets of treated observations matched within clusters and unmatched within clusters, respectively. In Algorithms 1 and 2, the summation symbol ($\sum$) denotes multiset sum.

```
# Algorithm 1: within-cluster matching
procedure MatchW(data)
    find $W_i$ for each i using Match function
    M := $\sum_i W_i$ # find matched within
    mdata := data[M] # extracts matched data
    if data contains outcome variable Y:
        estimate $\widehat{ATT}$ and $sd(\widehat{ATT})$ from model on mdata
    else $\widehat{ATT}$ <- $sd(\widehat{ATT})$ <- NULL
    return mdata, $\widehat{ATT}$ and $sd(\widehat{ATT})$

# Algorithm 2: for preferential within-cluster matching
Procedure{MatchPW(data)
    find $W_i$ and $\overline{W}_i$ for each i using Match function
    B := Match($\sum_i \overline{W}_i \cup$ all controls)
    M := $\sum_i W_i + B$
    mdata := data[M]  # extracts matched data
    if data contains outcome variable Y:
        estimate $\widehat{ATT}$ and $sd(\widehat{ATT})$ from model on mdata
    else $\widehat{ATT}$ <- $sd(\widehat{ATT})$ <- NULL
    return mdata, $\widehat{ATT}$ and $sd(\widehat{ATT})$
```

In the first two lines, common to both algorithms, the `Match` function is repeatedly run to produce the matched-within subsets M_i $i = , ..., J$. Then, in Algorithm 1 the sum of the M_i in line 3 gives the matched subset M. Algorithm 2 is similar but after finding the M_i's an "additional" subset B is found by recovering some unmatched units (line 3) and then combined to give the final matched dataset. If a response variable Y was included the output of both algorithms also contains an estimate of the ATT (default, but the user can choose also other estimands) and its standard error.

Functions in the CMatching package

CMatching can be freely downloaded from CRAN repository and it contains the functions listed in Table 2. The main function `CMatch` performs within-cluster and preferential within-cluster matching via subfunctions `MatchW` and `MatchPW`, respectively. The output of the main function can be passed to functions `CMatchBalance` and `summary` to provide summaries of covariates balance and other characteristics of the matched dataset. `CMatch` exploits the `Match` function (see **Matching**) implementing matching for unstructured data. Given a covariate `X` and a binary treatment `T`, the call `Match(X,T,...)` gives the set of indexes of matched treated and matched control units. The `CMatch` function has the same arguments plus the optional arguments `G` (specifying the cluster variable) and `type` to choose between within-cluster matching or preferential-within-cluster matching. We highlight that we chose to frame the **CMatch** in the `Match` function style so that `Matching` users can easily implement PSM with clustered data in a familiar setting.

Function	**Description**	Input	Output
`CMatch`	Match	X, T, G	A matched dataset
`MatchW`	Match within	X, T, G	A matched dataset
`MatchPW`	Match preferentially within	X, T, G	A matched dataset
`summary.Match`	S3 method for CMatch objects	A matched dataset	General summaries
`CMatchBalance`	Balance analysis	A matched dataset	Balance summaries

Table 2: Main input and output of functions in **CMatching** package.

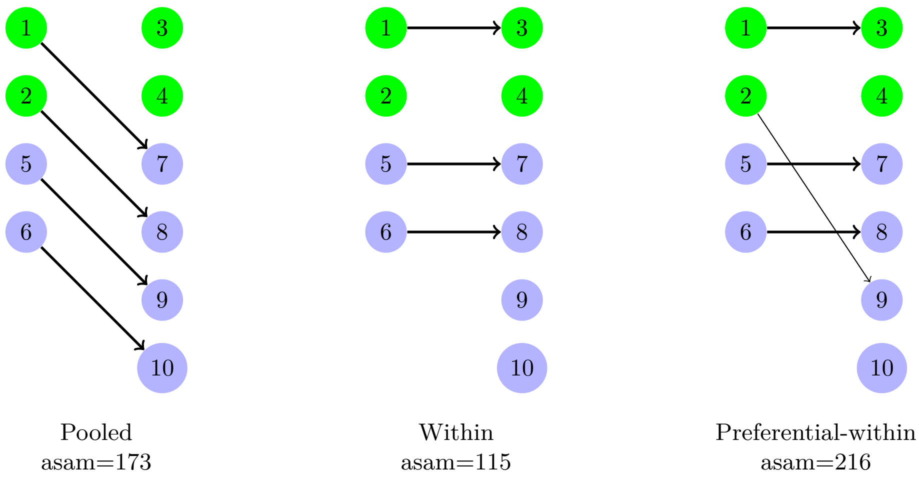


Figure 1: Different matching solutions for the toy dataset (`caliper = 2`). Green and violet circles indicate cluster 1 and cluster 2 units, respectively; arrows indicate matched pairs of treated (left) and control units (right). For each matching we report the absolute percent standardized mean difference of x in the matched subset (`asam`), a measure of residual imbalance.

A simple usage example

For an illustration let us consider an artificial dataset consisting of two clusters, the first containing two treated units and two control units, and the second containing two treated and four controls.

We use `g` for the cluster identifier, `x` for the value of the individual level confounder, and `t` for the binary treatment indicator:

```
> toy
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
id    1    2  3.0  4.0  5.0    6    7    8  9.0  10.0
g     1    1  2.0  2.0  1.0    1    2    2  2.0   2.0
t     1    1  1.0  1.0  0.0    0    0    0  0.0   0.0
x     1    1  1.1  1.1  1.4    2    1    1  1.3   1.3
```

We also fix a caliper of 2 (in standard deviation units of X, i.e., all units at distance greater or equal than $2 \cdot sd(x) \approx 2 \cdot 0.312 = 0.624$ will not be matched together) and we assume that the ATT is the target parameter. Although artificial the dataset aims at representing the general situation where pooled matching results in matched treated and control units not distributed homogeneously across clusters (see Figure 1, left).

The pooled, within and preferential-within matchings for the toy data are depicted in Figure 1. For each matching we report the `asam`, a measure of residual imbalance given by the absolute percent mean difference of x across treated and controls divided by standard deviation of the treated observations alone. The asam is widely used as a measure of imbalance (Stuart, 2010); its value in the unmatched data is 491. The pooled matching (left) is a complete matching, i.e., all the treated units could be matched. However, note that units in pairs 1-7 and 2-8 may differ in cluster level covariates. Matching within-cluster (center) guarantees perfect balance in cluster level covariates but it is incomplete because unit 2 cannot be matched within-cluster with the given caliper. This is a typical disadvantage of within-cluster matching with smaller clusters. Unit 2 is matched with 9 in the preferential within matching (right), which again is a complete matching.

The above matching solutions can be obtained easily using `CMatch` as follows. For the pooled matching it is enough to call `Match` (or, equivalently, `CMatch` without `type` specification) while for within and preferential-within matching it is enough to specify the appropriate `type` in the `Match` call:

```
#pooled matching
pm <- Match(Y=NULL, Tr=t, X=x, caliper=2)

# same output as before (with a warning about the absence of groups,
# ties=FALSE,replace=FALSE)
pm <- CMatch(type="within", Y=NULL, Tr=t, X=x, Group=NULL, caliper=2)

#within matching
wm <- CMatch(type="within", Y=NULL, Tr=t, X=x, Group=g, caliper=2)

#preferential-within matching
pwm <- CMatch(type="pwithin", Y=NULL, Tr=t, X=x, Group=g, caliper=2)
```

The output of these object is quite rich. However, a quick look at the matchings can be obtained by directly calling the index set of matched observations:

```
> pm$index.treated; pm$index.control
[1]  1  2   5  6
[1]  7  8  10 9
> wm$index.treated; wm$index.control
[1] 1 5 6
[1] 3 7 8
> pwm$index.treated; pwm$index.control
[1] 1 2 5 6
[1] 3 7 8 10
```

Note that vertical alignments in the table above correspond to arrows in Figure 1. With larger datasets and when multiple matches are allowed (i.e., when `replace=TRUE`) it is probably better to summarize the output. The output objects are of class `"CMatch"` and a `summary` method is available for these objects. The summary shows the number of treated and the number of controls *by group*. Moreover, when Y is not NULL it also shows the point estimate of ATT with its model-based estimate of the standard error:

```
> summary(wm)

Estimate...  0
```

```
SE.........  NULL

Original number of observations.............  10
Original number of treated obs..............  4
Original number of treated obs by group......

1 2
2 2

Matched number of observations..............  3
Matched number of observations  (unweighted).  3

Caliper (SDs)........................................   2
Number of obs dropped by 'exact' or 'caliper' .........   1
Number of obs dropped by 'exact' or 'caliper' by group

1 2
1 0
```

This summary method does not conflict with the corresponding method for class "Match" which is still available for objects of that class. From the summary above (see also Figure 1, center) we can easily see that matching within groups resulted in one unmatched treated unit from group two. The exact pairing can be recovered from the full output, in particular from the object mdata containing the list of matched and unmatched units. As we noticed in the introduction, it is essential to analyze covariate balance to evaluate the effectiveness of matching as a balancing tool. To this end objects of class "CMatch" can be input of the CMatchBalance function, a wrapper of MatchBalance which offers a large number of balance diagnostics:

```
> CMatchBalance( t~x , match.out=wm)

***** (V1) x *****
                       Before Matching                  After Matching
mean treatment........       1.05                      1.0667
mean control..........     1.3333                      1.1333
std mean diff.........    -490.75                     -115.47
(...)
```

From the output we see that the asam decreased from 491 to 115. One can more directly obtain the standardized difference in means of these matchings by subcomponents:

```
> bwm$After[[1]]["sdiff"]
$sdiff
[1] -115.4701

> bpwm$After[[1]]["sdiff"]
$sdiff
[1] -216.5064
```

Whilst artificial, the previous example prompts some general considerations:

- forcing within-cluster matching may result in suboptimal matches with respect to pooled matching. In the toy example, unit 2 is best matched with unit 8 but it is unmatched when type=within is chosen (or it could be matched with a less similar control in the same cluster). In both cases the increased bias (due to either incomplete matching or greater imbalance in the observed x) may be at least in part compensated by lower imbalance in cluster level variables;
- preferential-within matching may occasionally recover all unmatched treated units in the within step by matching them between in step 2. However, this complete matching is generally different from the complete pooled matching obtained by ignoring the clustering. In the toy example, unit 2 is recovered in the preferential step but the final matching has a higher imbalance than the pooled one. Again, it is up to the researcher to tune the trade-off between bias due to incomplete matching and bias due to unobserved differences in group covariates.

In applications, when cluster level confounders are unobserved, it is not straightforward to decide which of the matching strategies is the best. However, combining the within and preferential-within routines offered by **CMatching** with sound subject matter knowledge, the researcher can decide how much importance should be given to balance within-clusters based on the hypothesized strength of unobserved cluster level confounders. Note that **CMatching** uses the same caliper for clusters, under the assumption that the researcher is typically interested in estimating the causal effect of the treatment in the whole population of treated units and not by each cluster. This is the main difference between `MatchW` and `Matchby` from the **Matching** package. The latter exactly matches on a categorical variable and the caliper is recalculated in each subset and for this reason `MatchW` estimates generally differ from those obtained from `Matchby.` Another difference is that `Matchby` does not adjust standard errors for within-cluster correlation in the outcome. A third difference is that CMatching provides some statistics (e.g., number of drops) by cluster to better appreciate how well the matched dataset resembles the original clustered structure in terms for example of cluster size.

Demonstration of the CMatching package on NELS-88 data

The **CMatching** package includes several functions designed for matching with clustered data. In this section we illustrate the use of **CMatching** with a an educational example.

Schools dataset

The example is based on data coming from a nationally representative, longitudinal study of 8th graders in 1988 in the US, called NELS-88. **CMatching** contains a selected subsample of the NELS-88 data provided by Kraft and de Leeuw (1998) with 10 handpicked schools from the 1003 schools in the full dataset. The subsample, named `schools`, is a data frame with 260 observations on 19 variables recording either school or student's characteristics.

For the purpose of illustrating matching algorithms in **CMatching**, we consider estimation of the causal effect of doing homework on mathematical proficiency. More specifically, our treatment is a binary variable taking value 1 for students that spent *at least* one hour doing math homework per week and 0 otherwise. The latter is a transformation of the original variable "homework" giving two almost equal-sized groups. The outcome is math proficiency measured on a discrete scale ranging from 0 to 80. For simplicity we first attach the dataset (`attach(schools)`) and name the treatment and the outcome variables as `T` and `Y`, respectively. The variable `schid` is the school identifier and we rename it as `Group`:

```
> T <- ifelse(homework>1, 1, 0)
> Y <- math
> Group <- schid
```

Since the NELS-88 study was an observational study, we do not expect a simple comparison of math scores across treated and control students (those doing and those not doing math homework) to give an unbiased estimate of the "homework effect" because of possible student-level and school-level confounders. For the purpose of our example, we will consider the following student-level confounders: "ses" (a standardised continuous measure of family socio-economic status), "sex" (1 = male, 2 = female) and "white" (1 = white, 0 other race). The NELS-88 study also collected information on school characteristics. In general, a researcher needs to include all potential confounders in the matching procedure, irrespective of the hierarchical level. Here we considered one school-level confounder: "public" (1 = public schools, 0 = private) but it is plausible to assume that one or more relevant confounders at the school-level are not available. It is clear that, to make the unconfoundedness assumption more plausible, richer data should be used. For example, students' motivation and parents' involvement are potentially important confounders. Thus, the following estimates should be interpreted with caution.

Before illustrating the use of **CMatching**, it is useful to get a better understanding of the data structure. In the school dataset we have a fairly balanced number of treated and control units (128 and 132, respectively). However, in some schools we have more treated than control students, with proportion of treated ranging from 20% to 78%:

```
> addmargins(table(Group, T))

       T
Group     0   1 Sum
  7472   17   6  23
  7829    6  14  20
  7930   12  12  24
  24725  15   7  22
  25456  17   5  22
  25642  16   4  20
  62821  15  52  67
  68448   8  13  21
  68493  15   6  21
  72292  11   9  20
  Sum   132 128 260
```

From the table above we can notice that the total school sample size is fairly homogeneous with the exception of one school (code = 62821) where the number of treated students (52) is considerably higher than the number of control students (15). These considerations are important for the implementation of the within-cluster and preferential within-cluster matching algorithms. In fact, within-cluster matching can be difficult in groups where the proportion of treated units is high because there are relatively few controls that can potentially serve as a match. Preferential-within-cluster matching would be less restrictive.

This preliminary descriptive analysis is also useful to check if treated and control units are present in each group. In fact, if a group is only composed of treated or control students then within-cluster matching cannot be implemented. Groups with only treated or controls should be dropped before the within-cluster matching algorithm is implemented. We now describe how **Cmatching** can be used to implement matching in our school-clustered dataset.

Propensity score matching

CMatching requires to estimate the propensity score before implementing the matching. Here we estimate propensity scores using a simple logistic regression with only main effects and then estimate the predicted probability for each student:

```
> pmodel <- glm(T~ses + as.factor(sex) + white + public, family=binomial(link="logit"))
> eps <- fitted(pmodel)
```

We do not report the output of the propensity score model because in PSM the propensity score is only instrumental to implement the matching. Within-cluster propensity score matching can be implemented by using the function `CMatch` with the option `type="within"`:

```
> psm_w <- CMatch(type="within", Y=Y, Tr=T, X=eps, Group=Group)
```

The previous command implements matching on the estimated propensity score, `eps`, by using default settings of `Match` (one-to-one matching with replacement and a caliper of 0.25 standard deviations of the estimated propensity score). The output is an object of class `"CMatch"` and a customized summary method for objects of this class gives the estimated ATT and the main features of the matched dataset:

```
> summary(psm_w)

Estimate...  5.2353
SE.........  2.0677

Original number of observations..............  260
Original number of treated obs...............  128
Original number of treated obs by group......

 7472  7829  7930 24725 25456 25642 62821 68448 68493 72292
    6    14    12     7     5     4    52    13     6     9

Matched number of observations...............  119
Matched number of observations  (unweighted).  120

Caliper (SDs)........................................   0.25
Number of obs dropped by 'exact' or 'caliper' .........   9
Number of obs dropped by 'exact' or 'caliper' by group

 7472  7829  7930 24725 25456 25642 62821 68448 68493 72292
    0     2     0     0     0     0     2     5     0     0
```

The summary starts reporting the original total number of students in our sample (260), the total number of treated students (128) and how they are distributed across the different schools.

It is of utmost importance to check how many treated units could be matched to avoid bias from incomplete matching. For this reason the output indicates that 119 students in the treatment group found a match ("`Matched number of observations`"), while the remaining 9 were dropped. Note that the unweighted number of treated observations that found a match ("`Matched number of observations (unweighted)`") is different because of *ties*. Ties management can be controlled by option `ties` of the `Match` function: if `ties=TRUE` when one treated observation matches with more than one control unit all possible pairs are included in the matched dataset and each pair is weighted equal to the inverse of the number of matchedcontrols. If instead `ties=FALSE` is used ties are randomly broken. Note that the summary reports the number of *treated* matched and dropped units because it is assumed by default the ATT is the target estimand. Then the output also reports how the 9 unmatched treated students are distributed across schools. For example, we notice that in one school (68448), 5 of the 13 treated students did not find a match. This is because for these 5 students there was no control student in the same school with a propensity score falling within the caliper. The report also recalls the caliper, which was set to 0.25 standard deviations of the estimated propensity score in this example. The caliper can be set in standard deviation units using the homonymous argument `caliper`. It may be more useful to calculate the percentage of dropped units instead of the absolute numbers. These percentages are not reported in the summary but they can be easily retrieved from the `CMatch` output. For example, we can calculate the percentage of unmatched treated units, both overall and by school:

```
# percentage of drops
> psm_w$ndrops / psm_w$orig.treated.nobs

[1] 0.0703

# percentage of drops by school
> psm_w$orig.dropped.nobs.by.group / table(Group)

Group
 7472  7829  7930 24725 25456 25642 62821 68448 68493 72292
 0.00  0.10  0.00  0.00  0.00  0.00  0.03  0.24  0.00  0.00
```

confirming that the percentage of drops is very low in all schools. We could also similarly calculate the percentage of drops over *treated* observations, which turn out to be high for school `64448`. The next step before accepting a matching solution is the evaluation of the achieved balance of confounders across the treatment and control groups. To this end the package contains function `CMatchBalance` that can be applied to an object of class `"CMatch"` to compare the balance before and after matching (the matched dataset must be specified in the `match.out` argument):

```
> b_psm_w <- CMatchBalance(T~ses + as.factor(sex) + white + public, data=schools,
                            match.out=psm_w)

***** (V1) ses *****
                       Before Matching          After Matching
mean treatment........    0.23211                  0.24655
mean control..........   -0.36947                  0.14807
std mean diff.........     61.315                   10.086


***** (V2) as.factor(sex)2 *****
                       Before Matching          After Matching
mean treatment........    0.52344                  0.52941
mean control..........    0.46212                  0.56303
std mean diff.........     12.229                   -6.706


***** (V3) white *****
                       Before Matching          After Matching
mean treatment........    0.73438                   0.7563
mean control..........     0.7197                  0.71429
std mean diff.........     3.3103                   9.7458


***** (V4) public *****
                       Before Matching          After Matching
```

```
mean treatment........    0.59375                   0.57983
mean control..........    0.88636                   0.57983
std mean diff.........    -59.346                         0

(...)
```

The output reports the balance for each variable to allow the researcher to assess the effectiveness of matching in balancing each variable. Many balance metrics are provided but for simplicity of exposition we focus on comparing on the standardized mean difference between the two groups of students. Note that the asam can be easily obtained by averaging the standardized mean differences:

```
vec   <-vector()
for(i in 1:length()) {vec[[i]] <- b_psm_w$AfterMatching[[i]]$sdiff}
> mean(abs(vec))
[1] 6.634452
```

from which we can see that the initial asam of 34 (see Table 3) sharply decreased after matching. Balance improved dramatically for `ses` and `public` (results not shown). In fact, for the latter it was possible to attain exact matching. This is guaranteed by within-cluster matching because it forces treated and control students to belong to the same school. For the same reason, within-cluster matching also guarantees a perfect balance of all other school-level variables (even unobservable) not included in the propensity score estimation. The balance improved also for the `sex` variable but not for the dummy `white` (from 3.31% to 9.75%). In a real study, the investigator may attain a matching solution with an even better balance of the dummies for white and sex by changing the propensity score model or one or more options in the matching algorithms. For example, a smaller caliper could be tried. Note that `CMatchBalance` is a wrapper of the `MatchBalance` function (**Matching** package) so it measures balance globally and not by group. This is acceptable also in a hierarchical context since we first and foremost consider the overall balance. While a group-by-group balance analysis may be useful it is only the *average* `asam` which matters when estimating the ATT on the *whole* population of treated units.

We highlighted that the within-cluster algorithm always guarantees a perfect balance in all cluster-level confounders as in the example above. However, note that it was not possible to match some treated observations and in part this can be due to the matching constrained to happen only within clusters. The researcher can relax the constraint using preferential within-cluster matching. This algorithm can be invoked using the option `type="pwithin"` in the `CMatch` function:

```
> psm_pw <- CMatch(type="pwithin", Y=Y, Tr=T, X=eps, Group=Group)
> summary(psm_pw)
```

A comparison of results between within and preferential within matching is given in Table 3. The preferential within-cluster matching was able to match all treated students (`"Matched number of obs"` = 128), i.e., the number of unmatched treated students is 0 (`"Number of drops"` = 0). In this example, the 9 treated students that did not find a matched control within the caliper in the same school found a control match in another school. Looking to the overall balance attained by matching preferentially within, we can notice that preferential within-cluster matching showed a slightly higher asam than within-matching. In fact there is no clear "winner" between the two algorithms: the balance of the individual level variables `ses` and `white` improves slightly with the preferential within-cluster matching while for `sex` the within-cluster matching is considerably better (not shown). Importantly, using preferential within-cluster matching the absolute standardized mean difference for the school-level variable `public` is 3.2% This is not a high value because most of the treated units actually found a match within schools. However, this finding points to the fact that preferential within-cluster matching is not able to perfectly balance cluster level variables as the within-cluster approach.

Finally, having achieved a satisfactory balance with a very low number of drops we can estimate the ATT on the matched dataset. When the argument `Y` is not `NULL`, an estimate is automatically given otherwise the output of the `CMatch` function only gives information about the matching. The estimated average effect of studying math for at least one hour per week on students' math score is 5.24 with a standard error of 2.07 when matching within schools (Table 3). It is worth mentioning that the reported SE is model based and adjusts for non-independence of students within schools. From Table 3 we can see that the estimated ATT and SE for the preferential-within school approach are very similar to those obtained with the within-cluster approach. We stress that the estimated ATT should be considered carefully and only after checking the matching solution.

In conclusion, preferential within-cluster matching is expected to improve the solution of the within-cluster matching in terms of a reduced number of unmatched units. On the other hand,

	PSM		MAH†	
Statistics	**within**	**pwithin**	**within**	**pwithin**
`Orig. number of obs.`	260	260	260	260
`Orig. number of treated obs.`	128	128	128	128
`Matched number of treated obs.`	119	128	84	128
`Number of drops`	9	0	44	0
`ASAM before`	34.05	34.05	34.05	34.05
`ASAM after`	6.63	8.13	0.47	6.31
`ATT`	5.24	5.18	4.25	4.34
`SE`	2.07	2.14	2.23	1.83

† PSM: propensity score matching; MAH: Mahalanobis matching.

Table 3: Matching to allow a fair comparison of math test score of students doing (treated) and not doing homework in a school clustered dataset (NELS-88 data): comparing matching solutions obtained from CMatching.

within-cluster matching guarantees a perfect balance of school-level variables (both observed and unobserved) while preferential within does not. The researcher, choosing between the two algorithms, has to consider the trade-off between having a perfect balance of cluster level variables (within-cluster matching) or reducing the number of unmatched treated units (preferential within-cluster matching). The researcher can also implement both approaches and compare the results as a sort of sensitivity analysis.

Multivariate covariate matching

Instead of matching on the propensity score, the researcher may match directly on the covariates space. This strategy can be advantageous when the number of covariates is fairly low and it is expected to match exactly a large number of units on the original space. The syntax is very similar to the above for propensity score matching: the only difference is that the user indicates the covariate matrix instead of the propensity score in the X argument:

```
> mal_w <- CMatch(type="within", Y=Y, Tr=T, X=cbind(ses, sex, white, public),
                 Group=Group)
```

When X is a matrix, the covariate distance between a treated and a control unit is measured by Mahalanobis metrics, i.e., essentially the multivariate Euclidean distance scaled by the standard deviation of each covariate, a metrics which warrants an homogeneous imbalance reduction property for (approximately) ellipsoidally distributed covariates (Rubin, 1980). From Table 3, columns 3-4, we can see that the balance of covariates was indeed very good. Note that the estimated ATT using Mahalanobis matching is lower than the corresponding estimate obtained with propensity score matching. However, within-cluster matching using the Mahalanobis distance has generated a large number of unmatched treated units (44). Therefore, in this case preferential within-cluster matching could be used to avoid an high proportion of drops.

Other strategies

Other strategies for controlling unobserved cluster covariates in PSM have been suggested by Arpino and Mealli (2011). The basic idea is to account for the clustering in the estimation of the propensity score by using random- or fixed-effects models. This strategy can be combined with the matching strategies presented before in order to 'doubly' account for the clustering. This can be done easily with **CMatching**. As an example we consider estimating the propensity score with a logit model with dummies for schools and then matching preferentially within-schools using the estimated propensity score:

```
# estimate ps
> mod <- glm(T ~ ses + parented + public + sex + race + urban
             schid - 1, family=binomial(link="logit"), data=schools)
eps <- fitted(mod)

# match within on eps
> dpsm <- CMatch(type="pwithin", Y=math, Tr=T, X=eps, Group=NULL)
```

In concluding this section, we also mention some other matching strategies which can be implemented using **CMatching** and some programming effort. First, the utility of the algorithms naturally extends when there are more than two levels. In this case, it can be useful to match preferentially on increasingly general levels, for example by allowing individuals to be matched first between regions and then between countries. Another natural extension involves data where units have multiple membership at the same hierarchical level. In this case it is possible to combine match within (or preferential-within) across levels, for example by matching students both within schools and within living district (e.g. 1 out of 4 possible combinations).

Summary

In this paper we presented the package **CMatching** implementing matching algorithms for clustered data. The package allows users to implement two algorithms: i) within-cluster matching and ii) preferential within-cluster matching. The algorithms provide a model-based estimate of the causal effect and its standard error adjusted for within-cluster correlation among observations. In addition, a tailored summary method and a balance function are provided to analyze the output. We illustrated the case for within and preferential within-cluster matching analyzing data on students enrolled in different schools for which it is reasonable to assume important confounding at the school level. Finally, since the analysis of clustered observational data is an active area of research, we are willing to improve on standard error calculations for matching estimators with clustered data if new theoretical results in the causal inference literature will become available.

Bibliography

B. Arpino and M. Cannas. Propensity score matching with clustered data. An application to the estimation of the impact of caesarean section on the Apgar score. *Statistics in Medicine*, 35(12): 2074–2091, 2016. URL https://10.1002/sim.6880. sim.6880. [p7, 9, 10]

B. Arpino and F. Mealli. The specification of the propensity score in multilevel observational studies. *Computational Statistics & Data Analysis*, 55:1770–1780, 2011. URL https://dx.doi.org/10.1016/j.csda.2010.11.008. [p7, 19]

G. Cafri, W. Wang, P. H. Chan, and P. C. Austin. A review and empirical comparison of causal inference methods for clustered observational data with application to the evaluation of the effectiveness of medical devices. *Statistical Methods in Medical Research*, 0(0):0962280218799540, 2018. URL https://10.1177/0962280218799540. [p7, 10]

A. C. Cameron, J. B. Gelbach, and D. L. Miller. Robust inference with multiway clustering. *Journal of Business & Economic Statistics*, 29(2):238–249, 2011. URL https://10.1198/jbes.2010.07136. [p9]

A. Diamond and J. S. Sekhon. Genetic matching for estimating causal effects: A general multivariate matching method for achieving balance in observational studies. *Review of Economics and Statistics*, 95(3):932–945, 2013. URL https://doi.org/10.1162/REST_a_00318. [p8]

B. B. Hansen and S. O. Klopfer. Optimal full matching and related designs via network flows. *Journal of Computational and Graphical Statistics*, 15(3):609–627, 2006. URL https://10.1198/106186006X137047. [p8]

D. E. Ho, K. Imai, G. King, and E. A. Stuart. Matching as nonparametric preprocessing for reducing model dependence in parametric causal inference. *Political Analysis*, 15(3):199–236, 2007. URL https://10.1093/pan/mpl013. [p9]

P. W. Holland. Statistics and causal inference. *Journal of the American Statistical Association*, 81 (396):945–960, 1986. [p9]

S. M. Iacus, G. King, and G. Porro. Multivariate matching methods that are monotonic imbalance bounding. *Journal of the American Statistical Association*, 106(493):345–361, 2011. URL https://doi.org/10.1198/jasa.2011.tm09599. [p8]

G. W. Imbens and D. B. Rubin. *Causal Inference for Statistics, Social and Biomedical Sciences*. John Wiley & Sons, 2016. [p7]

I. Kraft and J. de Leeuw. *Introducing Multilevel Modeling*. London, Sage, 1998. [p14]

F. Li, A. M. Zaslavsky, and M. B. Landrum. Propensity score weighting with multilevel data. *Statistics in Medicine*, 32(19):3373–3387, 2013. ISSN 1097-0258. URL https://10.1002/sim.5786. [p7, 9]

J. H. Rickles and M. Seltzer. A two-stage propensity score matching strategy for treatment effect estimation in a multisite observational study. *Journal of Educational and Behavioral Statistics*, 39 (6):612–636, 2014. URL https://10.3102/1076998614559748. [p7, 10]

P. Rosenbaum and D. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70:41–55, 1983. URL https://10.2307/2335942. [p7]

P. Rosenbaum and D. Rubin. Constructing a control group using multivariate matched sampling methods that incorporate the propensity score. *The American Statistician*, 39:33–38, 1985. URL https://doi.org/10.1080/00031305.1985.10479383. [p9]

D. B. Rubin. Bias reduction using mahalanobis-metric matching. *Biometrics*, 36(2):293–298, 1980. URL https://10.2307/2529981. [p18]

F. Savje, J. Sekhon, and M. Higgins. *Quickmatch: Quick Generalized Full Matching*, 2018. URL https://CRAN.R-project.org/package=quickmatch. R package version 0.2.1. [p8]

J. S. Sekhon. Multivariate and propensity score matching software with automated balance optimization: The Matching package for R. *Journal of Statistical Software*, 42(7):1–52, 2011. URL https://dx.doi.org/10.18637/jss.v042.i07. [p8]

E. A. Stuart. Matching methods for causal inference: A review and a look forward. *Statist. Sci.*, 25 (1):1–21, 2010. URL https://10.1214/09-STS313. [p7, 12]

J. R. Zubizarreta, C. Kilcioglu, and J. P. Vielma. *Designmatch: Matched Samples That Are Balanced and Representative by Design*, 2018. URL https://CRAN.R-project.org/package=designmatch. R package version 0.3.1. [p8]

R. J. Zubizarreta and L. Keele. Optimal multilevel matching in clustered observational studies: A case study of the effectiveness of private schools under a large-scale voucher system. *Journal of the American Statistical Association*, 112:0, 2017. URL https://doi.org/10.1080/01621459.2016.1240683. [p7]

Massimo Cannas
University of Cagliari
Via Sant'Ignazio 87, 09123 Cagliari, Italy
(ORCiD: https://orcid.org/0000-0002-1227-5875)
massimo.cannas@unica.it

Bruno Arpino
University of Florence

Viale Morgagni, 59 50134 Firenze, Italy
RECSM-UPF
Carrer Ramon Trias Fargas 25-27, 08005 Barcelona, Spain
(ORCiD: https://orcid.org/0000-0002-8374-3066)
bruno.arpino@unifi.it

Time-Series Clustering in R Using the dtwclust Package

by Alexis Sardá-Espinosa

Abstract Most clustering strategies have not changed considerably since their initial definition. The common improvements are either related to the distance measure used to assess dissimilarity, or the function used to calculate prototypes. Time-series clustering is no exception, with the Dynamic Time Warping distance being particularly popular in that context. This distance is computationally expensive, so many related optimizations have been developed over the years. Since no single clustering algorithm can be said to perform best on all datasets, different strategies must be tested and compared, so a common infrastructure can be advantageous. In this manuscript, a general overview of shape-based time-series clustering is given, including many specifics related to Dynamic Time Warping and associated techniques. At the same time, a description of the **dtwclust** package for the R statistical software is provided, showcasing how it can be used to evaluate many different time-series clustering procedures.

Introduction

Cluster analysis is a task that concerns itself with the creation of groups of objects, where each group is called a cluster. Ideally, all members of the same cluster are similar to each other, but are as dissimilar as possible from objects in a different cluster. There is no single definition of a cluster, and the characteristics of the objects to be clustered vary. Thus, there are several algorithms to perform clustering. Each one defines specific ways of defining what a cluster is, how to measure similarities, how to find groups efficiently, etc. Additionally, each application might have different goals, so a certain clustering algorithm may be preferred depending on the type of clusters sought (Kaufman and Rousseeuw, 1990).

Clustering algorithms can be organized differently depending on how they handle the data and how the groups are created. When it comes to static data, i.e., if the values do not change with time, clustering methods can be divided into five major categories: partitioning (or partitional), hierarchical, density-based, grid-based, and model-based methods (Liao, 2005; Rani and Sikka, 2012). They may be used as the main algorithm, as an intermediate step, or as a preprocessing step (Aghabozorgi et al., 2015).

Time-series is a common type of dynamic data that naturally arises in many different scenarios, such as stock data, medical data, and machine monitoring, just to name a few (Aghabozorgi et al., 2015; Aggarwal and Reddy, 2013). They pose some challenging issues due to the large size and high dimensionality commonly associated with time-series (Aghabozorgi et al., 2015). In this context, dimensionality of a series is related to time, and it can be understood as the length of the series. Additionally, a single time-series object may be constituted by several values that change on the same time scale, in which case they are identified as multivariate time-series.

There are many techniques to modify time-series in order to reduce dimensionality, and they mostly deal with the way time-series are represented. Changing representation can be an important step, not only in time-series clustering, and it constitutes a wide research area on its own (cf. Table 2 in Aghabozorgi et al. (2015)). While choice of representation can directly affect clustering, it can be considered as a different step, and as such it will not be discussed further here.

Time-series clustering is a type of clustering algorithm made to handle dynamic data. The most important elements to consider are the (dis)similarity or distance measure, the prototype extraction function (if applicable), the clustering algorithm itself, and cluster evaluation (Aghabozorgi et al., 2015). In many cases, algorithms developed for time-series clustering take static clustering algorithms and either modify the similarity definition, or the prototype extraction function to an appropriate one, or apply a transformation to the series so that static features are obtained (Liao, 2005). Therefore, the underlying basis for the different clustering procedures remains approximately the same across clustering methods. The most common approaches are hierarchical and partitional clustering (cf. Table 4 in Aghabozorgi et al. (2015)), the latter of which includes fuzzy clustering.

Aghabozorgi et al. (2015) classify time-series clustering algorithms based on the way they treat the data and how the underlying grouping is performed. One classification depends on whether the whole series, a subsequence, or individual time points are to be clustered. On the other hand, the clustering itself may be shape-based, feature-based, or model-based. Aggarwal and Reddy (2013) make an additional distinction between online and offline approaches, where the former usually deals with grouping incoming data streams on-the-go, while the latter deals with data that no longer change.

In the context of shape-based time-series clustering, it is common to utilize the Dynamic Time Warping (DTW) distance as dissimilarity measure (Aghabozorgi et al., 2015). The calculation of the DTW distance involves a dynamic programming algorithm that tries to find the optimum warping path between two series under certain constraints. However, the DTW algorithm is computationally expensive, both in time and memory utilization. Over the years, several variations and optimizations have been developed in an attempt to accelerate or optimize the calculation. Some of the most common techniques will be discussed in more detail in Dynamic time warping distance.

The choice of time-series representation, preprocessing, and clustering algorithm has a big impact on performance with respect to cluster quality and execution time. Similarly, different programming languages have different run-time characteristics and user interfaces, and even though many authors make their algorithms publicly available, combining them is far from trivial. As such, it is desirable to have a common platform on which clustering algorithms can be tested and compared against each other. The **dtwclust** package, developed for the R statistical software, and part of its *TimeSeries* view, provides such functionality, and includes implementations of recently developed time-series clustering algorithms and optimizations. It serves as a bridge between classical clustering algorithms and time-series data, additionally providing visualization and evaluation routines that can handle time-series. All of the included algorithms are custom implementations, except for the hierarchical clustering routines. A great amount of effort went into implementing them as efficiently as possible, and the functions were designed with flexibility and extensibility in mind.

Most of the included algorithms and optimizations are tailored to the DTW distance, hence the package's name. However, the main clustering function is flexible so that one can test many different clustering approaches, using either the time-series directly, or by applying suitable transformations and then clustering in the resulting space. We will describe the new algorithms that are available in **dtwclust**, mentioning the most important characteristics of each, and showing how the package can be used to evaluate them, as well as how other packages complement it. Additionally, the variations related to DTW and other common distances will be explored.

There are many available R packages for data clustering. The **flexclust** package (Leisch, 2006) implements many partitional procedures, while the **cluster** package (Maechler et al., 2019) focuses more on hierarchical procedures and their evaluation; neither of them, however, is specifically targeted at time-series data. Packages like **TSdist** (Mori et al., 2016) and **TSclust** (Montero and Vilar, 2014) focus solely on dissimilarity measures for time-series, the latter of which includes a single algorithm for clustering based on p values. Another example is the **pdc** package (Brandmaier, 2015), which implements a specific clustering algorithm, namely one based on permutation distributions. The **dtw** package (Giorgino, 2009) implements extensive functionality with respect to DTW, but does not include the lower bound techniques that can be very useful in time-series clustering. New clustering algorithms like k-Shape (Paparrizos and Gravano, 2015) and TADPole (Begum et al., 2015) are available to the public upon request, but were implemented in MATLAB, making their combination with other R packages cumbersome. Hence, the **dtwclust** package is intended to provide a consistent and user-friendly way of interacting with classic and new clustering algorithms, taking into consideration the nuances of time-series data.

The rest of this manuscript presents the different logical units required for a time-series clustering workflow, and specifies how they are implemented in **dtwclust**. These build on top of each other and are not entirely independent, so their coherent combination is critical. The information relevant to the distance measures will be presented in Distance measures. Supported algorithms for prototype extraction will be discussed in Time-series prototypes. The main clustering algorithms will be introduced in Time-series clustering. Information regarding cluster evaluation will be provided in Cluster evaluation. The provided tools for a complete time-series clustering workflow will be described in Comparing clustering algorithms with **dtwclust**, and the final remarks will be given in Conclusion. Note that code examples are intentionally brief, and do not necessarily represent a thorough procedure to choose or evaluate a clustering algorithm. The data used in all examples is included in the package (saved in a list called `CharTraj`), and is a subset of the character trajectories dataset found in Lichman (2013): they are pen tip trajectories recorded for individual characters, and the subset contains 5 examples of the x velocity for each considered character.

Distance measures

Distance measures provide quantification for the dissimilarity between two time-series. Calculating distances, as well as cross-distance matrices, between time-series objects is one of the cornerstones of any time-series clustering algorithm. The **proxy** package (Meyer and Buchta, 2019) provides an extensible framework for these calculations, and is used extensively by **dtwclust**; Summary of distance measures will elaborate in this regard.

The l_1 and l_2 vector norms, also known as Manhattan and Euclidean distances respectively, are the most commonly used distance measures, and are arguably the only competitive l_p norms when measuring dissimilarity (Aggarwal et al., 2001; Lemire, 2009). They can be efficiently computed, but are only defined for series of equal length and are sensitive to noise, scale, and time shifts. Thus, many other distance measures tailored to time-series have been developed in order to overcome these limitations, as well as other challenges associated with the structure of time-series, such as multiple variables, serial correlation, etc.

In the following sections a description of the distance functions included in **dtwclust** will be provided; these are associated with shape-based time-series clustering, and either support DTW or provide an alternative to it. The included distances are a basis for some of the prototyping functions described in Time-series prototypes, as well as the clustering routines from Time-series clustering, but there are many other distance measures that can be used for time-series clustering and classification (Montero and Vilar, 2014; Mori et al., 2016). It is worth noting that, even though some of these distances are also available in other R packages, e.g., DTW in **dtw** or Keogh's DTW lower bound in **TSdist** (see Dynamic time warping distance), the implementations in **dtwclust** are optimized for speed, since all of them are implemented in C++ and have custom loops for computation of cross-distance matrices, including multi-threading support; refer to Practical optimizations for more information.

To facilitate notation, we define a time-series as a vector (or set of vectors in case of multivariate series) x. Each vector must have the same length for a given time-series. In general, x_i^v represents the i-th element of the v-th variable of the (possibly multivariate) time-series x. We will assume that all elements are equally spaced in time in order to avoid the time index explicitly.

Dynamic time warping distance

DTW is a dynamic programming algorithm that compares two series and tries to find the optimum warping path between them under certain constraints, such as monotonicity (Berndt and Clifford, 1994). It started being used by the data mining community to overcome some of the limitations associated with the Euclidean distance (Ratanamahatana and Keogh, 2004).

The easiest way to get an intuition of what DTW does is graphically. Figure 1 shows the alignment between two sample time-series x and y. In this instance, the initial and final points of the series must match, but other points may be warped in time in order to find better matches.

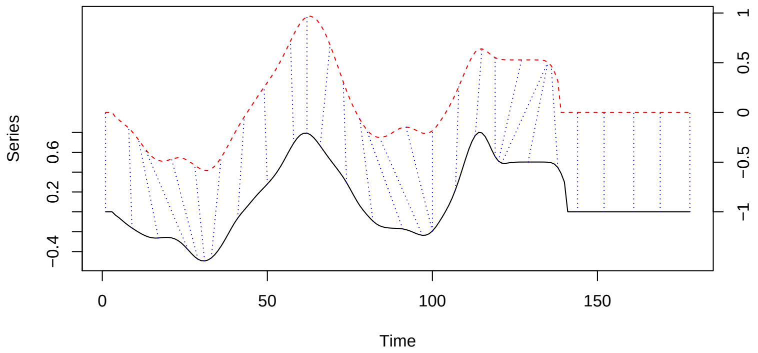


Figure 1: Sample alignment performed by the DTW algorithm between two series. The dashed blue lines exemplify how some points are mapped to each other, which shows how they can be warped in time. Note that the vertical position of each series was artificially altered for visualization.

DTW is computationally expensive. If x has length n and y has length m, the DTW distance between them can be computed in $O(nm)$ time, which is quadratic if m and n are similar. Additionally, DTW is prone to implementation bias since its calculations are not easily vectorized and tend to be very slow in non-compiled programming languages. A custom implementation of the DTW algorithm is included with **dtwclust** in the `dtw_basic` function, which has only basic functionality but still supports the most common options, and it is faster (see Practical optimizations).

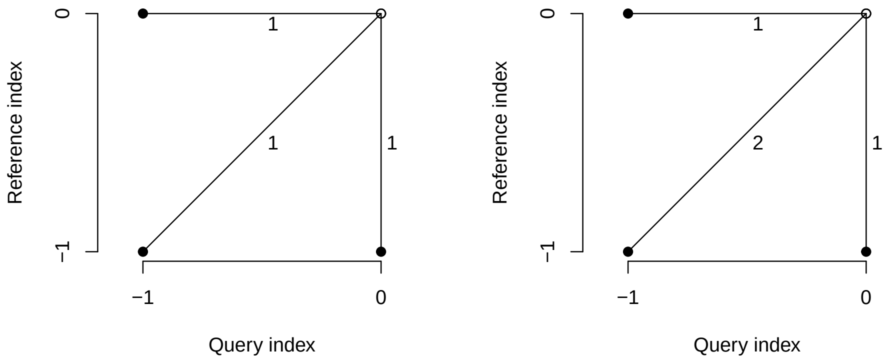
The DTW distance can potentially deal with series of different length directly. This is not necessarily an advantage, as it has been shown before that performing linear reinterpolation to obtain equal length may be appropriate if m and n do not vary significantly (Ratanamahatana and Keogh, 2004). For a more detailed explanation of the DTW algorithm see, e.g., Giorgino (2009). However, there are some aspects that are worth discussing here.

The first step in DTW involves creating a local cost matrix (LCM or lcm), which has $n \times m$ dimensions. Such a matrix must be created for every pair of distances compared, meaning that memory requirements may grow quickly as the dataset size grows. Considering x and y as the input series, for each element (i, j) of the LCM, the l_p norm between x_i and y_j must be computed. This is defined in Equation 1, explicitly denoting that multivariate series are supported as long as they have the same number of variables (note that for univariate series, the LCM will be identical regardless of the used norm). Thus, it makes sense to speak of a DTW$_p$ distance, where p corresponds to the l_p norm that was used to construct the LCM.

$$lcm(i,j) = \left(\sum_v |x_i^v - y_j^v|^p \right)^{1/p} \tag{1}$$

In the seconds step, the DTW algorithm finds the path that minimizes the alignment between x and y by iteratively stepping through the LCM, starting at $lcm(1,1)$ and finishing at $lcm(n,m)$, and aggregating the cost. At each step, the algorithm finds the direction in which the cost increases the least under the chosen constraints.

The way in which the algorithm traverses through the LCM is primarily dictated by the chosen step pattern. It is a local constraint that determines which directions are allowed when moving ahead in the LCM as the cost is being aggregated, as well as the associated per-step weights. Figure 2 depicts two common step patterns and their names in the **dtw** package. Unfortunately, very few articles from the data mining community specify which pattern they use, although in the author's experience, the `symmetric1` pattern seems to be standard. By contrast, the `dtw` and `dtw_basic` functions use the `symmetric2` pattern by default, but it is simple to modify this by providing the appropriate value in the `step.pattern` argument. The choice of step pattern also determines whether the corresponding DTW distance can be normalized or not (which may be important for series with different length). See Giorgino (2009) for a complete list of step patterns and to know which ones can be normalized.



(a) `symmetric1` step pattern **(b)** `symmetric2` step pattern

Figure 2: Two common step patterns used by DTW when traversing the LCM. At each step, the lines denote the allowed directions that can be taken, as well as the weight associated with each one.

It should be noted that the DTW distance does not satisfy the triangle inequality, and it is not symmetric in general, e.g., for asymmetric step patterns (Giorgino, 2009). The patterns in Figure 2 can result in a symmetric DTW calculation, provided no constraints are used (see the next section), or all series have the same length if a constraint is indeed used.

Global DTW constraints

One of the possible modifications of DTW is to use global constraints, also known as window constraints. These limit the area of the LCM that can be reached by the algorithm. There are many types of windows (see, e.g., Giorgino (2009)), but one of the most common ones is the Sakoe-Chiba window (Sakoe and Chiba, 1978), with which an allowed region is created along the diagonal of the LCM (see Figure 3). These constraints can marginally speed up the DTW calculation, but they are mainly used to avoid pathological warping. It is common to use a window whose size is 10% of the series' length, although sometimes smaller windows produce even better results (Ratanamahatana and Keogh, 2004).

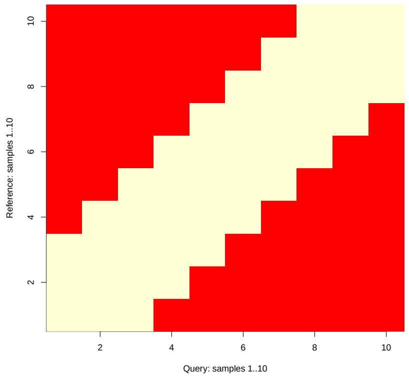


Figure 3: Visual representation of the Sakoe-Chiba constraint for DTW. The red elements will not be considered by the algorithm when traversing the LCM.

Strictly speaking, if the series being compared have different lengths, a constrained path may not exist, since the Sakoe-Chiba band may prevent the end point of the LCM to be reached (Giorgino, 2009). In these cases a slanted band window may be preferred, since it stays along the diagonal for series of different length and is equivalent to the Sakoe-Chiba window for series of equal length. If a window constraint is used with **dtwclust**, a slanted band is employed.

It is not possible to know a priori what window size, if any, will be best for a specific application, although it is usually agreed that no constraint is a poor choice. For this reason, it is better to perform tests with the data one wants to work with, perhaps taking a subset to avoid excessive running times.

It should be noted that, when reported, window sizes are always integers greater than zero. If we denote this number with w, and for the specific case of the slanted band window, the valid region of the LCM will be constituted by all valid points in the range $[(i, j - w), (i, j + w)]$ for all (i, j) along the LCM diagonal. Thus, at each step, at most $2w + 1$ elements may fall within the window for a given window size w. This is the convention followed by **dtwclust**.

Lower bounds for DTW

Due to the fact that DTW itself is expensive to compute, lower bounds (LBs) for the DTW distance have been developed. These lower bounds guarantee being less than or equal to the corresponding DTW distance. They have been exploited when indexing time-series databases, classification of time-series, clustering, etc. (Keogh and Ratanamahatana, 2005; Begum et al., 2015). Out of the existing DTW LBs, the two most effective are termed `LB_Keogh` (Keogh and Ratanamahatana, 2005) and `LB_Improved` (Lemire, 2009). The reader is referred to the respective articles for detailed definitions and proofs of the LBs, however some important considerations will be further discussed here.

Each LB can be computed with a specific l_p norm. Therefore, it follows that the l_p norms used for DTW and LB calculations must match, such that $\text{LB}_p \leqslant \text{DTW}_p$. Moreover, $\text{LB\_Keogh}_p \leqslant \text{LB\_Improved}_p \leqslant \text{DTW}_p$, meaning that `LB_Improved` can provide a tighter LB. It must be noted that the LBs are only defined for series of equal length and are not symmetric regardless of the l_p norm used to compute them. Also note that the choice of step pattern affects the value of the DTW distance, changing the tightness of a given LB.

One crucial step when calculating the LBs is the computation of the so-called envelopes. These envelopes require a window constraint, and are thus dependent on both the type and size of the window. Based on these, a running minimum and maximum are computed, and a lower and upper envelope are generated respectively. Figure 4 depicts a sample time-series with its corresponding envelopes for a Sakoe-Chiba window of size 15.

In order for the LBs to be worth it, they must be computed in significantly less time than it takes

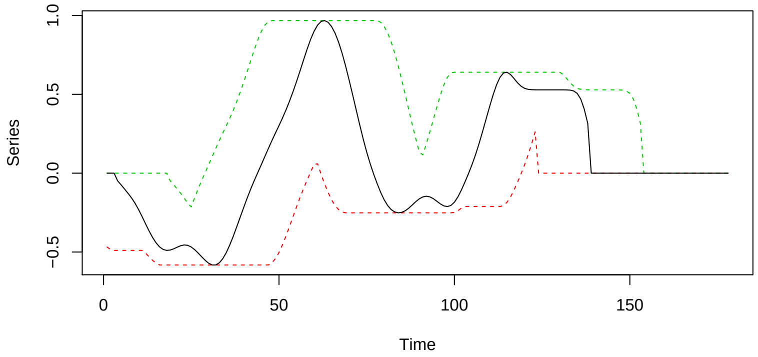


Figure 4: Visual representation of a time-series (shown as a solid black line) and its corresponding envelopes based on a Sakoe-Chiba window of size 15. The green dashed line represents the upper envelope, while the red dashed line represents the lower envelope.

to calculate the DTW distance. Lemire (2009) developed a streaming algorithm to calculate the envelopes using no more than $3n$ comparisons when using a Sakoe-Chiba window. This algorithm has been ported to **dtwclust** using the C++ programming language, ensuring an efficient calculation, and it is exposed in the `compute_envelope` function.

`LB_Keogh` requires the calculation of one set of envelopes for every pair of series compared, whereas `LB_Improved` must calculate two sets of envelopes for every pair of series. If the LBs must be calculated between several time-series, some envelopes can be reused when a given series is compared against many others. This optimization is included in the LB functions registered with **proxy** by **dtwclust**.

Global alignment kernel distance

Cuturi (2011) proposed an algorithm to assess similarity between time series by using kernels. He began by formalizing an alignment between two series x and y as π, and defined the set of all possible alignments as $\mathcal{A}(n, m)$, which is constrained by the lengths of x and y. It is shown that the DTW distance can be understood as the cost associated with the minimum alignment.

A Global Alignment (GA) kernel that considers the cost of all possible alignments by computing their exponentiated soft-minimum is defined, and it is argued that it quantifies similarities in a more coherent way. However, the GA kernel has associated limitations, namely diagonal dominance and a complexity $O(nm)$. With respect to the former, Cuturi (2011) states that diagonal dominance should not be an issue as long as one of the series being compared is not longer than twice the length of the other.

In order to reduce the GA kernel's complexity, Cuturi (2011) proposed using the triangular local kernel for integers shown in Equation 2, where T represents the kernel's order. By combining it with the kernel κ in Equation 3 (which is based on the Gaussian kernel κ_σ), the Triangular Global Alignment Kernel (TGAK) in Equation 4 is obtained. Such a kernel can be computed in $O(T \min(n, m))$, and is parameterized by the triangular constraint T and the Gaussian's kernel width σ.

$$\omega(i, j) = \left(1 - \frac{|i - j|}{T}\right)_+ \tag{2}$$

$$\kappa(x, y) = e^{-\phi_\sigma(x,y)} \tag{3a}$$

$$\phi_\sigma(x, y) = \frac{1}{2\sigma^2} \left\| x - y \right\|^2 + \log\left(2 - e^{-\frac{\|x-y\|^2}{2\sigma^2}}\right) \tag{3b}$$

$$\text{TGAK}(x, y, \sigma, T) = \tau^{-1} \left(\omega \otimes \frac{1}{2}\kappa\right)(i, x; j, y) = \frac{\omega(i, j)\kappa(x, y)}{2 - \omega(i, j)\kappa(x, y)} \tag{4}$$

The triangular constraint is similar to the window constraints that can be used in the DTW algorithm. When $T = 0$ or $T \to \infty$, the TGAK converges to the original GA kernel. When $T = 1$, the TGAK becomes a slightly modified Gaussian kernel that can only compare series of equal length. If $T > 1$, then only the alignments that fulfil $-T < \pi_1(i) - \pi_2(i) < T$ are considered.

Cuturi (2011) also proposed a strategy to estimate the value of σ based on the time-series themselves and their lengths, namely $c \cdot \text{med}(\|x - y\|) \cdot \sqrt{\text{med}(|\text{x}|)}$, where $\text{med}(\cdot)$ is the empirical median, c is some constant, and x and y are subsampled vectors from the dataset. This, however, introduces some randomness into the algorithm when the value of σ is not provided, so it might be better to estimate it once and re-use it in subsequent function evaluations. In **dtwclust**, the value of c is set to 1.

The similarity returned by the TGAK can be normalized with Equation 5 so that its values lie in the range $[0, 1]$. Hence, a distance measure for time-series can be obtained by subtracting the normalized value from 1. The algorithm supports multivariate series and series of different length (with some limitations). The resulting distance is symmetric and satisfies the triangle inequality, although it is more expensive to compute in comparison to DTW.

$$\exp\left(\log\left(\text{TGAK}(x, y, \sigma, T)\right) - \frac{\log\left(\text{TGAK}(x, x, \sigma, T)\right) + \log\left(\text{TGAK}(y, y, \sigma, T)\right)}{2}\right) \tag{5}$$

A C implementation of the TGAK algorithm is available at its author's website[1]. An R wrapper has been implemented in **dtwclust** in the `GAK` function, performing the aforementioned normalization and subtraction in order to obtain a distance measure that can be used in clustering procedures.

Soft-DTW

Following with the idea of the TGAK, i.e., of regularizing DTW by smoothing it, Cuturi and Blondel (2017) proposed a unified algorithm using a parameterized soft-minimum as shown in Equation 6 (where $\Delta(x, y)$ represents the LCM), and called the resulting discrepancy a soft-DTW, discussing its differentiability. Thanks to this property, a gradient function can be obtained, and Cuturi and Blondel (2017) developed a more efficient way to compute it. This can be then used to calculate centroids with numerical optimization as discussed in Soft-DTW centroid.

$$\text{dtw}_\gamma(x, y) = \min{}^\gamma\{\langle A, \Delta(x, y)\rangle, A \in \mathcal{A}(n, m)\} \tag{6a}$$

$$\min{}^\gamma\{a_1, \dots, a_n\} = \begin{cases} \min_{i \leqslant n} a_i, & \gamma = 0 \\ -\gamma \log \sum_{i=1}^{n} e^{-a_i/\gamma}, & \gamma > 0 \end{cases} \tag{6b}$$

However, as a stand-alone distance measure, the soft-DTW distance has some disadvantages: the distance can be negative, the distance between x and itself is *not* necessarily zero, it does not fulfill the triangle inequality, and also has quadratic complexity with respect to the series' lengths. On the other hand, it is a symmetric distance, it supports multivariate series as well as different lengths, and it can provide differently smoothed results by means of a user-defined γ.

Shape-based distance

The shape-based distance (SBD) was proposed as part of the k-Shape clustering algorithm (Paparrizos and Gravano, 2015); this algorithm will be further discussed in Shape extraction and k-Shape clustering. SBD is presented as a faster alternative to DTW. It is based on the cross-correlation with coefficient normalization (NCCc) sequence between two series, and is thus sensitive to scale, which is why Paparrizos and Gravano (2015) recommend z-normalization. The NCCc sequence is obtained by convolving the two series, so different alignments can be considered, but no point-wise warpings are made. The distance can be calculated with the formula shown in Equation 7, where $\|\cdot\|_2$ is the l_2 norm of the series. Its range lies between 0 and 2, with 0 indicating perfect similarity.

$$SBD(x, y) = 1 - \frac{\max\left(NCCc(x, y)\right)}{\|x\|_2 \, \|y\|_2} \tag{7}$$

This distance can be efficiently computed by utilizing the Fast Fourier Transform (FFT) to obtain the NCCc sequence, although that might make it sensitive to numerical precision, especially in 32-bit architectures. It can be very fast, it is symmetric, it was very competitive in the experiments performed in Paparrizos and Gravano (2015) (although the run-time comparison was slightly biased due to the slow MATLAB implementation of DTW), and it supports (univariate) series of different

[1] `http://marcocuturi.net/GA.html`, accessed on 2016-10-29

length directly. Additionally, some FFTs can be reused when computing the SBD between several series; this optimization is also included in the SBD function registered with **proxy** by **dtwclust**.

Summary of distance measures

The distances described in this section are the ones implemented in **dtwclust**, which serve as basis for the algorithms presented in Time-series prototypes and Time-series clustering. Table 1 summarizes the salient characteristics of these distances.

Distance	Computational cost	Normalized	Symmetric	Multivariate support	Support for length differences
LB_Keogh	Low	No	No	No	No
LB_Improved	Low	No	No	No	No
DTW	Medium	Can be*	Can be*	Yes	Yes
GAK	High	Yes	Yes	Yes	Yes
Soft-DTW	High	Yes	Yes	Yes	Yes
SBD	Low	Yes	Yes	No	Yes

Table 1: Characteristics of time-series distance measures implemented in **dtwclust**. Regarding the cells marked with an asterisk: the DTW distance can be normalized for certain step patterns, and can be symmetric for symmetric step patterns when either no window constraints are used, or all time-series have the same length if constraints are indeed used.

Nevertheless, there are many other measures that can be used. In order to account for this, the **proxy** package is leveraged by **dtwclust**, as well as other packages (e.g., **TSdist**). It aggregates all its measures in a database object called `pr_DB`, which has the advantage that all registered functions can be used with the `proxy::dist` function. For example, registering the autocorrelation-based distance provided by package **TSclust** could be done in the following way.

```
require("TSclust")

proxy::pr_DB$set_entry(FUN = diss.ACF, names = c("ACFD"),
                       loop = TRUE, distance = TRUE,
                       description = "Autocorrelation-based distance")

proxy::dist(CharTraj[3L:8L], method = "ACFD", upper = TRUE)

A.V3      A.V4      A.V5      B.V1      B.V2      B.V3
A.V3           0.7347970 0.7269654 1.3365966 0.9022004 0.6204877
A.V4 0.7347970           0.2516642 2.0014314 1.5712718 1.2133404
A.V5 0.7269654 0.2516642           2.0178486 1.6136650 1.2901999
B.V1 1.3365966 2.0014314 2.0178486           0.5559639 0.9937621
B.V2 0.9022004 1.5712718 1.6136650 0.5559639           0.4530352
B.V3 0.6204877 1.2133404 1.2901999 0.9937621 0.4530352
```

Any distance function registered with **proxy** can be used for time-series clustering with **dtwclust**. More details are provided in Clustering examples.

Practical optimizations

As mentioned before, one of the advantages of the distances implemented as part of **dtwclust** is that the core calculations are performed in C++, making them faster. The other advantage is that the calculations of cross-distance matrices leverage multi-threading. In the following, a series of comparisons against implementations in other packages is presented, albeit without the consideration of parallelization. Further information is available in the vignettes included with the package[2].

[2] Visible at `https://cran.r-project.org/web/packages/dtwclust/vignettes/`

One of DTW's lower bounds, `LB_Keogh`, is also available in **TSdist** as a pure R implementation. We can see how it compares to the C++ version included in **dtwclust** in Figure 5, considering different series lengths and window sizes. The time for each point in the graph was computed by repeating the calculation 100 times and extracting the median time.

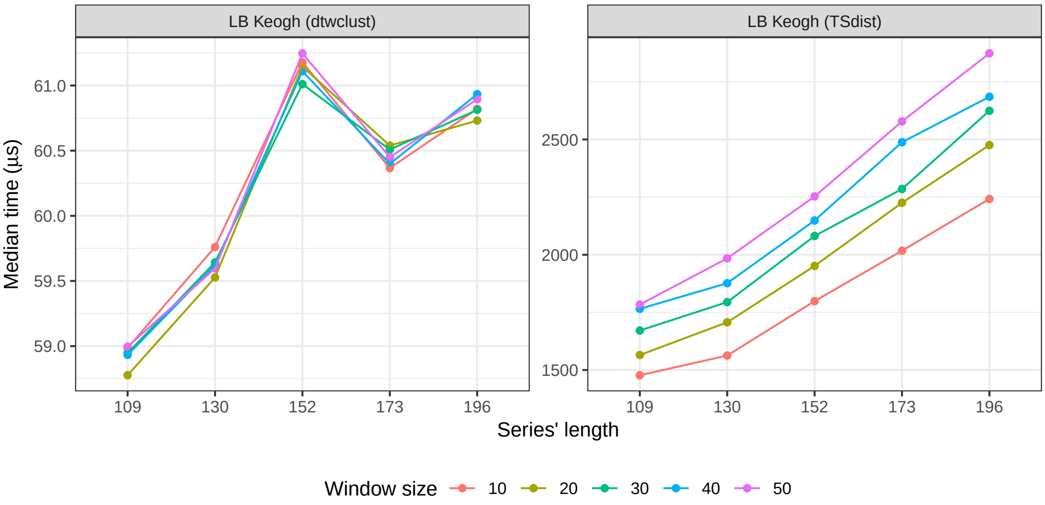


Figure 5: Execution times of two implementations of `LB_Keogh` considering different time series lengths and window sizes. Note the different vertical scales, although both are in microseconds. The package of each implementation is written between parentheses.

Similarly, the DTW distance is also available in the **dtw** package, and possesses a C core. The `dtw_basic` version included with **dtwclust** only supports a slanted window constraint (or none at all), and the `symmetric1` and `symmetric2` step patterns, so it performs less checks, and uses a memory-saving version where only 2 rows of the LCM are saved at a time. As with `LB_Keogh`, a comparison of the DTW implementations' execution times can be seen in Figure 6.

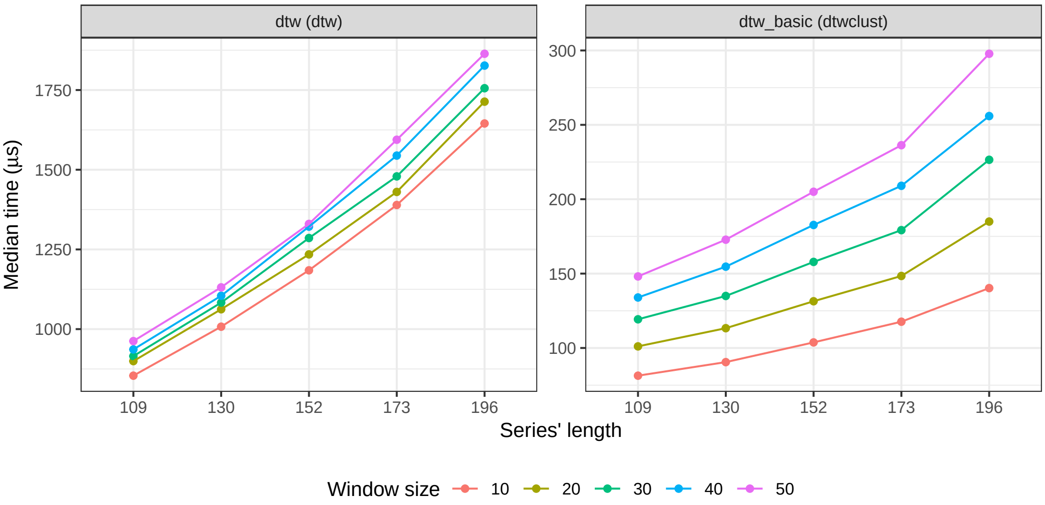


Figure 6: Execution times of two implementations of DTW considering different time series lengths and window sizes. Note the different vertical scales, although both are in microseconds. The package of each implementation is written between parentheses.

The time difference in single calculations is not so dramatic, but said differences accumulate when calculating cross-distance matrices, and become much more significant. The behavior of `LB_Keogh` can be seen in Figure 7, with a fixed window size of 30 and series of length 100. The implementation in **dtwclust** performs the whole calculation in C++, and only calculates the necessary warping envelopes once, although it can be appreciated that this does not have a significant effect.

The behavior of the DTW implementations can be seen in Figure 8. The **dtwclust** version is an order of magnitude faster, even single-threaded, and it can benefit from parallelization essentially

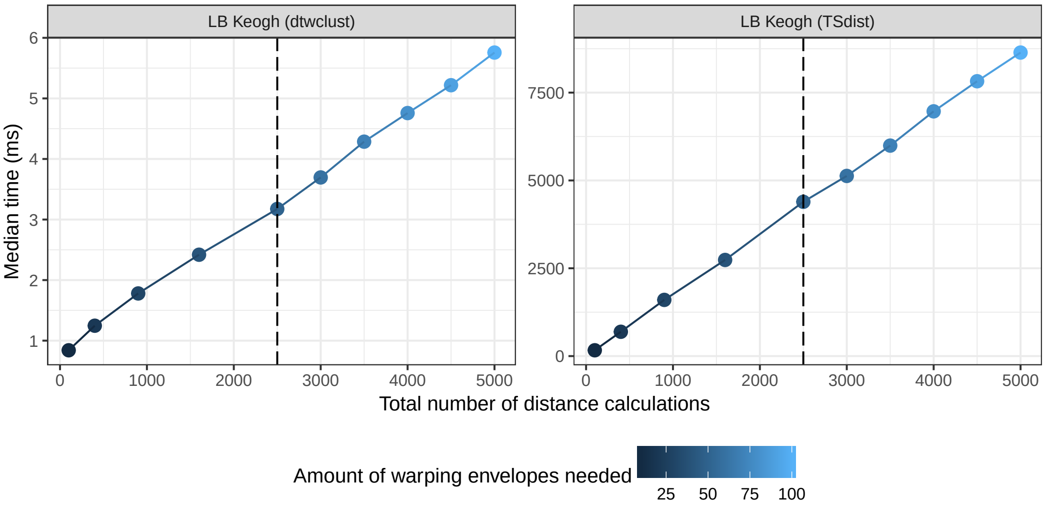


Figure 7: Execution times of the two implementations of `LB_Keogh` when calculating cross-distance matrices. The points on the left of the dashed line represent square matrices, whereas the ones on the right only have one dimension of the cross-distance matrix increased (the one that results in more envelope calculations). Note the different vertical scales, although both are in milliseconds. The package of each implementation is written between parentheses.

proportionally to the number of threads available.

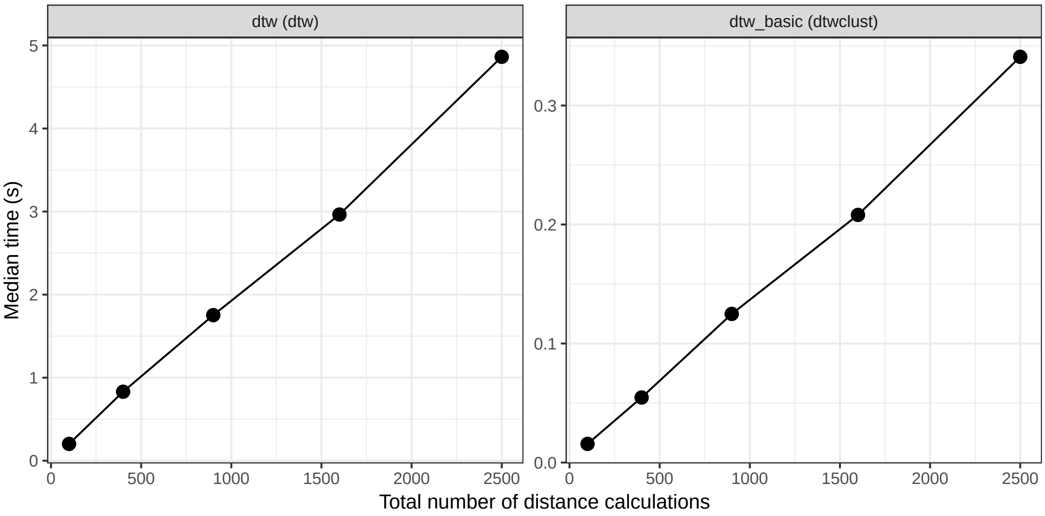


Figure 8: Execution times of the two implementations of DTW when calculating cross-distance matrices. Note the different vertical scales, although both are in seconds. The package of each implementation is written between parentheses.

Time-series prototypes

A very important step of time-series clustering is the calculation of so-called time-series prototypes. It is expected that all series within a cluster are similar to each other, and one may be interested in trying to define a time-series that effectively summarizes the most important characteristics of all series in a given cluster. This series is sometimes referred to as an average series, and prototyping is also sometimes called time-series averaging, but we will prefer the term “prototyping”, although calling them time-series centroids is also common.

Computing prototypes is commonly done as a sub-routine of a larger task. In the context of clustering (see Time-series clustering), partitional procedures rely heavily on the prototyping function, since the resulting prototypes are used as cluster centroids. Prototyping could even be a pre-processing step, whereby different samples from the same source can be summarized before clustering (e.g., for the character trajectories dataset, all trajectories from the same character can be summarized and then groups of similar characters could be sought), thus reducing the amount of data and execution time. Another example is time-series classification based on nearest-neighbors, which can be optimized by considering only group-prototypes as neighbors instead of the union of all groups. Nevertheless, it is important to note that the distance used in the overall task should be congruent with the chosen centroid, e.g., using the DTW distance for DTW-based prototypes.

The choice of prototyping function is closely related to the chosen distance measure and, in a similar fashion, it is not simple to know which kind of prototype will be better a priori. There are several strategies available for time-series prototyping, although due to their high dimensionality, what exactly constitutes an average time-series is debatable, and some notions could worsen performance significantly. The following sections will briefly describe some of the common approaches when dealing with time-series.

Partition around medoids

One approach is to use partition around medoids (PAM). A medoid is simply a representative object from a cluster, in this case also a time-series, whose average distance to all other objects in the same cluster is minimal. Since the medoid object is always an element of the original data, PAM is sometimes preferred over mean or median so that the time-series structure is not altered.

A possible advantage of PAM is that, since the data does not change, it is possible to precompute the whole distance matrix once and re-use it on each iteration, and even across different number of clusters and random repetitions. However, this is not suitable for large datasets since the whole distance matrix has to be allocated at once.

In the implementation included in the package, the distances between all member series are computed, and the series with minimum sum of distances is chosen as the prototype.

DTW barycenter averaging

The DTW distance is used very often when working with time-series, and thus a prototyping function based on DTW has also been developed in Petitjean et al. (2011). The procedure is called DTW barycenter averaging (DBA), and is an iterative, global method. The latter means that the order in which the series enter the prototyping function does not affect the outcome.

DBA requires a series to be used as reference (centroid), and it usually begins by randomly selecting one of the series in the data. On each iteration, the DTW alignment between each series in the cluster C and the centroid is computed. Because of the warping performed in DTW, it can be that several time-points from a given time-series map to a single time-point in the centroid series, so for each time-point in the centroid, all the corresponding values from all series in C are grouped together according to the DTW alignments, and the mean is computed for each centroid point using the values contained in each group. This is iteratively repeated until a certain number of iterations are reached, or until convergence is assumed.

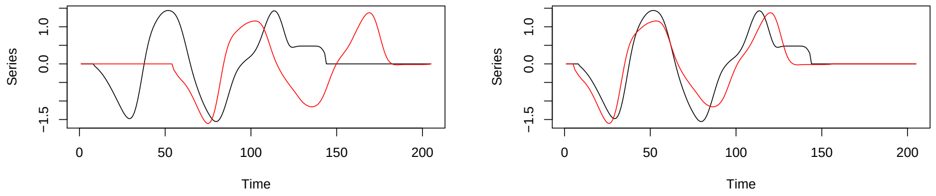
The **dtwclust** implementation of DBA is done in C++ and includes several memory optimizations. Nevertheless, it is more computationally expensive due to all the DTW calculations that must be performed. However, it is very competitive when using the DTW distance and, thanks to DTW itself, it can support series with different length directly, with the caveat that the length of the resulting prototype will be the same as the length of the reference series that was initially chosen by the algorithm, and that the `symmetric1` or `symmetric2` step pattern should be used.

Soft-DTW centroid

Thanks to the gradient that can be computed as a by-product of the soft-DTW distance calculation (see Soft-DTW), it is possible to define an objective function (see Equation (4) in Cuturi and Blondel (2017)) and subsequently minimize it with numerical optimization. In addition to the smoothing parameter of soft-DTW (γ), the optimization procedure considers the option of using normalizing weights for the input series, which noticeably alters the resulting centroids (see Figure 4 in Cuturi and Blondel (2017)). The clustering and classification experiments performed by Cuturi and Blondel (2017) showed that using soft-DTW (distance and centroid) provided quantitatively better results in many scenarios.

Shape extraction

A recently proposed method to calculate time-series prototypes is termed shape extraction, and is part of the k-Shape algorithm (see k-Shape clustering) described in Paparrizos and Gravano (2015). As with the corresponding SBD (see Shape-based distance), the algorithm depends on NCCc, and it first uses it to match two series optimally. Figure 9 depicts the alignment that is performed using two sample series.



(a) Series before alignment **(b)** Series after alignment

Figure 9: Visualization of the NCCc-based alignment performed on two sample series. After alignment, the second (red) series is either truncated and/or prepended/appended with zeros so that its length matches the first(black) series.

As with DBA, a centroid series is needed, so one is usually randomly chosen from the data. An exception is when all considered time-series have the same length, in which case no centroid is needed beforehand. The alignment can be done between series with different length, and since one of the series is shifted in time, it may be necessary to truncate and prepend or append zeros to the non-reference series, so that the final length matches that of the reference. This is because the final step of the algorithm builds a matrix with the matched series (row-wise) and performs a so-called maximization of Rayleigh Quotient to obtain the final prototype; see Paparrizos and Gravano (2015) for more details.

The output series of the algorithm must be z-normalized. Thus, the input series as well as the reference series must also have this normalization. Even though the alignment can be done between series with different length, it has the same caveat as DBA, namely that the length of the resulting prototype will depend on the length of the chosen reference. Technically, for multivariate series, the shape extraction algorithm could be applied for each variable v of all involved series, but this was not explored by the authors of k-Shape.

Summary of prototyping functions

Table 2 summarizes the time-series prototyping functions implemented in **dtwclust**, including the distance measure they are based upon, where applicable. It is worth mentioning that, as will be described in Time-series clustering, the choice of distance and prototyping function is very important for time-series clustering, and it may be ill-advised to use a distance measure that does not correspond to the one used by the prototyping function. Using PAM is an exception, because the medoids are not modified, so any distance can be used to choose a medoid. It is possible to use custom prototyping functions for time-series clustering (see Clustering examples), but it is important to maintain congruence with the chosen distance measure.

Prototyping function	Distance used	Algorithm used
PAM	—	Time-series with minimum sum of distances to the other series in the group.
DBA	DTW	Average of points grouped according to DTW alignments.
Soft-DTW centroid	Soft-DTW	Numerical optimization using the derivative of soft-DTW.
Shape extraction	SBD	Normalized eigenvector of a matrix created with SBD-aligned series.

Table 2: Time-series prototyping functions implemented in **dtwclust**, and their corresponding distance measures.

Time-series clustering

There are several clustering algorithms, but in general, they fall within 3 categories: hierarchical clustering, which induces a hierarchy in the data; partitional clustering, which creates crisp partitions of data; and fuzzy clustering, which creates fuzzy or overlapping partitions.

Hierarchical clustering is an algorithm that tries to create a hierarchy of groups in which, as the level in the hierarchy increases, clusters are created by merging the clusters from the next lower level, such that an ordered sequence of groupings is obtained; this may be deceptive, as the algorithms impose the hierarchical structure even if such structure is not inherent to the data (Hastie et al., 2009). In order to decide how the merging is performed, a (dis)similarity measure between groups should be specified, in addition to the one that is used to calculate pairwise similarities. However, a specific number of clusters does not need to be specified for the hierarchy to be created, and the procedure is deterministic, so it will always give the same result for a chosen set of (dis)similarity measures.

Hierarchical clustering has the disadvantage that the whole distance matrix must be calculated for a given dataset, which in most cases has a time and memory complexity of $O(N^2)$ if N is the total number of objects in the dataset. Thus, hierarchical procedures are usually used with relatively small datasets.

Partitional clustering is a strategy used to create partitions. In this case, the data is explicitly assigned to one and only one cluster out of k total clusters. The number of desired clusters must be specified beforehand, which can be a limiting factor. Some of the most popular partitional algorithms are k-means and k-medoids (Hastie et al., 2009). These use the Euclidean distance and, respectively, mean or PAM centroids (see Time-series prototypes).

Partitional clustering algorithms commonly work in the following way. First, k centroids are randomly initialized, usually by choosing k objects from the dataset at random; these are assigned to individual clusters. The distance between all objects in the data and all centroids is calculated, and each object is assigned to the cluster of its closest centroid. A prototyping function is applied to each cluster to update the corresponding centroid. Then, distances and centroids are updated iteratively until a certain number of iterations have elapsed, or no object changes clusters any more. Most of the proposed algorithms for time-series clustering use the same basic strategy while changing the distance and/or centroid function.

Partitional clustering procedures are stochastic due to their random start. Thus, it is common practice to test different starting points to evaluate several local optima and choose the best result out of all the repetitions. It tends to produce spherical clusters, but has a lower complexity, so it may be applied to very large datasets.

In crisp partitions, each member of the data belongs to only one cluster, and clusters are mutually exclusive. By contrast, fuzzy clustering creates a fuzzy or soft partition in which each member belongs to each cluster to a certain degree. For each member of the data, the degree of belongingness is constrained so that its sum equals 1 across all clusters. Therefore, if there are N objects in the data and k clusters are desired, an $N \times k$ membership matrix u can be created, where all the rows must sum to 1 (note that some authors use the transposed version of u).

Technically, fuzzy clustering can be repeated several times with different random starts, since u is initialized randomly. However, comparing the results would be difficult, since it could be that the values within u are shuffled but the overall fuzzy grouping remains the same, or changes very slightly, once the algorithm has converged. Note that it is straightforward to change the fuzzy partition to a crisp one by taking the argument of the row-wise maxima of u and assigning the respective series to the corresponding cluster only.

The main clustering function in **dtwclust** is `tsclust`, which supports all of the aforementioned clustering algorithms. Part of this support comes from functionality provided by other R packages. However, the advantage of using **dtwclust** is that it can handle time-series nuances, like series with different lengths and multivariate series. This is particularly important for partitional clustering, where both distance and prototyping functions must be applicable to time-series data. For brevity, the following sections will focus on describing the new clustering algorithms implemented in **dtwclust**, but more information can be obtained in the functions' documentation.

TADPole clustering

TADPole clustering was proposed in Begum et al. (2015), and is implemented in **dtwclust** in the `TADPole` function. It adopts a relatively new clustering framework and adapts it to time-series clustering with the DTW distance. Because of the way the algorithm works, it can be considered a kind of PAM clustering, since the centroids are always elements of the data. However, this algorithm is deterministic depending on the value of a cutoff distance (d_c).

The algorithm first uses the DTW distance's upper and lower bounds (Euclidean distance and `LB_Keogh` respectively) to find series with many close neighbors (in DTW space). Anything below d_c is considered a neighbor. Aided with this information, the algorithm then tries to prune as many DTW calculations as possible in order to accelerate the clustering procedure. The series that lie in dense areas (i.e., that have lots of neighbors) are taken as cluster centroids. For a more detailed explanation of each step, please refer to Begum et al. (2015).

TADPole relies on the DTW bounds, which are only defined for time-series of equal length. Consequently, it requires a Sakoe-Chiba constraint. Furthermore, it should be noted that the Euclidean distance is only valid as a DTW upper bound if the `symmetric1` step pattern is used (see Figure 2). Finally, the allocation of several distance matrices is required, making it similar to hierarchical procedures memory-wise, so its applicability is limited to relatively small datasets.

k-Shape clustering

The k-Shape clustering algorithm was developed by Paparrizos and Gravano (2015). It is a partitional clustering algorithm with a custom distance measure (SBD; see Shape-based distance), as well as a custom centroid function (shape extraction; see Shape extraction). It is also stochastic in nature, and requires z-normalization in its default definition. In order to use this clustering algorithm, the `tsclust` function should be called with partitional as the clustering type, SBD as the distance measure, shape extraction as the centroid function, and z-normalization as the preprocessing step. As can be appreciated, this algorithm uses the same strategy as k-means, but replaces both distance and prototyping functions with custom ones that are congruent with each other.

Clustering examples

In this example, three different partitional clustering strategies are used: the DTW$_2$ distance and DBA centroids, k-Shape, and finally TADPole. The results are evaluated using Variation of Information (see Cluster evaluation), with lower numbers indicating better results. Note that z-normalization is applied by default when selecting shape extraction as the centroid function. For consistency, all algorithms used the reinterpolated and normalized data, since some algorithms require series of equal length. A subset of the data is used for speed. The outcome should not be generalized to other data, and normalization/reinterpolation may actually worsen some of the algorithms' performance.

```
# Linear reinterpolation to same length
data <- reinterpolate(CharTraj, new.length = max(lengths(CharTraj)))
# z-normalization
data <- zscore(data[60L:100L])

pc_dtw <- tsclust(data, k = 4L, seed = 8L,
                  distance = "dtw_basic", centroid = "dba",
                  norm = "L2", window.size = 20L)

pc_ks <- tsclust(data, k = 4L, seed = 8L,
                 distance = "sbd", centroid = "shape")

pc_tp <- tsclust(data, k = 4L, type = "tadpole", seed = 8L,
                 control = tadpole_control(dc = 1.5, window.size = 20L))

sapply(list(DTW = pc_dtw, kShape = pc_ks, TADPole = pc_tp),
       cvi, b = CharTrajLabels[60L:100L], type = "VI")

   DTW.VI  kShape.VI TADPole.VI
0.5017081  0.4353306  0.4901096
```

As can be seen, using a distance registered with **proxy** can be done by simply specifying its name in the `distance` argument of `tsclust`. Using the prototyping functions included in **dtwclust** can be done by passing their respective names in the `centroid` parameter, but using a custom prototyping function is also possible. For example, a weighted mean centroid is implemented as follows. The usefulness of such an approach is of course questionable.

```
weighted_mean_cent <- function(x, cl_id, k, cent, cl_old, ..., weights) {
    x <- Map(x, weights, f = function(ts, w) { w * ts })
    x_split <- split(x, cl_id)
    new_cent <- lapply(x_split, function(xx) {
        xx <- do.call(rbind, xx)
        colMeans(xx)
    })
}

data <- reinterpolate(CharTraj, new.length = max(lengths(CharTraj)))
weights <- rep(c(0.9,1.1), each = 5L)
tsclust(data[1L:10L], type = "p", k = 2L,
        distance = "Manhattan",
        centroid = weighted_mean_cent,
        seed = 123,
        args = tsclust_args(cent = list(weights = weights)))

partitional clustering with 2 clusters
Using manhattan distance
Using weighted_mean_cent centroids

Time required for analysis:
user  system elapsed
0.024   0.000   0.023

Cluster sizes with average intra-cluster distance:

size  av_dist
1    5 15.05069
2    5 18.99145
```

Cluster evaluation

Clustering is commonly considered to be an unsupervised procedure, so evaluating its performance can be rather subjective. However, a great amount of effort has been invested in trying to standardize cluster evaluation metrics by using cluster validity indices (CVIs). Many indices have been developed over the years, and they form a research area of their own, but there are some overall details that are worth mentioning. The discussion here is based on Arbelaitz et al. (2013) and Wang and Zhang (2007), which provide a much more comprehensive overview.

In general, CVIs can be either tailored to crisp or fuzzy partitions. For the former, CVIs can be classified as internal, external or relative depending on how they are computed. Focusing on the first two, the crucial difference is that internal CVIs only consider the partitioned data and try to define a measure of cluster purity, whereas external CVIs compare the obtained partition to the correct one. Thus, external CVIs can only be used if the ground truth is known.

Note that even though a fuzzy partition can be changed into a crisp one, making it compatible with many of the existing "crisp" CVIs, there are also fuzzy CVIs tailored specifically to fuzzy clustering, and these may be more suitable in those situations. Fuzzy partitions usually have no ground truth associated with them, but there are exceptions depending on the task's goal (Lei et al., 2017).

Several of the best-performing CVIs according to Wang and Zhang (2007), Arbelaitz et al. (2013), and Lei et al. (2017) are implemented in **dtwclust** in the `cvi` function. Table 3 specifies which ones are available and some of their particularities.

There are some advantages and corresponding caveats with respect to the **dtwclust** implementations. Many internal CVIs require additional distance calculations, and some also compute so-called global centroids (a centroid that uses the whole dataset), which were calculated with, respectively, the Euclidean distance and a mean centroid in the original definition. The implementations in **dtwclust** change this, making use of whatever distance/centroid was utilized during clustering without further intervention from the user, so it is possible to leverage the distance and centroid functions that support time-series. Nevertheless, many CVIs assume symmetric distance functions, so the `cvi` function warns the user when this is not fulfilled.

Knowing which CVI will work best cannot be determined a priori, so they should be tested for each specific application. Many CVIs can be utilized and compared to each other, maybe using a majority vote to decide on a final result, but there is no best CVI, and it is important to conceptually understand what a given CVI measures in order to appropriately interpret its results. Furthermore, it should be noted that, due to additional distance and/or centroid calculations, computing CVIs can be prohibitive in some cases. For example, the Silhouette index effectively needs the whole distance matrix between the original series to be calculated.

CVIs are not the only way to evaluate clustering results. The **clue** package (Hornik, 2005,?) includes its own extensible framework for evaluation of cluster ensembles. It does not directly deal with the clustering algorithms themselves, rather with ways of quantifying agreement and consensus between several clustering results. As such, it is directly compatible with the results from **dtwclust**, since it does not care how a partition/hierarchy was created. Support for the **clue** package framework is included.

Cluster evaluation examples

In the following example, different numbers of clusters are computed, and, using internal CVIs, it is possible to assess which one resulted in a partition with more "purity". The majority of indices suggest using $k = 4$ in this case.

```
# subset
data <- CharTraj[1L:20L]
pc_k <- tsclust(data, k = 3L:5L, seed = 94L,
                distance = "dtw_basic", centroid = "pam")
names(pc_k) <- paste0("k_", 3L:5L)
sapply(pc_k, cvi, type = "internal")
                k_3          k_4          k_5
Sil    6.897035e-01 7.295148e-01 6.726453e-01
SF     1.105005e-11 1.345888e-10 1.074494e-10
CH     2.375816e+01 2.873765e+01 2.207096e+01
DB     4.141004e-01 3.225955e-01 2.858009e-01
DBstar 4.799175e-01 4.998963e-01 7.029138e-01
D      1.054228e+00 7.078230e-01 4.430916e-01
COP    1.176921e-01 7.768459e-02 7.153216e-02
```

If we choose the value of $k = 4$, we could then compare results among different random repetitions with help of the **clue** package (or with CVIs again).

CVI	Internal or external	Crisp or fuzzy partitions	Minimized or Maximized	Considerations
Rand	External	Crisp	Maximized	—
Adjusted rand	External	Crisp	Maximized	—
Jaccard	External	Crisp	Maximized	—
Fowlkes-Mallows	External	Crisp	Maximized	—
Variation of information	External	Crisp	Minimized	—
Soft rand	External	Fuzzy	Maximized	—
Soft adjusted rand	External	Fuzzy	Maximized	—
Soft variation of information	External	Fuzzy	Minimized	—
Soft normalized mutual information	External	Fuzzy	Maximized	—
Silhouette	Internal	Crisp	Maximized	Requires the whole cross-distance matrix.
Dunn	Internal	Crisp	Maximized	Requires the whole cross-distance matrix.
COP	Internal	Crisp	Minimized	Requires the whole cross-distance matrix.
Davies-Bouldin	Internal	Crisp	Minimized	Calculates distances to the computed cluster centroids.
Modified Davies-Bouldin (DB*)	Internal	Crisp	Minimized	Calculates distances to the computed cluster centroids.
Calinski-Harabasz	Internal	Crisp	Maximized	Calculates a global centroid.
Score function	Internal	Crisp	Maximized	Calculates a global centroid.
MPC	Internal	Fuzzy	Maximized	—
K	Internal	Fuzzy	Minimized	Calculates a global centroid.
T	Internal	Fuzzy	Minimized	—
SC	Internal	Fuzzy	Maximized	Calculates a global centroid.
PBMF	Internal	Fuzzy	Maximized	Calculates a global centroid.

Table 3: Cluster validity indices included in **dtwclust**. The first four are calculated with the `comPart` function from the **flexclust** package. The Silhouette index is calculated with the `silhouette` function in the **cluster** package. Internal fuzzy CVIs use the nomenclature from Wang and Zhang (2007).

```
require("clue")

pc_4 <- tsclust(data, type = "p", k = 4L,
                distance = "dtw_basic", centroid = "pam",
                control = partitional_control(nrep = 5L),
                seed = 95L)

names(pc_4) <- paste0("r_", 1L:5L)
pc_4 <- cl_ensemble(list = pc_4)
cl_dissimilarity(pc_4)

Dissimilarities using minimal Euclidean membership distance:
         r_1      r_2      r_3      r_4
r_2 3.464102
r_3 0.000000 3.464102
r_4 0.000000 3.464102 0.000000
r_5 0.000000 3.464102 0.000000 0.000000

table(Medoid = cl_class_ids(cl_medoid(pc_4)),
      "True Classes" = rep(c(4L, 3L, 1L, 2L), each = 5L))
       True Classes
Medoid 1 2 3 4
     1 5 0 0 0
     2 0 5 0 0
     3 0 0 5 0
     4 0 0 0 5
```

Comparing clustering algorithms with dtwclust

As we have seen, there are several aspects that must be considered for time-series clustering. Some examples are:

- Pre-processing of data, possibly changing the decision space.
- Type of clustering (partitional, hierarchical, etc.).
- Number of desired or expected clusters.
- Choice of distance measure, along with its parameterization.
- Choice of centroid function and its parameterization. This may also depend on the chosen distance.
- Evaluation of clustering results.
- Computational cost, which depends not only on the size of the dataset, but also on the complexity of the aforementioned aspects.

In order to facilitate more laborious workflows, **dtwclust** includes the `compare_clusterings` function which, along with its helper functions, optimizes the way the different clustering algorithms can be executed. Its main advantage is that it leverages parallelization. Using parallelization is not something that is commonly explored explicitly in the literature, but it can be extremely useful in practical applications. In the case of time-series clustering, parallel computation can result in a very significant reduction in execution times.

Handling parallelization has been greatly simplified in R by different software packages. The implementations done in **dtwclust** use the **foreach** package (Revolution Analytics and Weston, 2017) for multi-processing, and **RcppParallel** for multi-threading (Allaire et al., 2018). Thanks to **foreach**, the parallelized workflow can be executed not only in a local machine, but also in a computing cluster. In order to avoid data copies and communication overhead in these scenarios, `compare_clusterings` is coded in a way that, by default, less data is returned from the parallel processes. Nevertheless, as will be shown shortly, the results can be fully re-created in the main process on demand.

With this infrastructure, it is possible to cover the whole clustering workflow with **dtwclust**.

Parallelized workflow example

This example uses the **doParallel** package (Microsoft Corporation and Weston, 2018), which is one of the options that provides a parallel backend for **foreach**.

The configuration is specified with two helper functions: `compare_clusterings_configs` and `pdc_configs`. It tests partitional clustering with DTW distance and DBA centroids, exploring different values for window size and norm. The value of the window size can have a very significant effect on clustering quality (Dau et al., 2016)[3], but there is no single size that performs best on all datasets, so it is important to assess its effect on each specific case.

Since the ground truth is known in this scenario, an external CVI is chosen for evaluation: the adjusted Rand index. The `cvi_evaluators` function generates functions that can be passed to `compare_clusterings` which, internally, use the `cvi` function (see Cluster evaluation).

```
require("doParallel")
workers <- makeCluster(detectCores())
invisible(clusterEvalQ(workers, library(dtwclust)))
registerDoParallel(workers)

cfg <- compare_clusterings_configs(
    types = "partitional",
    k = 20L,
    controls = list(
        partitional = partitional_control(
            iter.max = 20L
        )
    ),
    distances = pdc_configs(
        "distance",
        partitional = list(
            dtw_basic = list(
                window.size = seq(from = 10L, to = 30L, by = 5L),
                norm = c("L1", "L2")
            )
        )
    ),
    centroids = pdc_configs(
        "centroid",
        share.config = c("p"),
        dba = list(
            window.size = seq(from = 10L, to = 30L, by = 5L),
            norm = c("L1", "L2")
        )
    ),
    no.expand = c(
        "window.size",
        "norm"
    )
)

evaluators <- cvi_evaluators("ARI", ground.truth = CharTrajLabels)

comparison <- compare_clusterings(CharTraj, types = "partitional",
                                  configs = cfg, seed = 8L,
                                  score.clus = evaluators$score,
                                  pick.clus = evaluators$pick)

stopCluster(workers); registerDoSEQ()

# some rows and columns from the results data frame
head(comparison$results$partitional[, c("config_id", "distance", "centroid",
                                        "window.size_distance", "norm_distance",
                                        "ARI")])
  config_id  distance centroid window.size_distance norm_distance       ARI
1   config1 dtw_basic      dba                   10            L1 0.6021905
2   config2 dtw_basic      dba                   10            L2 0.6589223
3   config3 dtw_basic      dba                   15            L1 0.5306598
```

[3] The strategy presented in this reference is also included in **dtwclust** in the `ssdtwclust` function, and it is implemented by leveraging `compare_clusterings`.

```
4   config4 dtw_basic      dba                     15            L2 0.4733479
5   config5 dtw_basic      dba                     20            L1 0.4474698
6   config6 dtw_basic      dba                     20            L2 0.5840729
```

Based on the ARI, one of the configurations was picked as the best one, and it is possible to obtain the clustering object by calling `repeat_clustering`:

```
clusters <- repeat_clustering(CharTraj, comparison, comparison$pick$config_id)

matrix(clusters@cluster, ncol = 5L, byrow = TRUE)
      [,1] [,2] [,3] [,4] [,5]
 [1,]    5    5    5    5    5
 [2,]    7    7    7    7    7
 [3,]   18   18   18   18   18
 [4,]   15   15   15   15   15
 [5,]   17   17   17   17   17
 [6,]    4    4    4    4    9
 [7,]    2    2    2    2    2
 [8,]    3    3    3    3   11
 [9,]    6    6    6    6    6
[10,]   20   20   20   20   20
[11,]   10   10   10   10   10
[12,]   10   19   19   19   19
[13,]   20   20   20   20   12
[14,]   14    8   16    8    8
[15,]    4    4    4    4    4
[16,]    2    2    2    2    2
[17,]    1    1    1   14    1
[18,]    6    6    6    6    6
[19,]   13   13   13   13    9
[20,]   18   12   17   17   17
```

Conclusion

In this manuscript a general overview of shape-based time-series clustering was provided. This included a lot of information related to the DTW distance and its corresponding optimizations, such as constraints and lower bounding techniques. At the same time, the **dtwclust** package for R was described and showcased, demonstrating how it can be used to test and compare different procedures efficiently and unbiasedly by providing a common infrastructure.

The package implements several different routines, most of which are related to the DTW algorithm. Nevertheless, its modular structure enables the user to customize and complement the included functionality by means of custom algorithms or even other R packages, as it was the case with **TSdist** and **clue**. These packages are more specialized, dealing with specific tasks (respectively: distance calculations and cluster evaluation). By contrast, **dtwclust** provides a more general purpose clustering workflow, having enough flexibility to allow for the most common approaches to be used.

The goal of this manuscript was not to give a comprehensive and thorough explanation of all the discussed algorithms, but rather to provide information related to what has been done in the literature, including some more recent propositions, so that the reader knows where to start looking for further information, as well as what can or cannot be done with **dtwclust**.

Choosing a specific clustering algorithm for a given application is not an easy task. There are many factors to take into account and it is not possible to know a priori which one will yield the best results. The included implementations try to use the native (and heavily optimized) R functions as much as possible, relying on compiled code where needed, so we hope that, if time-series clustering is required, **dtwclust** can serve as a starting point.

Bibliography

C. C. Aggarwal and C. K. Reddy. Time-series data clustering. In *Data Clustering: Algorithms and Applications*, chapter 15. CRC Press, 2013. [p22]

C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In J. V. den Bussche and V. Vianu, editors, *International Conference on Database Theory*, pages 420–434. Springer-Verlag, 2001. [p24]

S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah. Time-series clustering — a decade review. *Information Systems*, 53:16–38, 2015. URL https://doi.org/10.1016/j.is.2015.04.007. [p22, 23]

J. Allaire, R. Francois, K. Ushey, G. Vandenbrouck, M. Geelnard, and Intel. *RcppParallel: Parallel Programming Tools for 'Rcpp'*, 2018. URL https://CRAN.R-project.org/package=RcppParallel. R package version 4.4.2. [p39]

O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Pérez, and I. Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256, 2013. URL https://doi.org/10.1016/j.patcog.2012.07.021. [p36, 37]

N. Begum, L. Ulanova, J. Wang, and E. Keogh. Accelerating dynamic time warping clustering with a novel admissible pruning strategy. In *Conference on Knowledge Discovery and Data Mining*, KDD '15. ACM, 2015. ISBN 978-1-4503-3664-2/15/08. URL https://doi.org/10.1145/2783258.2783286. [p23, 26, 35]

D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, volume 10, pages 359–370. Seattle, WA, 1994. [p24]

A. M. Brandmaier. pdc: An R package for complexity-based clustering of time series. *Journal of Statistical Software*, 67(5):1–23, 2015. URL https://doi.org/10.18637/jss.v067.i05. [p23]

M. Cuturi. Fast global alignment kernels. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 929–936, 2011. [p27, 28]

M. Cuturi and M. Blondel. Soft-DTW: a differentiable loss function for time-series. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 894–903, International Convention Centre, Sydney, Australia, 2017. PMLR. URL http://proceedings.mlr.press/v70/cuturi17a.html. [p28, 32]

H. A. Dau, N. Begum, and E. Keogh. Semi-supervision dramatically improves time series clustering under dynamic time warping. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 999–1008. ACM, 2016. URL https://doi.org/10.1145/2983323.2983855. [p40]

T. Giorgino. Computing and visualizing dynamic time warping alignments in R: The dtw package. *Journal of Statistical Software*, 31(7):1–24, 2009. URL https://doi.org/10.18637/jss.v031.i07. [p23, 25, 26]

T. Hastie, R. Tibshirani, and J. Friedman. Cluster analysis. In *The Elements of Statistical Learning 2nd Edition*, chapter 14.3. Springer-Verlag, 2009. [p34]

K. Hornik. A CLUE for CLUster ensembles. *Journal of Statistical Software*, 14(12):1–25, 2005. URL https://doi.org/10.18637/jss.v014.i12. [p37]

K. Hornik. *clue: Cluster Ensembles*, 2019. URL https://CRAN.R-project.org/package=clue. R package version 0.3-57. [p37, 212]

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data. An Introduction to Cluster Analysis*, volume 1, chapter 1. John Wiley & Sons, 1990. [p22]

E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005. URL https://doi.org/10.1007/s10115-004-0154-9. [p26]

Y. Lei, J. C. Bezdek, J. Chan, N. X. Vinh, S. Romano, and J. Bailey. Extending information-theoretic validity indices for fuzzy clustering. *IEEE Transactions on Fuzzy Systems*, 25(4):1013–1018, 2017. URL https://doi.org/10.1109/TFUZZ.2016.2584644. [p37]

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. [p23]

D. Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition*, 42(9):2169 – 2180, 2009. ISSN 0031-3203. URL https://doi.org/10.1016/j.patcog.2008.11.030. [p24, 26, 27]

T. W. Liao. Clustering of time series data: A survey. *Pattern recognition*, 38(11):1857–1874, 2005. URL https://doi.org/10.1016/j.patcog.2005.01.025. [p22]

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml. [p23]

M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2019. URL https://CRAN.R-project.org/package=cluster. R package version 2.0.8. [p23]

D. Meyer and C. Buchta. *proxy: Distance and Similarity Measures*, 2019. URL https://CRAN.R-project.org/package=proxy. R package version 0.4-23. [p23]

Microsoft Corporation and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2018. URL https://CRAN.R-project.org/package=doParallel. R package version 1.0.14. [p39]

P. Montero and J. A. Vilar. TSclust: An R package for time series clustering. *Journal of Statistical Software*, 62(1):1–43, 2014. URL https://doi.org/10.18637/jss.v062.i01. [p23, 24]

U. Mori, A. Mendiburu, and J. A. Lozano. Distance measures for time series in R: The TSdist package. *R Journal*, 8(2):451–459, 2016. URL https://journal.r-project.org/archive/2016/RJ-2016-058/index.html. [p23, 24]

J. Paparrizos and L. Gravano. k-Shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1855–1870, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-2758-9. URL https://doi.org/10.1145/2949741.2949758. [p23, 28, 33, 35]

F. Petitjean, A. Ketterlin, and P. Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678 – 693, 2011. ISSN 0031-3203. URL https://doi.org/10.1016/j.patcog.2010.09.013. [p32]

S. Rani and G. Sikka. Recent techniques of clustering of time series data: A survey. *International Journal of Computer Applications*, 52(15), 2012. URL https://doi.org/10.5120/8282-1278. [p22]

C. A. Ratanamahatana and E. Keogh. Everything you know about dynamic time warping is wrong. In *Third Workshop on Mining Temporal and Sequential Data*. Citeseer, 2004. [p24, 25, 26]

Revolution Analytics and S. Weston. *foreach: Provides Foreach Looping Construct for R*, 2017. URL https://CRAN.R-project.org/package=foreach. R package version 1.4.4. [p39]

H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49, 1978. ISSN 0096-3518. URL https://doi.org/10.1109/TASSP.1978.1163055. [p26]

W. Wang and Y. Zhang. On fuzzy cluster validity indices. *Fuzzy sets and systems*, 158(19):2095–2117, 2007. URL https://doi.org/10.1016/j.fss.2007.03.004. [p36, 37, 38]

Alexis Sardá-Espinosa
alexis.sarda@gmail.com
Disclaimer: The software package was developed
independently of any organization or institution
that is or has been associated with the author.

mixedsde: A Package to Fit Mixed Stochastic Differential Equations

by Charlotte Dion, Simone Hermann, Adeline Samson

Abstract Stochastic differential equations (SDEs) are useful to model continuous stochastic processes. When (independent) repeated temporal data are available, variability between the trajectories can be modeled by introducing random effects in the drift of the SDEs. These models are useful to analyze neuronal data, crack length data, pharmacokinetics, financial data, to cite some applications among other. The `R` package focuses on the estimation of SDEs with linear random effects in the drift. The goal is to estimate the common density of the random effects from repeated discrete observations of the SDE. The package **mixedsde** proposes three estimation methods: a Bayesian parametric, a frequentist parametric and a frequentist nonparametric method. The three procedures are described as well as the main functions of the package. Illustrations are presented on simulated and real data.

Introduction

Continuous stochastic processes are usually observed discretely in time (with equidistant time points or not) leading to times series, although their intrinsic nature is of continuous time. While discrete time stochastic models such as auto-regressive models (ARMA, GARCH, ...) have been widely developed for time series with equidistant times, more and more attention have been focused on Stochastic Differential Equations (SDEs). Examples of applications where SDEs have been used include dynamics of thermal systems (Bacher and Madsen, 2011), solar and wind power forecasting (Iversen et al., 2014), neuronal dynamics (Ditlevsen and Samson, 2014), pharmacokinetic/pharmacodynamic (PK/PD) (Hansen et al., 2014), crack growth (Hermann et al., 2016). Estimation for SDE is available in different softwares. We can cite among others the computer software CTSM with a (extended) Kalman filter approach (Kristensen and Madsen, 2003), the **sde** package which proposes several tools for the simulation and the estimation of a variety of SDEs, and more recently the R-packages **Sim.DiffProc** (Guidoum and Boukhetala, 2017) and **yuima** (Iacus, 2018) (the last one proposes also some tools for quantitative finance).

Depending on the applications, independent repeated temporal measures might be available. For examples, drug concentration of several subjects is usually observed in PK; dynamics of several neurons is measured along time; time to crack lengths can be measured repeatedly in crack growth study. Each trajectory represents the behavior of a unit/subject. The functional form is similar for all the trajectories. Fitting the overall data simultaneously obviously improves the quality of estimation, but one has to take into account these variabilities between experiments. This is the typical framework of mixed-effects models where some parameters are considered as random variables (random effects) and proper to each trajectory. Hence the random effects represent the particularity of each subject. Some parameters can also be considered as common to all the trajectories (fixed effects).

In this work the model of interest is thus a mixed-effects stochastic differential equation (MSDE), mixed-effects for both fixed and random effects. The **mixedsde** package has been developed to estimate the density of the random effects from the discrete observations of M independent trajectories of a MSDE. It is available from the Comprehensive R Archive Network (CRAN Dion et al., 2016). The package's development is actively continued with the latest source code available from a GitHub repository `https://github.com/charlottedion/mixedsde`.

More precisely, we focus on MSDE with linear drift. We consider M diffusion processes $(X_j(t), t \geqslant 0)$, $j = 1, \ldots, M$ with dynamics ruled by SDE, for $t \in [0, T]$

$$\begin{cases} dX_j(t) & = (\alpha_j - \beta_j X_j(t))dt + \sigma a(X_j(t))dW_j(t) \\ X_j(0) & = x_j \end{cases} \tag{1}$$

where $(W_j)_{1 \ldots j \ldots M}$ are M independent Wiener processes, (α_j, β_j) are two (random) parameters, $\sigma a(X_j(\cdot))$ is the diffusion coefficient with a a known function and σ an unknown constant. The initial condition x_j is assumed fixed (and known) in the paper with possibly different values for each trajectory.

In the package, we restrict the models to the two famous SDEs with linear drift, namely the Ornstein-Uhlenbeck model (OU) with $a(x) = 1$ and the Cox-Ingersoll-Ross model (CIR) with $a(x) = \sqrt{x}$. For the CIR model, we assume that $x_j > 0$, $\sigma > 0$, $\alpha_j > \sigma^2/2$ and $\beta_j > 0$ to ensure that the process never crosses zero.

The random parameters are denoted ϕ_j and belong to $\mathbb{R}^d$ with either $d = 1$ or $d = 2$:

- $(d = 1)$ $\phi_j = \alpha_j$ random and for all $j = 1, \dots, M$, $\beta_j = \beta$ fixed,
- $(d = 1)$ $\phi_j = \beta_j$ random and for all $j = 1, \dots, M$, $\alpha_j = \alpha$ fixed,
- $(d = 2)$ $\phi_j = (\alpha_j, \beta_j)$ random.

The ϕ_j's are assumed independent and identically distributed (*i.i.d.*) and independent of the W_j's. The **mixedsde** package aims at estimating the random effects ϕ_j and their distribution whose density is denoted f, from N discrete observations of the M trajectories $(X_j(t))_j$ from Equation 1 at discrete times $t_0 = 0 < t_1 < \ldots < t_N = T$ (not necessarily equidistant).

Context: To the best of our knowledge, this is the first package in `R` language dedicated to the estimation of MSDE. The main software considering mixed models is MONOLIX (2003) but methods for mixed stochastic differential equations are not implemented for `R`. One package named **PSM** (Mortensen and Klim, 2013) provides functions for estimation of linear and non-linear mixed-effects models using stochastic differential equations. But the model includes measurement noise and proposes only parameter estimation. Moreover, there is no mathematical property about the used estimators. In this context, the package presented is this paper is pioneer.

Estimation procedures for MSDE have been proposed in the non-parametric and the parametric frameworks, with a frequentist and a Bayesian point of view. The parametric approaches assume Gaussian random effects ϕ_j. Among other references, for parametric maximum likelihood estimation, we can cite Ditlevsen and de Gaetano (2005); Picchini et al. (2010) (Hermite expansion of the likelihood); Delattre et al. (2013) (explicit integration of the Girsanov likelihood) or Delattre et al. (2016) (mixture of Gaussian distributions for the random effects); for parametric Bayesian estimation, we can cite Oravecz et al. (2009) (restricted to Ornstein-Uhlenbeck) and Hermann et al. (2016) (general methodology); for non-parametric estimation, we can cite Comte et al. (2013); Dion (2014); Dion and Genon-Catalot (2015) (kernel estimator and deconvolution estimators).

Three estimation procedures are implemented in the **mixedsde** package: a kernel nonparametric estimator (Dion and Genon-Catalot, 2015), a parametric maximum likelihood estimator (Delattre et al., 2013) and a parametric Bayesian estimator (Hermann et al., 2016). The parametric frequentist and Bayesian approaches assume the random effects Gaussian. The Bayesian approach seems the most appropriate method for a small time of observation T and a small number of trajectories M. The nonparametric approach can be used when no prior idea on the density is available and when T and M are both large enough. Finally, the parametric frequentist estimation can be used with a large number of discrete observations.

This paper reviews in Section 8.2 the three estimation methods. An overview of the **mixedsde** package is given in Section 8.3 through a description of the main functions and of other related companion functions. The practical use of this package is illustrated in Section 8.4 on simulated data and in Section 8.5 on one real dataset in neuronal modeling.

Density estimation in mixed stochastic differential models

We briefly recall the methodology of the three estimators implemented in the **mixedsde** package. We start with the nonparametric approach, then the frequentist parametric Gaussian method and finally the Bayesian parametric Gaussian method.

Nonparametric estimation of the random effects density

The first step of the nonparametric approach is to estimate the random effects. The idea is to maximize the likelihood of the process X_j^{φ} solution of the stochastic differential equation with fixed φ. Assuming continuous observations of $(X_j(t), 0 \leqslant t \leqslant T)$, the likelihood function is obtained with the Girsanov formula:

$$\ell_T(\varphi) = \exp\left(\int_0^T \frac{\alpha - \beta X_j^{\varphi}(s)}{\sigma^2 a^2(X_j^{\varphi}(s))} dX_j(s) - \frac{1}{2}\int_0^T \frac{(\alpha - \beta X_j^{\varphi}(s))^2}{\sigma^2 a^2(X_j^{\varphi}(s))} ds\right).$$

Maximizing the likelihood yields to the following estimator of ϕ_j

$$A_j := V_j^{-1} U_j \tag{2}$$

where U_j and V_j are the two sufficient statistics of the model. They are explicit depending on the form of the random effects:

- α_j random and β known

$$U_j := \int_0^T \frac{1}{\sigma^2 a^2(X_j(s))} dX_j(s) + \beta \int_0^T \frac{X_j(s)}{\sigma^2 a^2(X_j(s))} ds, \; V_j := \int_0^T \frac{1}{\sigma^2 a^2(X_j(s))} ds,$$

- β_j random and α known

$$U_j := -\int_0^T \frac{X_j(s)}{\sigma^2 a^2(X_j(s))} dX_j(s) + \alpha \int_0^T \frac{X_j(s)}{\sigma^2 a^2(X_j(s))} ds, \quad V_j := \int_0^T \frac{X_j(s)^2}{\sigma^2 a^2(X_j(s))} ds,$$

- (α_j, β_j) random, denote $b(x) = (1, -x)^t$ with u^t the transposition of vector u. Here U_j is a column vector with size 2×1 and $V_j = (V_{j,k,\ell})_{k,\ell \in \{1,2\}}$ a 2×2 symmetric matrix:

$$U_j := \int_0^T \frac{b}{\sigma^2 a^2}(X_j(s)) dX_j(s), \; V_j := \int_0^T \frac{b\, b^t}{\sigma^2 a^2}(X_j(s)) ds. \tag{3}$$

Truncated versions of this estimator have been introduced for theoretical reasons. In the bidimensional case $\phi_j = (\alpha_j, \beta_j)$, Dion and Genon-Catalot (2015) propose the following estimator

$$\widehat{A_j} := A_j \mathbf{1}_{B_j}, \quad B_j := \{V_j \geqslant \kappa\sqrt{T} I_2\} = \{\min(\lambda_{1,j}, \lambda_{2,j}) \geqslant \kappa\sqrt{T}\} \tag{4}$$

with I_2 the 2×2 identity matrix and $\lambda_{i,j}, i = 1,2$ the two eigenvalues of the symmetric non negative matrix V_j, and κ a numerical constant that has been calibrated (Dion and Genon-Catalot, 2015). In the one-dimensional case $\phi_j = \beta_j$ with $\alpha = 0$, Genon-Catalot and Larédo (2016) propose

$$\widehat{A_j} := A_j \mathbf{1}_{V_j \geqslant \kappa\sqrt{T}} \tag{5}$$

with κ a numerical constant calibrated in practice. Based on these estimators of the ϕ_j's, we can proceed to the second step, the estimation of their density f. Several nonparametric estimators of f have been proposed (see Comte et al., 2013, for example). In the package **mixedsde**, we focus on the kernel estimator of f. Let us introduce the kernel function $K : \mathbb{R}^d \to \mathbb{R}$, with $d = 1,2$ depending on the dimension of ϕ_j. We assume K to be a $\mathcal{C}^2$ function satisfying

$$\int K(u)du = 1, \quad \|K\|^2 = \int K^2(u)du < +\infty, \quad \int (\nabla K(u))^2 du < +\infty$$

(with ∇K the gradient of K). A bandwidth $h \in (\mathbb{R}^+)^d$, for $d = 1,2$, is used to define the function

$$K_h(x) = \frac{1}{h} K\left(\frac{x}{h}\right), x \in \mathbb{R}^d.$$

Note that in the bidimensional case, $h = (h_1, h_2)$ and the two marginal bandwidths are different. The nonparametric estimator of the density f of ϕ_j is

$$\widehat{f_h}(x) = \frac{1}{M} \sum_{j=1}^{M} K_h(x - A_j). \tag{6}$$

and the estimator $\widehat{\widehat{f_h}}(x) = \frac{1}{M} \sum_{j=1}^{M} K_h(x - \hat{A}_j)$ is computed when the truncated estimator $\hat{A}_j$ is different than A_j.

In the **mixedsde** package, Gaussian kernel estimators are implemented with the `R` -functions `density` (available in package **stats**) when $d = 1$ and `kde2d` (available in package **MASS** Venables and Ripley (2016)) when $d = 2$ with an automatic selection of the bandwidth h. Note that when there is only one random effect, the bandwidth is selected by unbiased cross-validation with the argument `bw="ucv"`, or as the default value given by the rule-of-thumb if the chosen bandwidth is too small. Note that the estimator is unstable for small variance of the random effects.

It is important to notice that the two random effects are not assumed independent. When there is only one random effect, the fixed parameter has to be entered by the user.

The computation of $A_j = V_j^{-1} U_j$ does not require the knowledge of σ^2 as it appears both in U_j

and V_j. It requires however the evaluation of the two continuous integrals in U_j and V_j while observing the trajectories (X_j) at discrete times $(t_0, t_1, \ldots, t_N)$. For $\Delta_k = t_{k+1} - t_k$, $k = 0, \ldots, N-1$, the non-stochastic integrals $\int_0^T g(X_j(s))ds$ for the functions $g = \frac{b}{a^2}$ or $g = \frac{b\,b^t}{a^2}$ are approximated by

$$\int_0^T g(X_j(s))ds \approx \sum_{k=0}^{N-1} g(X_j(t_k))\Delta_k.$$

For the stochastic integrals, we use the following simple discretization

$$\int_0^T g(X_j(s))dX_j(s) \approx \sum_{k=0}^{N-1} g(X_j(t_k))(X_j(t_{k+1}) - (X_j(t_k)))\Delta_k.$$

Note that there is no integrability issue for these two types of integrals considering the two functions $g = \frac{b}{a^2}$ or $g = \frac{b\,b^t}{a^2}$ involved in the sufficient statistics.

Frequentist parametric estimation approach

In this section and the following one, we assume that the random parameters ϕ_j are Gaussian:

- when $d = 1$, $\phi_j \sim \mathcal{N}(\mu, \omega^2)$ with $\mu \in \mathbb{R}$,
- when $d = 2$, $\phi_j \sim \mathcal{N}(\mu, \Omega)$ with $\mu \in \mathbb{R}^2$ and a diagonal covariance matrix $\Omega = \text{diag}(\omega_1^2, \omega_2^2)$.

For the bidimensional case $d = 2$ we estimate by maximum likelihood the parameters $\theta := (\mu, \Omega)$. We define the likelihood function assuming first that the trajectories are continuously observed, similarly to the nonparametric approach (Section 8.2.1). Thanks to the Girsanov formula, the likelihood function of the j^{th} trajectory X_j is

$$L(X_j, \theta) = \frac{1}{\sqrt{\det(I_2 + \Omega V_j)}} \exp\left[-\frac{1}{2}(\mu - V_j^{-1}U_j)' R_j^{-1}(\mu - V_j^{-1}U_j)\right] \exp\left(\frac{1}{2}U_j' V_j^{-1} U_j\right)$$

with $R_j^{-1} = (I_2 + V_j\Omega)^{-1}V_j$ and I_2 is the 2×2 identity matrix.

For the case $d = 1$, the parameters to estimate are $\theta := (\mu, \omega, \psi)$ where ψ denotes the fixed effect α or β. We adopt the subscript r for the value of random, equal to 1 or 2, and c for the position of the common fixed effect (thus 2 or 1). The likelihood function of the j^{th} trajectory X_j is

$$\begin{aligned} L(X_j, \theta) = & \frac{1}{\sqrt{1 + \omega^2 V_{j,r,r}}} \exp\left[-\frac{1}{2}V_{j,r,r}(1 + \omega^2 V_{j,r,r})^{-1}(\mu - V_{j,r,r}^{-1}(U_{j,r} - \psi V_{j,c,r}))^2\right] \\ & \times \exp\left(\psi U_{j,c} - \frac{\psi^2}{2}V_{j,c,c}\right) \exp\left(\frac{1}{2}(U_{j,r} - \psi V_{j,r,c})^2 V_{j,r,r}^{-1}\right) \end{aligned}$$

with the notations U, V from Equation 3. Details on this formula are available in the Appendix 8.6.

The likelihood function is defined as $L(\theta) = \prod_{j=1}^M L(X_j, \theta)$. The maximum likelihood estimator $\widehat{\theta} := (\widehat{\mu}, \widehat{\Omega}, \widehat{\psi})$ when $d = 1$ and $\widehat{\theta} := (\widehat{\mu}, \widehat{\Omega})$ when $d = 2$ is defined by

$$\widehat{\theta} = \arg\max_{\theta} L(\theta) = \arg\max_{\theta} \prod_{j=1}^M L(X_j, \theta). \tag{7}$$

This estimator is not explicit. In the **mixedsde** package, the function `optim` is used to maximize numerically the likelihood. The maximum is generally not unique and depend on the initialization. A good initialization is another estimator, for example the moment estimator of θ. Function `optim` is thus initialized with the mean and the variance of the estimators A_j of the random parameters (see Equation 2). Sufficient statistics U_j and V_j are discretized as explained in Section 8.2.1.

Note that this parametric approach requires the knowledge of σ^2 to compute the sufficient statistics U_j and V_j because V_j appears alone in R_j. We plug the following estimator of σ^2

$$\widehat{\sigma^2} = \frac{1}{M}\sum_{j=1}^M \left(\frac{1}{N}\sum_{k=0}^{N-1} \frac{(X_j(t_{k+1}) - X_j(t_k))^2}{\Delta_k a^2(X_j(t_k))}\right). \tag{8}$$

Selection of (non-nested) models can be performed with the BIC criteria, defined by $-2\log L(\widehat{\theta}) + 2\log(M)$ for model with one random effect and $-2\log L(\widehat{\theta}) + 4\log(M)$ with two random effects and the AIC criteria defined by $-2\log L(\widehat{\theta})+2$ for one random effect and $-2\log L(\widehat{\theta})+4$ for two random effects. These asymptotic criteria indicate the trade-off between maximizing fit and minimizing model complexity. Note that their theoretical properties are guaranteed only when σ^2 is known.

Theoretical results are obtained on these estimators in the continuous observations context under the asymptotic regime $T\to\infty$, $N\to\infty$, see (Dion and Genon-Catalot, 2015; Delattre et al., 2013). For discrete observations, similar results are obtained in the high frequency context: $T=n\Delta$, $n\to\infty$ ($\Delta\to 0$). Nevertheless, in practice the points may not be equidistant and the package allows a non-regular grid. The influence of T is lighter in the parametric strategy. Moreover, asymptotic normality is obtained under the additional assumption $n/N\to\infty$.

Bayesian parametric approach

For the Bayesian approach we assume similarly to the frequentist parametric estimation method a Gaussian distribution for ϕ_j, with a diagonal covariance matrix $\Omega=\mathrm{diag}(\omega_1^2,\omega_2^2)$. In this method, we estimate in the same time the diffusion coefficient σ. The parameters of interest are thus $\theta=(\mu,\Omega,\sigma)$ and we want to estimate their posterior distribution $p(\theta|(X_j(t_k))_{j=1,\ldots,M,k=1,\ldots,N})$. Let denote $\mathbf{X}_{1:M}=(X_j(t_k))_{j=1,\ldots,M,k=1,\ldots,N}$ in the following.

We now introduce prior distributions implemented in **mixedsde** package for the parameters θ:

$$\begin{aligned}\mu &\sim \mathcal{N}(m,V),\ V=\mathrm{diag}(v)\\ \omega_i^2 &\sim \mathrm{IG}(\alpha_{\omega,i},\beta_{\omega,i}),\ i=1,2\\ \sigma^2 &\sim \mathrm{IG}(\alpha_\sigma,\beta_\sigma),\end{aligned}$$

where IG is the Inverse Gamma distribution which is conjugate to the normal likelihood and $m,V,\alpha_{\omega,i},\ \beta_{\omega,i},\alpha_\sigma,\beta_\sigma$ are hyperparameters fixed by the user. The case of only one random effect is nested by setting ω_1^2 or ω_2^2 equal to zero.

The aim is to calculate the posterior distribution $p(\theta|\mathbf{X}_{1:M})$ which is not explicit for the whole vector of parameters. Therefore, we simulate it through a Gibbs sampler (see e.g., Robert and Casella, 2004). Here, we have a true transition density of both processes that is used for the likelihood, see Iacus (2008). For a general hierarchical diffusion approach based on the Euler approximation, see Hermann et al. (2016).

Analogically to the frequentist approach, there is a first step: sample from the full conditional posterior of the random effects $p(\phi_j|(X_j(t_k))_{k=1,\ldots,N},\theta)$, $j=1,\ldots,M$. This is done by a Metropolis Hastings (MH) algorithm.

The second step is the estimation of the hierarchical parameters μ and Ω. Full conditional posteriors $p(\mu|\phi_1,\ldots,\phi_M,\Omega)$ (resp. $p(\Omega|\phi_1,\ldots,\phi_M,\mu)$) are Gaussian (resp. inverse Gamma) and can, for example, be found in Hermann et al. (2016).

The last step of the Gibbs sampler is sampling from the full conditional posterior of σ^2. For the CIR model, this is also conducted by a MH step. For the OU model, the inverse Gamma distribution is conjugate to the normal likelihood. The full conditional posterior distribution is given by

$$\sigma^2|\mathbf{X}_{1:M},\phi_1,...,\phi_M\sim$$
$$\mathrm{IG}\left(\alpha_\sigma+\frac{MN}{2},\beta_\sigma+\frac{1}{2}\sum_{j=1}^{M}\sum_{k=1}^{N}\frac{\beta_j}{1-e^{-2\beta_j\Delta_k}}\left(X_j(t_k)-\frac{\alpha_j}{\beta_j}-\left(X_j(t_{k-1})-\frac{\alpha_j}{\beta_j}\right)e^{-\beta_j\Delta_k}\right)^2\right).$$

In the case of one random effect, there is one additional Gibbs sampler step for the fixed effect, that is also conducted through a MH algorithm.

In the package, the starting values for the Gibbs sampler are set equal to the mean of the prior distributions. In all the MH algorithms, one each has to choose a proposal density. In the package **mixedsde**, we use a normal density for all location parameters with mean equal to the last chain iteration and a proposal variance that has to be chosen. For the CIR model, the proposal distribution for σ^2 is chosen by $\sqrt{\sigma^2}\sim\mathcal{N}(\sqrt{\sigma^2_{\text{prev}}},\text{variance})$ where σ^2_{prev} is the previous value of σ^2. The remaining question is how to choose the suitable proposal variance. This variance controls the chain dependence and the acceptance rate. If the variance is small, the acceptance rate is large and the chains gets very dependent. If the proposal variance is large, only few candidates are accepted with the advantage of weakly dependent chains. This problem is solved in the package with an adaptive Metropolis-within Gibbs algorithm (Rosenthal, 2011) using the proposal distribution $\mathcal{N}(0,e^{2l})$ with l the logarithm of the standard deviation of the increment. This parameter is chosen so that the acceptance rate is approximately 0.44 which is proposed to be optimal in the Metropolis-within Gibbs sampler (Rosenthal, 2011). It is proposed to add/subtract an adoption amount $\delta(n) = \min(0.1, n^{-1/2})$ to/from t after every 50th iteration and adapt the proposal variance if the acceptance rate is smaller than 0.3 or larger than 0.6.

Predictions

In many cases, one is not only interested in parameter estimation but also in the prediction for future observations. The first step is the prediction of a future random effect ϕ_{pred}. The simulation of a new random effect is direct for the frequentist parametric approach sampling from $\mathcal{N}(\widehat{\mu}, \widehat{\Omega})$. For the nonparametric approach, first note that $\widehat{f}_h$ is an estimator given on a discrete grid $\{x_1, \ldots, x_n\}$, i.e. a vector of corresponding $\{p_1, \ldots, p_n\}$ after normalisation. Simulating from the estimator $\widehat{f}_h$ can therefore be performed simulating a discrete variable from vector $\{x_1, \ldots, x_n\}$ with (normalized) probabilities $\{p_1, \ldots, p_n\}$. For the Bayesian approach, a new ϕ_{pred} is sampled from the predictive distribution $p(\phi_{\text{pred}}|\mathbf{X}_{1:M}) = \int p(\phi_{\text{pred}}|\mu, \Omega)p(\mu, \Omega|\mathbf{X}_{1:M})\, d(\mu, \Omega)$ where the posterior of μ and Ω is approximated by the results of the Gibbs sampler. This distribution is not explicit, and hence we suggest to sample over a grid through inversion method, equal to the nonparametric case.

Given a new random effect ϕ_{pred}, we are able to simulate predictive trajectories. This is performed using the transition density $p(X(t_k)|X(t_{k-1}), \phi_{\text{pred}}, \sigma^2)$ for the frequentist approach. The starting points of the process x_j are the observed ones. For the Bayesian approach, we implement two prediction settings. Firstly, analogously to the frequentist approach a new trajectory is simulated using the transition density $p(X(t_k)|X(t_{k-1}), \phi_{\text{pred}}, \sigma^2)$ where ϕ_{pred} is sampled from the MCMC (Markov chain Monte Carlo) posterior distribution $p(\phi|X_{1:M})$. Secondly, we can calculate the predictive distribution

$$p(X(t_i)|\mathbf{X}_{1:M}) = \int p(X(t_i)|\phi_{\text{pred}}, \sigma^2)p(\phi_{\text{pred}}, \sigma^2|\mathbf{X}_{1:M})\, d(\phi_{\text{pred}}, \sigma^2)$$

in each time point. We can then calculate only the quantiles for a prediction interval or to draw directly samples from the predictive distribution. For this predictive distribution, we take the starting point $x_j = x_0$ to be the same for all series. If the starting points would vary, this is an additional random effect whose density has to be estimated. This is not implemented in the estimation procedure and will, therefore, left out for the prediction.

It is then interesting to compare the new trajectories with the real ones. If the number of new trajectories is large enough we compute an empirical confidence interval.

Overview of the mixedsde functions

This Section presents an overview of the functions implemented in the package. Illustrations of the code are given in Section 8.4.

Data

Data is a matrix `X` of size $M \times N$ for M trajectories with N time points. The time points are not necessarily equidistant but are the same for the M trajectories. These time points are gathered in the vector `times` of length N. Real datasets are available on the package, and detailed on Section 8.5.

To lead a simulation study, the function `mixedsde.sim` allows to generate a list with a $M \times N$ matrix `X` of M trajectories on the interval $[0, T]$ with N equidistant points (default value 100) and a vector `times` with the equidistant times. This function leans on function `sde.sim` available via package **sde** (Iacus, 2006) to simulate SDE. One has to choose: `model` either OU or CIR; `random` that fixes the position and the number of random effects: `random = 1` for α_j random, `random = 2` for β_j random or `random = c(1,2)` for α_j and β_j random; σ the diffusion coefficient; `invariant`, default value 0 means that X_0 is 0 (default) or fixed by the user, value 1 means that X_0 is generated from the invariant distribution (see details in the package documentation); `density.phi` to choose the distribution of the random effect (see package documentations).

Main function

Main function is `mixedsde.fit` producing estimation of the random effects and their common density. Inputs of `mixedsde.fit` are

Class	Freq.fit	Bayes.fit
Method	out	out
Method	plot	plot
Method	–	plot2compare
Method	print	print
Method	summary	summary
Method	pred	pred
Method	valid	valid

Table 1: Summary of the different methods for the two S4-classes `Freq.fit` and `Bayes.fit` resulting of the package **mixedsde**.

- `X` a $M \times N$ matrix containing the trajectories by rows.
- `times` The vector of observations times.
- `model` The chosen model either OU or CIR.
- `random` It fixes the position and the number of random effects: `random = 1` for α_j random, `random = 2` for β_j random or `random = c(1,2)` for α_j and β_j random.
- `estim.method` The estimation method: `nonparam` (see Section 8.2.1), `paramML` (see Section 8.2.2) or `paramBayes` (see Section 8.2.3).
- `fixed` The value of the fixed effect β (resp. α) when `random = 1` (resp. `random = 2`), default 0. (Only for the frequentist approaches).
- `estim.fix` 1 if the fixed effect is estimated, default 0. (Only for the frequentist parametric approach when `random=1` or 2).
- `gridf` The x-axis grid on which the random effect distribution is computed: we recommend a fine grid with at least 200 points, default value is a sequence of length 500 starting in $0.8 \times \min_j \widehat{\phi}_j$ and ending in $1.2 \times \max_j \widehat{\phi}_j$. (Only for the frequentist approaches).
- `prior` The list of prior parameters `m,v,alpha.omega,beta.omega,alpha.sigma, beta.sigma` for `paramBayes` method: Default values are calculated based on the estimations $(A_j)_j$ for the first $\min(3, \lceil M \cdot 0.1 \rceil)$ series and main estimation is only made with the remaining $\lfloor M \cdot 0.9 \rfloor$. (Only for the Bayesian approach).
- `nMCMC` The length of the Markov chain for `paramBayes` method. (Only for the Bayesian approach).

Note that for the frequentist approach if there is only one random effect, then the user has the choice: fix it to a value of the user choice (using: `fixed`= the value and `estim.fix`=0) or estimate it through the package (choosing `estim.fix=1`. In the following we describe the related methods, proposed in the package, they are summarized in Table 1.

Outputs

Output of `mixedsde.fit` is a S4 class called `Freq.fit` for the frequentist approaches and `Bayes.fit` for the Bayesian approach. Results of the estimation procedure are available as a list applying function `out` to the `Freq.fit` (resp. `Bayes.fit`) object.

Elements of `Freq.fit` are:

- `sigma2` Estimator $\widehat{\sigma^2}$ given in Equation 8 of the diffusion coefficient.
- `estimphi` Estimator $(A_j)_j$ given in Equation 2 of the random effects.
- `estimphi.trunc` The truncated estimator $(\widehat{A_j})_j$ given in Equation 4 or 5 of the random effects.
- `estim.fixed` The estimator of the fixed effect if `random = 1` or 2, `estim.method = paramML; estim.fix = 1`, default 0.
- `gridf` The x-axis grid on which the random effect distribution is computed.
- `estimf` The estimator of the density of the random effects (for both `paramML` method with Equation 7 and `nonparam` method with Equation 6).
- `cutoff` Binary M-vector of binary values indicating the truncated trajectories, default `FALSE` when no truncation.
- `estimf.trunc` The truncated estimation of the density of the random effects.
- `mu` Estimation of Gaussian mean of the random effects (only for `paramML` method from Equation 7).
- `omega` Estimation of Gaussian variance matrix of the random effects (only for `paramML` method method from Equation 7).
- `aic` and `bic` AIC and BIC criteria (only for `paramML` method).
- `index` Indices of trajectories used for the estimation, excluded are trajectories with $V_j = 0$ or $V_j = +\infty$ (one random effect) or $\det V = +\infty$ (two random effects), trajectories containing negative values for `CIR` model.

Elements of `Bayes.fit` are:

- `sigma2` Trace of the Markov chain simulated from the posterior of σ^2.
- `mu` Trace of the Markov chain simulated from the posterior of μ .
- `omega` Trace of the Markov chain simulated from the posterior of ω^2.
- `alpha` Trace of the Markov chain simulated from the posterior of α_j, `nMCMC`$\times M$ matrix if α is random effect, `nMCMC`$\times 1$ otherwise.
- `beta` Trace of the Markov chain simulated from the posterior of β_j, `nMCMC`$\times M$ matrix if β is random effect, `nMCMC`$\times 1$ otherwise.
- `burnIn` A proposal for the burn-in phase.
- `thinning` A proposal for the thin rate.
- `ind.4.prior` The indices used for the prior parameter calculation, $M + 1$ if prior parameters were specified.

Outputs `burnIn` and `thinning` are only proposals for a burn-in phase and a thin rate. The proposed `burnIn` is calculated by dividing the Markov chains into 10 blocks and calculate the 95% credibility intervals and the respective mean. Starting in the first one, the block is taken as burn-in as long as the mean of the current block is not in the credibility interval of the following block or vice versa. The thinning rate is proposed by the first lag which leads to a chain autocorrelation of less than 80%. It is not easy to automate these choices, so it is highly recommended by the authors to plot the chains and look at the mixing property (the chain should not be piecewise constant).

Command `plot()` applied to a `Freq.fit` object produces a frequencies histogram of $(A_j(T))_j$ (one or two according to the number of random effects) with the estimated density (red curve) and the truncated estimator if available (dotted grey red curve) and a quantile-quantile graph with the quantiles of the A_j's versus the quantiles of a normal sample of the same length, with the same empirical mean and standard deviation. This illustrates the normality of the sample. Applying this function to the nonparametric results indicates if the Gaussian assumption of the parametric approach is appropriate. When `plot()` is applied to a `Bayes.fit` object, one can choose four different options, named `style`. The default value is `chains`, it plots the Markov chains for the different parameter values. `acf` leads to the corresponding autocorrelation functions, `density` to the approximated densities for each parameter and `cred.int` leads to the credibility intervals of the random parameters with the input parameter `level` with default 0.05. For all options, with the input parameter `reduced = TRUE`, the burn-in period is excluded and a thinning rate is taken, default is `FALSE`. There is also a possibility to include the prior means in the plots by lines with `plot.priorMean = TRUE`, default is `FALSE`.

In the Bayesian estimation the influence of prior parameters is interesting, thus for the `Bayes.fit` object, there is a second plot method, named `plot2compare` where three estimation objects can be compared. For reasons of clarity, only the densities are compared, with the default `reduced =`

TRUE. Here, there is also a possibility to include `true.values`, a list of the true parameters for the comparison in a simulation example.

Command `summary()` applied to a `Freq.fit` object computes the kurtosis and the skewness of the distribution, $\widehat{\sigma^2}$, the empirical mean and standard deviation computed from the estimators $(A_j)_j$, $\widehat{\mu}$, $\widehat{\Omega}$ (and the fixed effect $\widehat{\alpha}$ or $\widehat{\beta}$), AIC, BIC criteria for the frequentist MLE method. When applied to a `Bayes.fit` object, it computes means and credibility interval (default level 95%) for each parameter $(\mu, \Omega, \sigma, \alpha, \beta)$. Here, there is also a possibility to choose the burn-in and the thinning rate manually by the input parameters `burnIn` and `thinning`.

Command `print()` applied to a `Freq.fit` object returns the use or not of the cutoff and the vector of excluded trajectories. When applied to a `Bayes.fit` object, it returns the acceptance rates of the MCMC procedure.

Validation methods

Validation of a mixed model, obtained with function `valid`, is an individual validation. Indeed, the validation of estimation of trajectory number j is obtained comparing it to M new trajectories simulated with parameters (α, β) fixed to the estimator A_j (or $\widehat{A}_j$) in the frequentist approaches and to the posterior means in the Bayesian approach. Inputs of the function are

- `Freq.fit` or `Bayes.fit` object.
- `plot.valid` 1 to generate a figure (default value is 1).
- `numj` A specific individual trajectory to validate (default: randomly chosen between 1 and M).
- `Mrep` The number of simulated trajectories (default value 100).

Each observation $X_{\text{numj}}(t_k)$ is compared with the `Mrep` simulated values $(X^1_{\text{numj}}(t_k), \ldots, X^{M_{\text{rep}}}_{\text{numj}}(t_k))$, for $k = 1, \ldots, N$.

Outputs are the list of the $(X^1_{\text{numj}}(t_k), \ldots, X^{M_{\text{rep}}}_{\text{numj}}(t_k))$. If `plot.valid`=1, two plots are produced. Left: plot of the `Mrep` new trajectories (black) and the true trajectory number `numj` (in grey/red). Right: quantile-quantile plot of the quantiles of a uniform distribution and the N quantiles obtained comparing $X_{\text{numj}}(t_k)$ with the `Mrep` simulated values $(X^1_{\text{numj}}(t_k), \ldots, X^{M_{\text{rep}}}_{\text{numj}}(t_k))$, for $k = 1, \ldots, N$.

This is an empirical method. The recent work Kuelbs and Zinn (2015) on depth and quantile regions for stochastic processes (see for example Zuo and Serfling (2000) for depth functions definitions) should provide the theoretical context for a more extensive study. This could be done in further works.

Prediction methods

Prediction (see Section 8.2.4) is implemented in function `pred`. Main inputs of the function are

- `Freq.fit` or `Bayes.fit` object.
- `invariant` TRUE if the new trajectories are simulated according to the invariant distribution.
- `level` The level of the empiric prediction intervals (default 0.05).
- `plot.pred` TRUE to generate a figure (default TRUE).

(and optional plot parameters). Function `pred` applied to a `Freq.fit` object returns a list with predicted random effects `phipred`, predicted trajectories `Xpred` and indexes of the corresponding true trajectories `indexpred` (see Section 8.2.4 for details of simulation). If `plot.pred = TRUE` (default) three plots are produced. Left predicted random effects versus estimated random effects. Middle: true trajectories. Right predicted trajectories and their empirical 95% prediction intervals (default value `level=0.05`). The prediction can also be done from the truncated estimator $\widehat{\widehat{f}_h}$ based on the $\widehat{A}_j$ given by Equation 5, if the argument `pred.trunc = 1`.

Function `pred` applied to a `Bayes.fit` object returns a S4 class object `Bayes.pred`. The first element of this class is `Xpred`, which depends on the input parameters. Including the input `trajectories = TRUE`, matrix `Xpred` contains the M drawn trajectories by rows (see first method described for the Bayesian approach in Section 8.2.4). Default is `trajectories = FALSE` which leads to the calculation of the predictive distribution explained in Section 8.2.4. With the input `only.interval = TRUE` (default), only the quantiles for the 1- `level` prediction interval are calculated, stored in `qu.l` and `qu.u`. Input `only.interval = FALSE` provides additionally `Xpred` containing `sample.length` (default 500) samples from the predictive distribution in each time point of the observations (except the first). In both cases, with `plot.pred = TRUE`, two figures are produced. On the left side, the data trajectories are compared with the prediction intervals and on the right side, the coverage rate is depicted which is stored in entry `coverage.rate`, namely the amount of series covered by the prediction intervals for each time point. The last class entry `estim` stores the results from the `Bayes.fit` object in a list. Other input parameters are `burnIn` and `thinning` which allow for the choice of other burn-in phase and thinning rate than proposed in the `Bayes.fit` object.

For the `Bayes.pred` class object, two plot methods are available. `plot()` repeats the figures that are created with the `plot.pred = TRUE` command in the `pred` method. `plot2compare()` compares up to three `Bayes.pred` objects, where in a first figure the prediction intervals are presented in colors black, red and green and the observed data series in grey and in a second figure the corresponding coverage rates are compared. With the input parameter `names` a vector of characters to be written in a legend can be indicated.

Note that to avoid over-fitting, we recommend to use only 2/3 of the data for the estimation of the density f and the last third for the prediction.

Package mixedsde through simulated examples

In this part two simulated examples are given to illustrate the strengths of each proposed method. Two datasets are simulated according to:

1. CIR model with one non-Gaussian random effect $\beta_j \sim \Gamma(1.8, 0.8)$, $\alpha_j = 1$, $T = 50$, $M = 200$, $N = 1000$:

   ```
   R> model1 <- "CIR"; random1 <- 2; fixed1 <- 1; sigma1 <- 0.1 ; M1 <- 200;
   R> T1 <- 50; N1 <- 1000; X01 <- 1; density.phi1 <- "gamma";
   +  param1 <- c(1.8,0.8);

   R> simu1 <- mixedsde.sim(M = M1, T = T1, N = N1, model = model1,
   +  random =random1, fixed = fixed1, density.phi = density.phi1,
   +  param = param1, sigma = sigma1, X0 = X01)
   R> X1<- simu1$X; phi1 <- simu1$phi; times1 <-simu1$times
   ```

2. OU model with one Gaussian random effect $\alpha_j \sim \mathcal{N}(3, 0.5^2)$, $\beta_j = 5$, $T = 1$, $M = 50$, $N = 500$:

   ```
   R> model2 <- "OU"; random2 <- 1; sigma2 <- 0.1; fixed2 <- 5; M2 <- 50;
   +  T2 <- 1;N2 <- 500; X02 <- 0; density.phi2 <- "normal";
   +  param2 <- c(3, 0.5);
   R> simu2 <- mixedsde.sim(M = M2, T = T2, N = N2, model = model2,
   +  random  = random2, fixed = fixed2, density.phi = density.phi2,
   +  param = param2, sigma = sigma2, X0 = X02)
   R> X2 <- simu2$X; phi2 <- simu2$phi; times2 <- simu2$times
   ```

Example 1 has non Gaussian random effect, the nonparametric method is the most appropriate approach. Example 2 has T small and Gaussian random effect, nonparametric method is therefore not the most appropriate approach. Parametric methods should performed better than the non-parametric one as the number of trajectories $M2 = 50$ is not large (and only 2/3 are used for the estimation of f). A small number of trajectories is especially a good framework to apply the Bayesian estimation method.

Frequentist nonparametric estimation

We illustrate nonparametric estimation on Example 1. Code for the nonparametric estimation is

```
R> estim.method <- 'nonparam'
R> estim_nonparam <- mixedsde.fit(times = times1, X = X1, model = model1,
+  random = random1, fixed = fixed1, estim.method = estim.method)
R> outputsNP <- out(estim_nonparam)  # stores the results in a list
```

Summary function provides:

```
R> summary(estim_nonparam)
     [,1]    [,2]
[1,] "sigma" "0.099868"
```

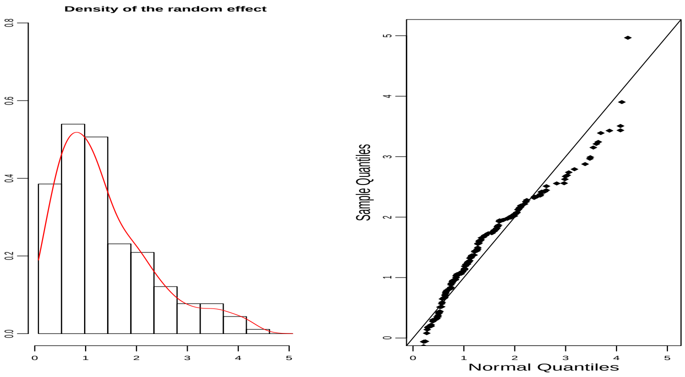


Figure 1: Simulated example 1 (CIR with one Gamma random effect), nonparametric estimation. Left: histogram of estimated random effects (A_j) and nonparametric estimation of f. Right: qqplot of (A_j) versus a Normal sample (true distribution is Gamma).

```
Random effect:
                    [,1]
empiric mean 1.355403
empiric sd   0.939410
kurtosis     3.695013
skewness     1.083577
```

As expected kurtosis is larger than 3 and skewness is positive which means that the distribution is right-tail. Figure 1 is provided by

```
R> plot(estim_nonparam)
```

Nonparametric estimation fits well the histogram of (A_j) (left plot) and we see that the random effects are non-Gaussian (right plot). Because we are working on simulated data, we can compare the estimations with the true random effects and the true f:

```
# Comparison of the true f and its estimation
R> gridf1 <- outputsNP$gridf
# True density function
R> f1 <- dgamma(gridf1, shape = param1[1], scale = param1[2])
# Nonparametric estimated density function
R> fhat <- outputsNP$estimf
R> plot(gridf1, f1, type='l', lwd=2, xlab='', ylab='')
R> lines(gridf1, fhat, col='red')
# Comparison of the true random effects and their estimations
# Estimated random effects
R> phihat1 <- outputsNP$estimphi
R> plot(phi1, phihat1, type = "p", pch = 18, xlab='', ylab='')
R> abline(0, 1)
```

This results in Figure 2. On the left plot, the estimated density (dotted curve) is very close to the true density f (plain line). The right plot shows that A_j is a good estimation of ϕ_j. This confirms that the nonparametric approach performs well for this settings. Validation of the MSDE is produced by function `valid`. The two graphs on the right of Figure 5 are obtained by

```
R> validationCIR <- valid(estim_nonparam)
```

Prediction are obtained with `pred` and similar Figure 6 (not shown) can be obtained with

```
R> predNPCIR <- pred(estim_nonparam)
```

Frequentist parametric estimation

We present the parametric estimation on Example 2. The code is

```
# Parametric estimation
R> estim.method<-'paramML';
R> estim_param <- mixedsde.fit(times2, X = X2, model = model2,
+  random = random2, estim.fix = 1, estim.method = 'paramML' )
```

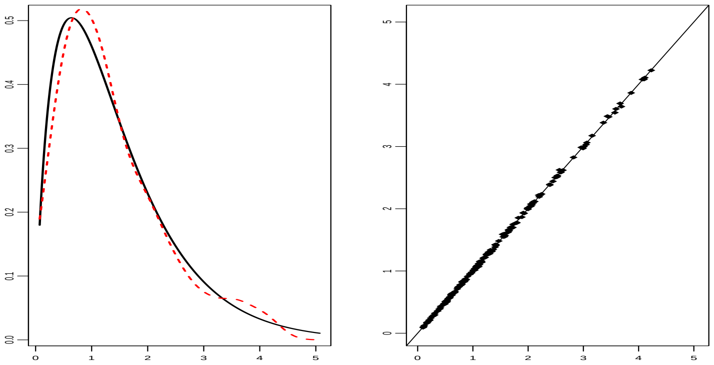


Figure 2: Simulated example 1 (CIR with one Gamma random effect), nonparametric estimation, comparison to the truth. Left: estimation $\widehat{f}$ (dotted line) and true density f (plain line). Right: Estimated random effects A_j versus true random effects ϕ_j.

```
# Store the results in a list:
R> outputsP <- out(estim_param)
```

Summary function provides:

```
R> summary(estim_param)
     [,1]    [,2]
[1,] "sigma" "0.109144"

Random and fixed effects:
                 [,1]
estim.fixed  4.914685
empiric mean 2.955582
MLE mean     2.955512
empiric sd   0.536956
MLE sd       0.519955
kurtosis     2.472399
skewness     0.427223
     [,1]  [,2]
[1,] "BIC" "-3780.383134"
[2,] "AIC" "-3795.335809"
```

Kurtosis is, as expected, close to 3 and skewness close to 0. The diffusion parameter σ is well estimated (true value 0.1). The fixed effect is also well estimated (true value 5). Empirical mean and standard deviations are very close to MLE (estimator of the mean is the same in that case) and close to the real ones (3, 0.5). Then, Figure 3 (left and right) is provided by

```
R> plot(estim_param)
```

The small number of observations makes the estimation harder, nevertheless here, the histogram seems pretty well fitted by the parametrically estimated density. Because we are working on simulated data, we can compare the estimations with the true random effects and the true f:

```
# Comparison of the true f and its estimation
R> gridf2 <- outputsP$gridf
# True density
R> f2 <- dnorm(gridf2, param2[1], param2[2])
# Parametric estimated density
R> fhat_param <- outputsP$estimf
R> plot(gridf2, f2, type = 'l', lwd = 2, xlab = '', ylab = '')
R> lines(gridf2, fhat_param, col='red', lty = 2, lwd = 2)
# Comparison of the true random effects and their estimations
# Estimated random effects
R> phihat2 <- outputsP$estimphi
R> plot(phi2, phihat2, type="p", pch=18, xlab='', ylab='')
R> abline(0, 1)
```

This results in Figure 4. It shows that estimation of the density is satisfactory (left) and estimation of the random effects is very good (right). Validation of the MSDE is produced by function `valid`.

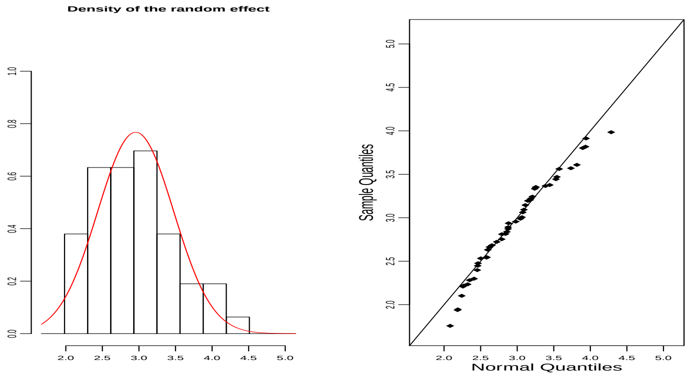


Figure 3: Simulated example 2 (OU with one Gaussian random effect) frequentist parametric estimation. Left: histogram of the (A_j) and Gaussian parametric estimation of f. Right parametric qqplot of (A_j) versus a Normal sample.

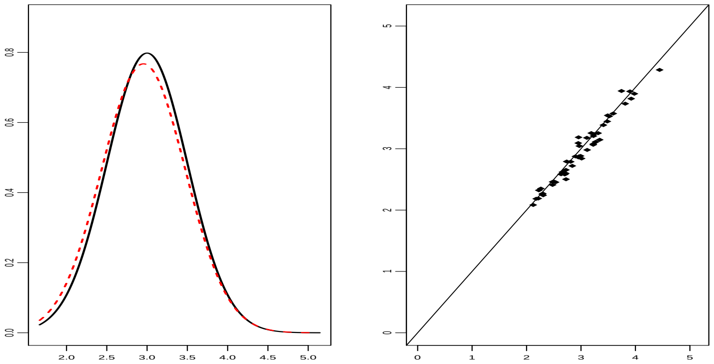


Figure 4: Simulated example 2 (OU with one Gaussian random effect) frequentist parametric estimation, comparison to the truth. Left: parametric estimation $\mathcal{N}(\widehat{\mu}, \widehat{\omega}^2)$ (dotted line) and true f (plain line). Right: true ϕ_j versus estimated random effects A_j.

For example the individual validation of the first trajectory is plotted Figure 5, the first two graphs on the left, using

```
R> validationOU <- valid(estim_param)
```

This illustrates the good estimation of the random effects: a beam of trajectories with the true one in the middle and the lining up of the quantiles.

Finally, we can predict some trajectories using `pred`. Predictions are shown on Figure 6, as a result of

```
R> predPOU <- pred(estim_param)
```

Beam of 32 predicted trajectories (right) is close to the true ones (middle). The lining up of the predicted random effects versus the estimated random effects (left) shows the goodness of the prediction from the estimated density, thus of the estimation of the density.

Bayesian estimation

Bayesian method is applied to Example 2. Priors are constructed from the true values, but default values can be used.

```
R> prior2 <- list( m = c(param2[1],  fixed2), v = c(param2[1], fixed2),
+  alpha.omega = 11, beta.omega = param2[2] ^ 2 * 10, alpha.sigma = 10,
+  beta.sigma = sigma2 ^ 2 * 9)
R> estim.method <- 'paramBayes'
R> estim_bayes <- mixedsde.fit(times = times2, X = X2, model = 'OU',
+  random = random2,  estim.method = estim.method, prior = prior2, nMCMC = 10000)
R> outputsBayes <- out(estim_bayes)
```

Figure 7 is produced by

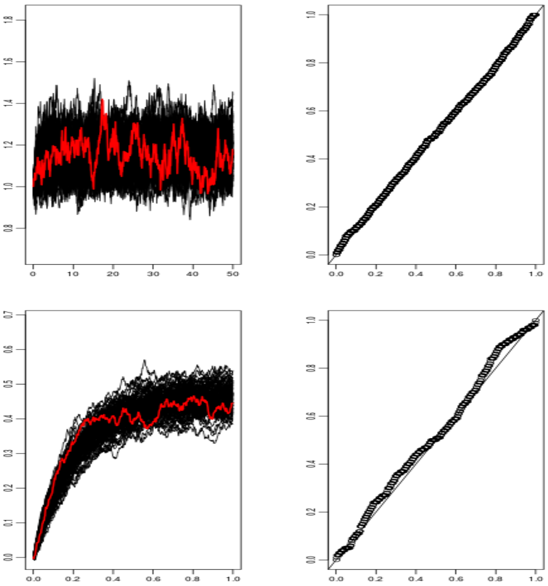


Figure 5: Simulated examples frequentist approaches, outputs of `valid` method. Two top plots: frequentist nonparametric estimation on example 1 (CIR process). Two bottom plots: frequentist parametric estimation on example 2 (OU process).

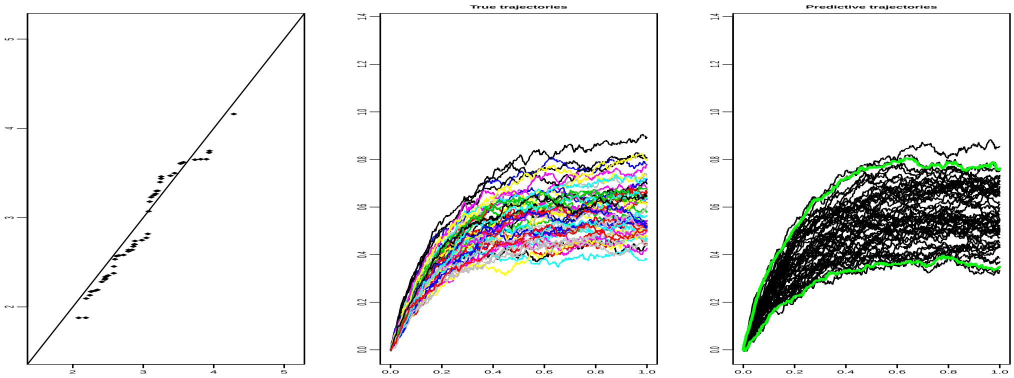


Figure 6: Simulated example 2 (OU with one Gaussian random effect), frequentist parametric estimation. Left: predicted random effects versus estimated random effects. Middle: true trajectories. Right: predicted trajectories in black and 95% prediction interval in grey (green).

```
R> plot(estim_bayes)
```

Traces of the Markov chains of μ_1, β, ω_1^2 and σ are plotted, showing that all chains converge and have the correct location. Command `print()` yields acceptance rates of the MH algorithm:

```
R> print(estim_bayes)

acceptance rates for random effect:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.5569  0.5646  0.5676  0.5682  0.5718  0.5805

acceptance rate for fixed effect: 0.4248
```

The fixed effect β has a small acceptance rate, explaining the dependent chain (Figure 7 top right). This is due to a very sharp likelihood because of the large amount of observations $(N \cdot M)$ in comparison to the random effect (N).

Predictions in the Bayesian framework and the corresponding Figure 8 is obtained by

```
R> pred.result <- pred(estim_bayes)
```

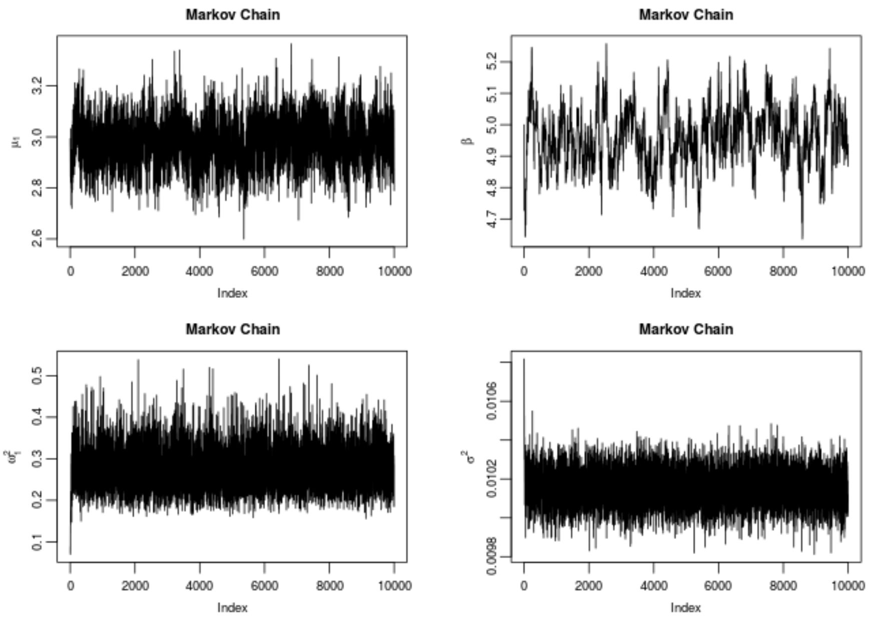


Figure 7: Simulated example 2 (OU with one Gaussian random effect) Bayesian estimation. Markov chains of μ_1, β, ω_1^2 and σ^2.

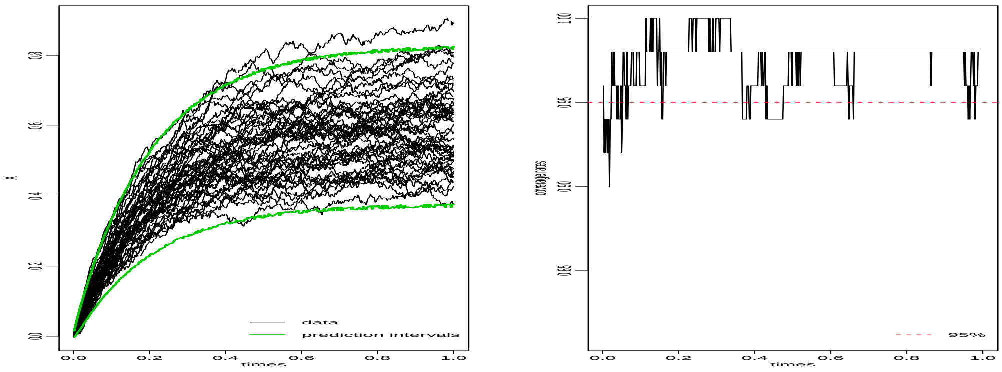


Figure 8: Simulated example 2 (OU with one Gaussian random effect) Bayesian estimation. Left: predicted trajectories in black and 95% prediction interval in grey (green). Right: coverage rates: amount of observed values covered by the prediction intervals.

Figure 8 shows the beam of simulated data trajectories together with the 95% prediction interval. Coverage rates are shown on the right plot and we see that the intervals hold the level.

Package mixedsde through a real data example

A real dataset is available (`neuronal.data.rda`) through lists of a matrix `X` and a vector `times`. We detail below the analysis of this dataset, following the next steps: run the two random effects model with both the parametric and nonparametric procedure; choose the number of random effects depending on the variability of the estimators $(A_{j,1}, A_{j,2})$, on the shape of $\widehat{f}_h$ and the variance $\widehat{\Omega}$.

These data are available thanks to Rune Berg and Jufang He. Details on data acquisition can be found in Lansky et al. (2006).

Neuronal data

Neurons are the basement of nervous system and each neuron is connected with around 1000 other neurons. They are communicating through emission of electrical signal. We focus on the dynamic of the neuron membrane potential between two spikes emission measured in volts as the difference of ions concentration between the exterior and the interior of the cell. Data are obtained from

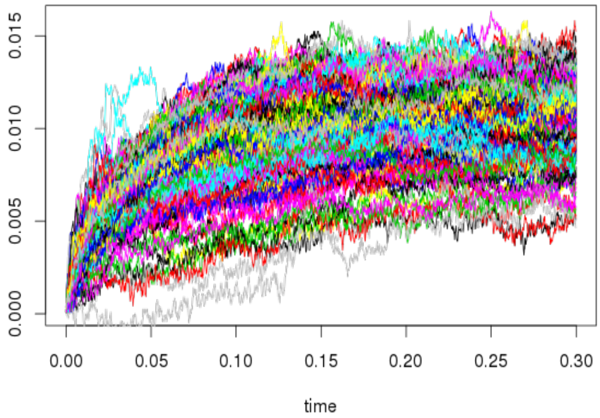


Figure 9: Neuronal data.

one single neuron of a pig. Data are composed of $M = 240$ membrane potential trajectories with $N = 2000$ equidistant observation times. Time step is $\delta = 0.00015$ [s] and observation time is $T = 0.3$ [s]. Data are uploaded using `data("neuronal.data")`. They are presented on Figure 9.

These data have been previously analysed with a Ornstein-Uhlenbeck model with one additive random effect (α_j): Picchini et al. (2008) and Picchini et al. (2010) use parametric methods assuming the normality of the random effect, and Dion (2014) with a nonparametric method. Here α_j represents the local average input that the neuron receives after the j^{th} spike. The initial voltage (the value following a spike) is assumed to be equal to the resting potential and set to zero: $x_j = 0$. Parameter β_j (non negative) is the time constant of the neuron. It was fixed in Picchini et al. (2008) and Picchini et al. (2010).

In this new analysis, we assume that both α_j and β_j may change from one trajectory to another because of other neurons or environment influence, for example. Indeed, the form of each trajectory lead us to think that this is the good model: each one has its mean-reverting value and its own speed to reach this value. There is no reason to assume that the speed is always the same, but looking at the trajectories the stationary state seems to be reached nearly at the same time, thus the second random effect should have a small variance.

Fitting with MSDEs

Our goal is also to compare the two models OU and CIR, both with two random effects, and two approaches: the nonparametric density estimation and the parametric density estimation. Let us remark that for the CIR model the algorithm removes two trajectories: 168 and 224, because they contain negatives values. For two random effects the command is

```
R> estim <- mixedsde.fit(times, X = X, model = model, random = c(1,2),
+  estim.method = estim.method)
```

and they can be found in the help data file (command `?neuronal.data`). We first apply the

two frequentist approaches on models with two random effects. Kurtosis and skewness of the distribution of the estimation A_j of the random effects given in Table 2 are not closed to a symmetric distribution. The bidimensional density of (α_j, β_j) is estimated for both models with the parametric and nonparametric methods running function `mixedsde.fit`. Figure 10 gives the 4 estimated marginals. The blue (black) is for the OU model and the green (grey) for the CIR model. The dotted lines are the estimations from the parametric method, the plain lines for the nonparametric estimation. Parametric and nonparametric estimators are close, except for the second random effect with the OU model. Indeed, parametric estimation produces a small variance for the second random effect, suggesting it could be fixed. Would this assumption be valid, it explains the difference with the nonparametric estimator which is not stable if the variance is to small. Estimation of σ is $\widehat{\sigma} = 0.0136$ for the OU model and $\widehat{\sigma} = 0.163$ for the CIR model.

To compare with previous literature results, we focus on the OU model. To select the number and the position of the random effects, we run the code with one random effect, additive or multiplicative:

	OU	CIR
Aj1 Kurtosis	6.17	11.70
Skewness	0.96	2.32
Aj2 Kurtosis	6.68	7.07
Skewness	0.96	2.32

Table 2: Neuronal data. Kurtosis and skewness estimations for samples $(A_{j,1})$'s and $(A_{j,2})$'s, for OU and CIR models.

`random = 1` or `random = 2`, for both models estimating the common fixed parameter with the parametric frequentist strategy. Estimators of the means μ_1, μ_2 and standard deviations ω_1, ω_2 are given in Table 3. Criteria AIC and BIC are also given in Table 3. From this table, we can see that models `random = 1` and `random = c(1,2)` are the best according to the BIC and AIC criteria.

Finally, in Table 4 we compare the BIC and AIC criteria for `random = 1` when the value of the fixed effect is plugged in: the one we obtained in Table 3 and to values obtained in Picchini et al. (2008) and Picchini et al. (2010). The preferred model is the one minimizing both criteria. Thus, the OU model with one additive random effect $\phi_j = \alpha_j$ and $\widehat{\beta} = 37.22$ seems to be the best model to describe these data. The summary method gives for the kurtosis: 4.55 and for the skewness -0.95. Also $\widehat{\sigma} = 0.0136$. Estimated densities obtained for this model with $\widehat{\beta} = 37.22$ are given in Figure 11. The dotted line is the result of the parametric estimation and the plain line of the nonparametric estimation, plotted on the histogram of the $A_j(T)$'s. The nonparametric estimation detects a left tail that is not detected by the parametric one. Otherwise both estimators are very close.

The OU model with `random = 1` is then validated with `valid` function. Figure 12 illustrates the result for a random trajectory (number 141): 100 simulated trajectories (black) and true trajectory (X_{141}, red) (left plot) and quantiles of the true measurement among the 100 simulated points at each time points versus uniform quantiles. The qq-plot is satisfactory (compared to the graph obtained on simulated data Figure 5).

Finally some prediction plots are performed (not shown) with the `pred` method and they confirm that model OU with `random = c(1,2)` with the parameters obtain from the parametric estimation, and the OU model with `random = 1` and $\widehat{\beta} = 37.22$ produce very close trajectories and could be both validated.

We then apply the Bayesian procedure. As already mentioned, for the Bayesian procedure, large data sets are a problem because of the very long running time. Therefore, we thin the data set by 10. That means, every 10th data point of the series is used for the estimation and also for the prediction. Even with this thinning, one estimation with 20000 samples takes half an hour.

Based on the best model selected by the frequentist approach, the OU model with one random effect $\phi_j = \alpha_j$ is fitted. No prior knowledge is available, we therefore leave this information out and let the algorithm take the first 10%, i.e. 24, series for the calculation of the prior parameter, as described in Section 8.3.2. Figure 13 plots the Markov chains estimated from the remaining $M - 24 = 216$ trajectories and show good convergence of the chains. Bayesian point estimations, i.e. posterior means, are $\widehat{\mu}_1 = 0.34$, $\widehat{\omega}_1 = \sqrt{\widehat{\omega}_1^2} = 0.06$, $\widehat{\beta} = 33$ and $\widehat{\sigma} = \sqrt{\widehat{\sigma}^2} = 0.01$. Compared to frequentist estimation (Table 4), we notice that these results are a compromise between Picchini et al. (2010) and frequentist estimation.

In Figure 14, we can see a comparison of the prediction results for all three cases, α, β or both being random effects. The black and the green lines are very similar, which means, that the prediction intervals are nearly the same for α and both parameters being random effects. This confirms the frequentist conclusion of Table 3. Therefore, it could be enough to treat α as random and β as fixed effect.

	μ_1	ω_1	μ_2	ω_2	BIC	AIC
`random=c(1,2)`	0.38	0.06	37.30	1.10	-3229.67	-3247.59
`random=2`	0.37	- -	37.70	7.47	-3082.36	-3103.40
`random=1`	0.38	0.06	37.22	- -	-3227.47	-3248.51

Table 3: Neuronal data. MLE given by Equation 7, BIC and AIC criteria, for OU model, depending on the number of random effects (with `estim.fix=1` for `random = 1` or `random = 2`).

	μ_1	ω_1	β	BIC	AIC
β from Picchini 2008	0.27	0.04	25.64	-2971.59	-2980.55
β from Picchini 2010	0.47	0.08	47.00	-3043.89	-3052.86
Previous estimator MLE of β Table 3	0.38	0.06	37.22	**-3240.55**	**-3249.51**

Table 4: Neuronal data. Results obtained with `random=1` for the OU model, where the value of the fixed effect β is plugged in.

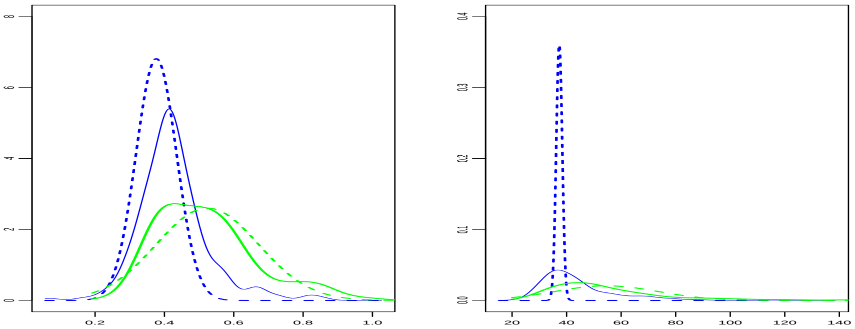


Figure 10: Neuronal data. Frequentist estimated marginals of the bidimensionnal density of the random effects obtained from 4 estimators. Left: α_j's density, right: β_j's density. CIR model in green (grey), OU in blue (black). Nonparametric in plain line, parametric in dotted line.

Discussion

In this paper we illustrate the functionality of the package **mixedsde** for inference of stochastic differential equations with random and/or fixed effects. This package, and mainly the function `misedsde.fit`, can be used to choose the best model to fit some data. It allows to compare two models: OU or CIR with one or two random effects. The three estimation methods can be used to help the decision maker. Nevertheless each method can be more appropriate to a specific situation, as explained before: the Bayesian method is recommended for a small number of observations, the frequentist nonparametric is a good tool with two random effects and no prior available. In particular the frequentist parametric proposes for a large sample, an estimation of the fixed effect and of the

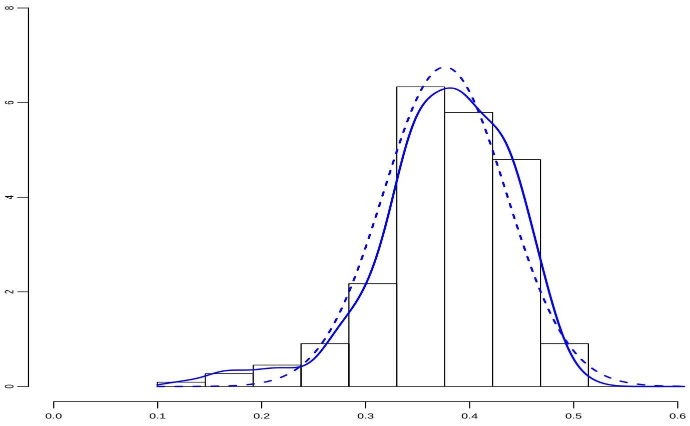


Figure 11: Neuronal data, OU model, α random, β fixed to the estimator obtained by the maximum likelihood estimator. Histogram of the A_j's estimators of the $\phi_j = \alpha_j$. Estimator of the density f: $\mathcal{N}(\mu, \omega^2)$ parametric estimation in blue (black) dotted line, non-parametric estimation blue (black) plain line.

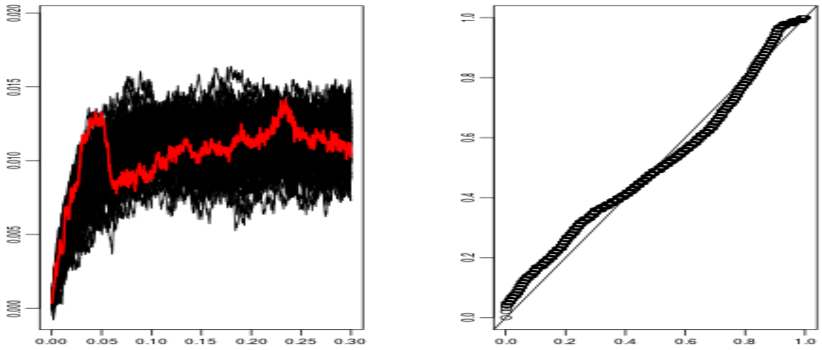


Figure 12: Neuronal data, OU model, α random, β fixed, validation of the frequentist approaches. Individual validation of trajectory 232. Left: 100 simulated trajectories in black and true trajectory (X_j) in grey (red). Right: quantiles of the true measurement among the 100 simulated points at each time points versus uniform quantiles.

parameters of the Gaussian distribution for the fixed effect when there is only one. A neuronal dataset is studied with the three methods. Furthermore, other real data should be investigated with the present package.

Recently, the parameter estimation method developed in Delattre et al. (2016) for random effects distributed according to a Gaussian mixture distribution has been implemented in the R package **MseParEst** (Delattre and Dion, 2016).

Acknowledgements

The authors would like to thank Vincent Brault and Laurent Bergé for technical help on the package. This work has been partially supported by the LabExPERSYVAL-Lab(ANR-11-LABX-0025-01).

The second author, Simone Hermann, was financially supported by Project B5 “Statistical methods for damage processes under cyclic load” of the Collaborative Research Center “Statistical modeling of nonlinear dynamic processes” (SFB 823) of the German Research Foundation (DFG).

Bibliography

P. Bacher and H. Madsen. Identifying Suitable Models for the Heat Dynamics of Buildings. *Energy and Buildings*, 43:1511–1522, 2011. [p44]

F. Comte, V. Genon-Catalot, and A. Samson. Nonparametric Estimation for Stochastic Differential

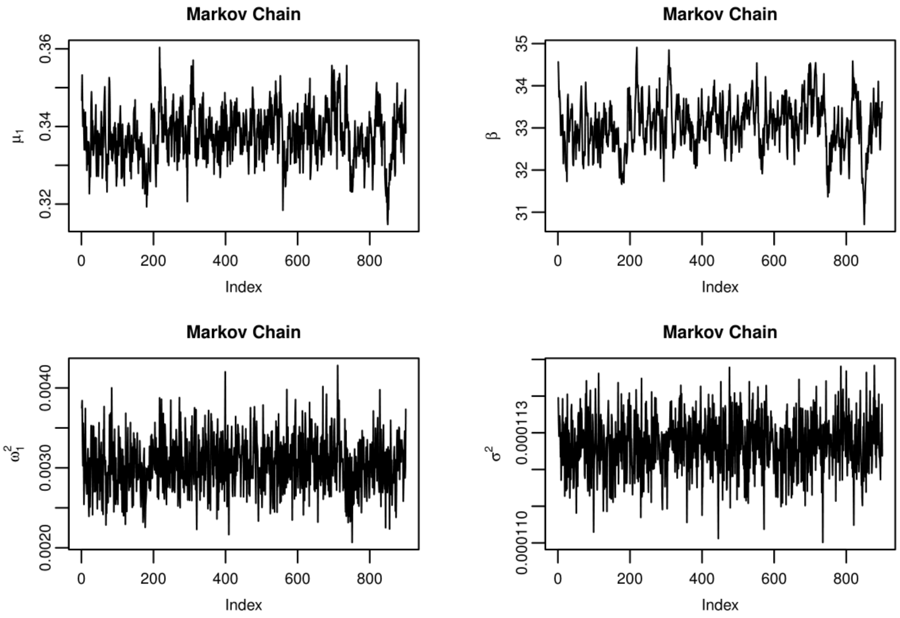


Figure 13: Neuronal data, OU model, α random, β fixed, Bayesian estimation. Reduced Markov chains (less the burn-in phase and the thinning rate).

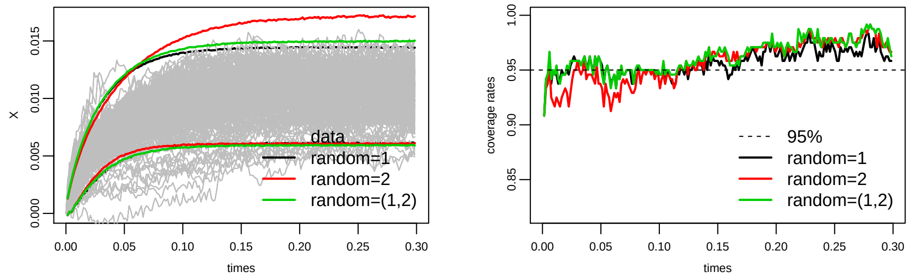


Figure 14: Neuronal data. Bayesian prediction results using the OU model, left: pointwise 95% prediction intervals and data series, right: coverage rates, which means the amount of observed values covered by the prediction intervals.

Equation with Random Effects. *Stochastic Processes and their Applications*, 7:2522–2551, 2013. [p45, 46]

M. Delattre and C. Dion. **MsdeParEst***: Parametric Estimation in Mixed-Effects Stochastic Differential Equations*, 2016. URL `https://CRAN.R-project.org/package=MsdeParEst`. R package version 1.7. [p62]

M. Delattre, V. Genon-Catalot, and A. Samson. Maximum Likelihood Estimation for Stochastic Differential Equations with Random Effects. *Scandinavian Journal of Statistics*, 40:322–343, 2013. [p45, 48]

M. Delattre, V. Genon-Catalot, and A. Samson. Mixture of Stochastic Differential Equations with Random Effects: Application to Data Clustering. *to appear in Journal of Statistical Planning and Inference*, 2016. [p45, 62]

C. Dion. Nonparametric Estimation in a Mixed-Effect Ornstein-Uhlenbeck Model. *Metrika*, 2014. [p45, 59]

C. Dion and V. Genon-Catalot. Bidimensional Random Effect Estimation in Mixed Stochastic Differential Model. *Statistical Inference for Stochastic Processes*, 18(3):1–28, 2015. [p45, 46, 48]

C. Dion, S. Hermann, and A. Samson. **mixedsde***: Mixed Stochastic Differential Equations*, 2016. URL `https://CRAN.R-project.org/package=mixedsde`. R package version 4.0. [p44]

S. Ditlevsen and A. de Gaetano. Mixed Effects in Stochastic Differential Equation. *REVSTAT - Statistical Journal*, 2:137–153, 2005. [p45]

S. Ditlevsen and A. Samson. Estimation in Partially Observed Stochastic Morris-Lecar Neuronal Model with Particle Filter and Stochastic Approximation Methods. *The annals of applied statistics*, 8:674–702, 2014. [p44]

V. Genon-Catalot and C. Larédo. Estimation for Stochastic Differential Equations with Mixed Effects. *Statistics*, pages 1–22, 2016. [p46]

A. C. Guidoum and K. Boukhetala. **Sim.DiffProc***: Simulation of Diffusion Processes*, 2017. URL https://CRAN.R-project.org/package=Sim.DiffProc. R package version 4.0. [p44]

A. Hansen, A. K. Duun-Henriksen, R. Juhl, S. Schmidt, K. Norgaard, J. B. Jorgensen, and H. Madsen. Predicting Plasma Glucose from Interstitial Glucose Observations Using Bayesian Methods. *Journal of diabetes science and technology*, 8:321–330, 2014. [p44]

S. Hermann, K. Ickstadt, and C. Müller. Bayesian Prediction of Crack Growth Based on a Hierarchical Diffusion Model. *to appear in: Applied Stochastic Models in Business and Industry*, 2016. [p44, 45, 48]

S. M. Iacus. **sde***: Stochastic Differential Equations*, 2006. URL http://CRAN.R-project.org/package=sde. R package version 2.0-15. [p49]

S. M. Iacus. *Simulation and Inference for Stochastic Differential Equations*. Springer-Verlag, 2008. [p48]

S. M. Iacus. **yuima***: The YUIMA Project Package for SDEs*, 2018. URL https://CRAN.R-project.org/package=yuima. R package version 1.8.1. [p44]

E. B. Iversen, J. M. Morales, J. K. Moller, and H. Madsen. Probabilistic Forecasts of Solar Irradiance Using Stochastic Differential Equations. *Environmetrics*, 25:152–164, 2014. [p44]

N. R. Kristensen and H. Madsen. Continuous time stochastic modelling, ctsm 2.3—mathematics guide. Technical report, DTU, 2003. [p44]

J. Kuelbs and J. Zinn. Limit Theorems for Quantile and Depth Regions for Stochastic Processes. *to appear in High dimensional probability VII-Progress in Probabiliy*, 2015. [p52]

P. Lansky, P. Sanda, and J. He. The Parameters of the Stochastic Leaky Integrate-and-Fire Neuronal Model. *Journal of Computational Neuroscience*, 21:211–223, 2006. [p58]

MONOLIX. *MONOLIX Software. MOdèles NOn LInéaires à Effets miXtes*, 2003. URL http://www.lixoft.com/. LIXOFT and INRIA. [p45]

S. B. Mortensen and S. Klim. **PSM***: Population Stochastic Modelling*, 2013. URL https://CRAN.R-project.org/package=PSM. R package version 0.8-10. [p45]

Z. Oravecz, F. Tuerlinckx, and J. Vandekerckhove. A Hierarchical Ornstein-Uhlenbeck Model for Continuous Repeated Measurement Data. *Psychometrica*, 74:395–418, 2009. [p45]

U. Picchini, S. Ditlevsen, A. De Gaetano, and P. Lansky. Parameters of the Diffusion Leaky Integrate-and-Fire Neuronal Model for a Slowly Fluctuating Signal. *Neural Computation*, 20: 2696–2714, 2008. [p59, 60]

U. Picchini, A. De Gaetano, and S. Ditlevsen. Stochastic Differential Mixed-Effects Models. *Scandinavian Journal of statistics*, 37(1):67–90, 2010. [p45, 59, 60]

C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag, 2004. [p48]

J. S. Rosenthal. Optimal Proposal Distributions and Adaptive MCMC. *Handbook of Markov Chain Monte Carlo*, pages 93–112, 2011. [p48, 49]

W. N. Venables and B. D. Ripley. **MASS***: Modern Applied Statistics with S*, 2016. URL https://CRAN.R-project.org/package=MASS. R package version 7.3-45. [p46]

Y. Zuo and R. Serfling. General Notions of Statistical Depth Function. *Annals of statistics*, 28: 461–482, 2000. [p52]

Appendix

When there is one random effect, what is the likelihood function and the MLE of the fixed effect?

Assume that we are in the case of `random = 1`, thus the process is

$$dX_j(t) = (\alpha - \phi_j X_j(t))dt + \sigma a(X_j(t))dW_j(t).$$

Let us compute the log-likelihood function when $\phi_j = \varphi$ fixed. We omit the subscript j in the following. We use the same notation as for `random=c(1,2)`, where $V = (V_{k,\ell})_{k,\ell\in\{1,2\}}$ is a symmetric matrix size 2×2. We have :

$$\begin{aligned}\log L(X, \alpha, \varphi) &= \int_0^T \frac{b(X(s), \varphi)}{\sigma^2(X(s))} dX(s) - \frac{1}{2}\int_0^T \frac{b^2(X(s), \varphi)}{\sigma^2(X(s))} ds \\ &= \alpha U_1 - \frac{\alpha^2}{2} V_{1,1} + \varphi[U_2 - \alpha V_{1,2}] - \frac{\varphi^2}{2} V_{2,2}.\end{aligned}$$

We assume that the random effect is Gaussian with density f_ξ, and denote $\xi = (\mu, \omega)$ and $\theta := (\mu, \omega, \alpha)$. Thus,

$$L(X, \theta) = \int \exp\left(\alpha U_1 - \frac{\alpha^2}{2} V_{1,1} + \varphi[U_2 - \alpha V_{1,2}] - \frac{\varphi^2}{2} V_{2,2}\right) f_\xi(\varphi) d\varphi = \int \exp(\mathcal{E}(\varphi)) d\varphi.$$

We find:

$$\begin{aligned}\mathcal{E}(\varphi) &= \alpha U_1 - \frac{\alpha^2}{2} V_{1,1} - \frac{1}{2}\left[\varphi^2(V_{2,2} + \omega^{-2}) - 2\varphi(U_2 - \alpha V_{1,2} + \mu\omega^{-2})\right] \\ &= \alpha U_1 - \frac{\alpha^2}{2} V_{1,1} - \frac{1}{2\Sigma^2}(\varphi - m)^2 + \frac{m^2}{2\Sigma^2} - \frac{1}{2}\mu^2\omega^{-2}\end{aligned}$$

with

$$m = \frac{\mu + \omega^2 U_2 - \omega^2 V_{1,2}\alpha}{1 + \omega^2 V_{2,2}}, \quad \Sigma^2 = \frac{\omega^2}{1 + \omega^2 V_{2,2}}.$$

Finally after simplification we get:

$$\frac{m^2}{2\Sigma^2} - \frac{1}{2}\mu^2\omega^{-2} = -\frac{1}{2}(1 + \omega^2 V_{2,2})^{-1} V_{2,2}[\mu - V_{2,2}^{-1}(U_2 - \alpha V_{1,2})]^2 + \frac{1}{2V_{2,2}}(U_2 - \alpha V_{1,2})^2.$$

Thus for `random=1` we get

$$L(X, \theta) = \frac{1}{\sqrt{1 + \omega^2 V_{2,2}}} \exp\left[\alpha U_1 - \frac{\alpha^2}{2} V_{1,1} - \frac{V_{2,2}}{2(1 + \omega^2 V_{2,2})}[\mu - V_{2,2}^{-1}(U_2 - \alpha V_{1,2})]^2 + \frac{(U_2 - \alpha V_{1,2})^2}{2V_{2,2}}\right].$$

Then, when `random = 2` the roles of α and φ are exchanged. To implement a general formula, we note: r for random: 1 or 2, and c for the number of the common effect. We denote ψ the fixed effect and we the get the general formula:

$$L(X, \theta) = \frac{1}{\sqrt{1 + \omega^2 V_{r,r}}} \exp\left[\psi U_c - \frac{\psi^2}{2} V_{c,c} - \frac{V_{r,r}}{2(1 + \omega^2 V_{r,r})}[\mu - V_{r,r}^{-1}(U_r - \psi V_{c,r})]^2 + \frac{(U_r - \psi V_{r,c})^2}{2V_{r,r}}\right].$$

Charlotte Dion
Laboratory Jean Kuntzmann
Université Grenoble Alpes
Batiment IMAG
700 avenue Centrale
Campus de Saint Martin d'Hères BP 53
38041 Grenoble cedex 09 - France
and
Laboratory MAP5
Université Paris Descartes 45, rue des Saint-Pères
75005 Paris
and
Laboratory LSTA
Université Pierre et Marie Curie 4, place Jussieu
75005 Paris

charlotte.dion@upmc.fr

Simone Hermann
Mathematik Raum 744
Anschrift: Fakultät Statistik
Technische Universität Dortmund
44221 Dortmund
hermann@statistik.tu-dortmund.de

Adeline Samson
Laboratory Jean Kuntzmann
Université Grenoble Alpes
Batiment IMAG
700 avenue Centrale
Campus de Saint Martin d'Hères BP 53
38041 Grenoble cedex 09 - France
adeline.leclercq-samson@univ-grenoble-alpes.fr

Indoor Positioning and Fingerprinting: The R Package ipft

by Emilio Sansano, Raúl Montoliu, Óscar Belmonte and Joaquín Torres-Sospedra

Abstract Methods based on Received Signal Strength Indicator (RSSI) fingerprinting are in the forefront among several techniques being proposed for indoor positioning. This paper introduces the R package **ipft**, which provides algorithms and utility functions for indoor positioning using fingerprinting techniques. These functions are designed for manipulation of RSSI fingerprint data sets, estimation of positions, comparison of the performance of different positioning models, and graphical visualization of data. Well-known machine learning algorithms are implemented in this package to perform analysis and estimations over RSSI data sets. The paper provides a description of these algorithms and functions, as well as examples of its use with real data. The **ipft** package provides a base that we hope to grow into a comprehensive library of fingerprinting-based indoor positioning methodologies.

Introduction

Intelligent spaces, as a particularity of the concept known as Ambient Intelligence (AmI) (Aarts and Wichert, 2009; Werner et al., 2005), where agents communicate and use technology in a non-intrusive way, have an interest in both open and closed environments. Since people spend 90% of time indoors (Klepeis et al., 2001), one of the most relevant aspects of AmI is indoor localization, due to the large number of potential applications: industrial and hospital applications, passenger transport, residences, assistance to emergency services and rescue, localization and support guide for the disabled, leisure applications, etc. It is expected that the global market for this type of location will grow from USD 7.11 billion in 2017 to USD 40.99 billion by 2022 (Research and markets, 2017), being among the key technologies in the future. This is a technology that has already awakened but that in a short period of time will suffer a big explosion, as happened with the systems of positioning by satellite in exteriors and its applications.

This paper introduces the R package **ipft** (Sansano, 2017), a collection of algorithms and utility functions to create models, make estimations, analyze and manipulate RSSI fingerprint data sets for indoor positioning. Given the abundance of potential applications for indoor positioning, the package may have a broad relevance in fields such as pervasive computing, Internet of Things (IoT) or healthcare, among many others.

The main progress in indoor location systems has been made during the last years. Therefore, both the research and commercial products in this area are new, and researchers and industry are currently involved in the investigation, development and improvement of these systems. We believe that the R language is a good environment for machine learning and data analysis related research, as its popularity is constantly growing [1], researchers related to indoor positioning have explicitly selected R as developing framework for their experiments (Quan et al., 2017; Harbicht et al., 2017; Popleteev et al., 2011), it is well maintained by an active community, and provides an ecosystem of good-quality packages that leverage its potential to become a standard programming platform for researchers. There are some open source applications and frameworks to build indoor positioning services, such as FIND [2], Anyplace [3] or RedPIN [4], based on fingerprinting techniques but, as far as we know, there is not any public framework or package that provides functions and algorithms to manipulate fingerprinting datasets and experiment with positioning algorithms.

RSSI (Received Signal Strength Indicator) positioning systems are based on measuring the intensities of the received radio signals of the emitting devices (beacons) that are available at a particular position, and comparing them with a previously built RSSI data set (yub Lee et al., 2013). RSSI is used to measure the relative quality of a received signal to a client device, and each chipset manufacturer is free to define their own scale for this term. The value read by a device is given on a logarithmic scale and can correspond to an instant reading or a mean of some consecutive readings.

In this scenario, a fingerprint is an RSSI feature vector composed of received signal values from different emitting devices or beacons, associated to a precise position. In the last years, this technique is becoming increasingly important for indoor localization (Liu et al., 2007; He and Chan, 2016), since Wi-Fi is generally available in indoor environments where GPS signals cannot penetrate,

[1] https://stackoverflow.blog/2017/10/10/impressive-growth-r/
[2] https://www.internalpositioning.com#about
[3] https://anyplace.cs.ucy.ac.cy
[4] http://redpin.org

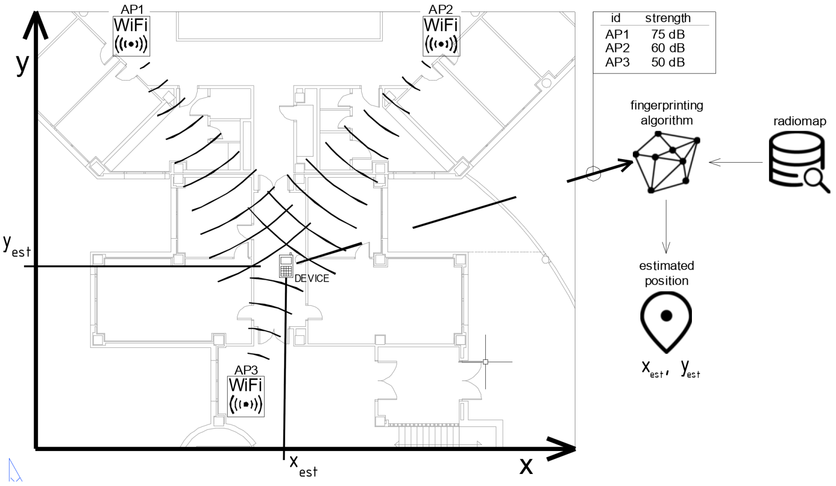


Figure 1: During the on-line phase, once the radio map has been built, the fingerprinting algorithm uses it to estimate the device's position by comparing the RSSI values heard by the device with the ones stored in the radio map.

and the wireless access points (WAPs) can be used as emitting devices (Li et al., 2006). Other types of indoor localization RF emitters, such as Bluetooth (Wang et al., 2013), RFID (Liu et al., 2011), or Ultra Wide Band (UWB) (Gigl et al., 2007), can be also used in combination with Wi-Fi access points or as a standalone positioning system.

The RSSI fingerprinting localization approach requires two phases of operation: a training phase, also known as off-line or survey phase, and a positioning phase, sometimes referred as on-line, runtime or tracking phase. In the training phase, multidimensional vectors of RSSI values (the fingerprints) are generated and associated with known locations. These measurements are used to build a data set (also known as radio map) that covers the area of interest. This data set can include, along with the collected RSSI values and the location coordinates, many other useful parameters, as the device type used in the measurements or its orientation. Later, during the positioning phase, an RSSI vector collected by a device is compared with the stored data to generate an estimation of its position (Figure 1).

Despite the increasing interest in RSSI positioning (Xiao et al., 2016), this topic has not been explicitly covered yet by any publicly available R package. The proposed package has been developed to provide users with a collection of fundamental algorithms and tools to manipulate RSSI radio maps and perform fingerprinting analysis. While fundamental algorithms and similarity measurement functions are implemented to provide a main framework for research and comparison purposes, these are highly customizable, to allow researchers to tailor those methods with their own parameters and functions.

This paper describes these algorithms and their implementation, and provides examples of how to use them. The remainder of the paper is structured as follows: Section Problem statement. Terminology and notation defines the fingerprinting problem statement and the nomenclature that will be used in the rest of the paper. An overview of the implemented algorithms is given in Section An overview of the implemented algorithms. Section Data wrangling outlines some data wrangling techniques included in the package. Section Positioning algorithms describes the implemented positioning algorithms. Section Beacon position estimation presents the included methods for access point position estimation. Then, Section Data clustering discuses some tools and functions included to create clusters or groups of fingerprints. Section Plotting functions illustrates the use of the plotting functions also included in the package. In all these sections, functions are described and explored using practical examples, and particular emphasis is placed on how to use them with real world examples and data sets. Finally, the paper is summarized in Section Summary.

Problem statement. Terminology and notation

This section provides a brief and general introduction to the principles of fingerprinting positioning, as well as a description of the notation and terminology that will be used in the next sections. The terms described here are related to general concepts of fingerprinting techniques, while the remaining of the paper describes the particular implementation of these concepts in the `ipft` package.

The main goal of the indoor localization techniques is to determine the position of a user in an indoor environment, where the GPS signal might not be received. This objective might require the use of an existing infrastructure, the deployment of a new one, the use of the so-called signals-of-opportunity (Yang et al., 2014), or even a combination of some of these techniques. Many of these techniques take advantage of the radio-frequency signals emitted by devices, whose position can be known or not, to estimate the user's position from the perceived strength of these signals. There are many kinds of devices that can be used for this purpose, such as Wi-Fi access points, bluetooth beacons, RFID or UWB devices, but for all of them, the information provided for a given position, the fingerprint, can be stored as a vector of received signal strength intensities (RSSI), whose length is determined by the number of detected emitters.

A radio map, or a fingerprinting data set, is composed of a set of collected fingerprints and the associated positions where the measurements were taken, and may contain some additional variables, such as the the type of device used or a time stamp of the observation, among any other useful data. Let $\mathcal{D}$ be a fingerprinting data set. Then:

$$\mathcal{D} = \{\mathcal{F}, \mathcal{L}\}$$

where $\mathcal{F}$ is the set of collected fingerprints and $\mathcal{L}$ is the set of associated locations.

For research purposes, a fingerprinting data set is usually divided into training and test sets. The training data set is used to store the fingerprints and location data to create models of the environment that can be used to estimate the position of a new fingerprint. The test data set is used to test the models obtained from the training data, and to compute the errors from the results of the position estimation.

Let $\mathcal{D}_{train}$ be a training data set:

$$\mathcal{D}_{train} = \{\mathcal{F}_{train}, \mathcal{L}_{train}\}$$

where

$$\mathcal{F}_{train} = \left\{\lambda_1^{tr}, \lambda_2^{tr}, ..., \lambda_n^{tr}\right\}$$

$$\mathcal{L}_{train} = \left\{\tau_1^{tr}, \tau_2^{tr}, ..., \tau_n^{tr}\right\}$$

$\mathcal{D}_{train}$ is composed of n fingerprints, stored as n vectors of RSSI measurements ($\lambda_i^{tr},\ i \in [1, 2, ..., n]$), and n locations ($\tau_i^{tr},\ i \in [1, 2, ..., n]$), stored as vectors, representing the position associated with its correspondent fingerprint. Each fingerprint consists of q RSSI values ($\rho_{h,i}^{tr},\ h \in [1, ..., q]$), where q is the number of beacons considered when building the training set:

$$\lambda_i^{tr} = \left\{\rho_{1,i}^{tr}, \rho_{2,i}^{tr}, ..., \rho_{q,i}^{tr}\right\},\ i \in [1, ..., n]$$

and each associated position is composed of one or more values, depending on the number of dimensions to be considered and the coordinate system used. The position can be given as a vector of values representing its coordinates, although on multi-floor and multi-building environments labels can be used to represent buildings, floors, offices, etc. Let l be the number of dimensions of a position vector. Then:

$$\tau_i^{tr} = \left\{\nu_{1,i}^{tr}, \nu_{2,i}^{tr}, ..., \nu_{l,i}^{tr}\right\},\ i \in [1, ..., n]$$

The test data set is also composed of a collection of fingerprints associated to known positions. This data set is used for testing purposes, during research or during model building adjustments, to assess the model's performance by comparing its estimation of the positions with the ground truth.

The situation is different in real applications, where the goal is to estimate the unknown position of the receiver given the RSSI values detected at a particular location, using a previously built model. In this case, the test data set is just composed of a unique fingerprint, and the objective is to estimate the actual location of the receiver. Therefore, no information about its location is provided.

The test data set is composed of m observations:

$$\mathcal{D}_{test} = \{\mathcal{F}_{test}, \mathcal{L}_{test}\}$$

where

$$\mathcal{F}_{test} = \left\{\lambda_1^{ts}, \lambda_2^{ts}, ..., \lambda_m^{ts}\right\}$$

$$\mathcal{L}_{test} = \left\{ \tau_1^{ts}, \tau_2^{ts}, ..., \tau_m^{ts} \right\}$$

To be able to compare the test observations with the training fingerprints, the number of RSSI values of its respective fingerprints has to be the same, and the position in the RSSI vector must represent the same beacon in both data sets. Therefore, each one of the m observations of the test data set is composed of a fingerprint with q RSSI values:

$$\lambda_j^{ts} = \left\{ \rho_{1,j}^{ts}, \rho_{2,j}^{ts}, ..., \rho_{q,j}^{ts} \right\}, \; j \in [1, ..., m]$$

and a location vector with the same spatial dimensions as the training location vectors:

$$\tau_j^{ts} = \left\{ \nu_{1,j}^{ts}, \nu_{2,j}^{ts}, ..., \nu_{l,j}^{ts} \right\}, \; j \in [1, ..., m]$$

The notation depicted above will be used in the remaining of the paper to represent the fingerprinting data. Symbols i and j will be used to represent iterations over the training and test data sets, respectively, while h will be used to iterate over the beacons present in each fingerprint.

An overview of the implemented algorithms

This section presents an introduction to the main functions, included in the **ipft**[5] package, that implement fingerprinting-based indoor localization methods. The package also provides two data sets for training and validation purposes that are briefly described in this section.

The `ipft` package implements three algorithms to build models to estimate the position of a receiver in an indoor environment. Two of these implementations are based on the well known k-Nearest Neighbors algorithm (*knn*) (Cover and Hart, 1967) to, given an RSSI vector, select the k most similar training examples from the radio map. The similarity between the RSSI value vectors can be measured, for example, as the *euclidean* distance between them, but other distance functions may be used (Torres-Sospedra et al., 2015b). The selection of a method to compute this measure can be provided to the function in two ways, either choosing one of the already implemented distance measurements (*euclidean*, *manhattan*, etc.), or by way of a reference to a function implemented by the user that returns the distance (the lower, the more similar or 'closer') between two matrices or vectors. Once the k neighbors are selected, the location of the user is estimated as the weighted average of the neighbors positions.

The first implementation, corresponding to the function `ipfKnn`, may behave in a deterministic way, finding the k more similar neighbors using a deterministic similarity function such as the euclidean or manhattan distances, or in a probabilistic way, using similarity functions such as LDG (Logarithmic Gaussian Distance) or PLGD (Penalized Logarithmic Gaussian Distance) (Cramariuc et al., 2016b), that are based upon statistical assumptions on the RSSI measurement error. The similarity function can be chosen from the set of implemented options or provided by the user via a custom function. This implementation is discussed in the Section The `ipfKnn` function.

The other implementation of the knn algorithm assumes a probabilistic nature for the received signal distribution (Roos et al., 2002) and uses collections of many fingerprints at each particular position, acquired during the training phase. Therefore, the radio map is composed of several groups, where a group is a set of fingerprints (vectors of RSSI values) that share the same location. Assuming that the RSSI value for a specific beacon can be modeled as a random variable following a normal distribution (Haeberlen et al., 2004), any of these collections, or groups, of fingerprints can be represented by the statistical parameters of this distribution, in this case, the mean and the standard deviation. This implies that the original data set can be transformed into a new type of data structure by storing the mean and the standard deviation of every detected beacon for every group. All the original data for a group is transformed into two vectors, one storing the means and the other the standard deviations. The trustworthiness of the data in the new data set will depend on the number of measurements for every location of the original data. It is assumed that the more measurements for a particular location, the more reliable will be their inferred statistical parameters.

The implementation of this probabilistic-based method takes the original radio map and a set of group indices, and fits these groups of measurements to a normal (Gaussian) distribution for every beacon and every location, so that the signal intensity distribution is determined by the

[5] The **ipft** package is available at CRAN and can be installed as any other R package:

```
> install.packages("ipft")
```

The package has to be loaded into the main environment to use it for the first time in an R session:

```
> library("ipft")
```

mean and the standard deviation of the Gaussian fit. Then, given a test fingerprint, the algorithm estimates its position by selecting the k most probable locations, making explicit use of the statistical parameters of the data stored in the radio map to optimize the probabilities in the assignment of the estimated position by computing a similarity function based on a summatory of probabilities. This approach is implemented through the `ipfProbabilistic` function and is described in the Section The `ipfProbabilistic` function.

Finally, the third implemented algorithm is based on a scenario where the location of the beacons is known, and an estimation of the fingerprint position can be made using the log-distance path loss model (Seybold, J.S., 2005). The strength of the received signal at a particular point can be modeled as a function of the logarithmic distance between the receiver and the emitter and some parameters related to the environment properties and the devices characteristics. Therefore, as this method uses an analytical model to evaluate the position, no radio map is needed to train a model to compare fingerprints with, since the position might be estimated from the fingerprint data and the position of the beacons. This method is implemented by the function `ipfProximity` and is described in Section The `ipfProximity` function.

The previous functions `ipfKnn`, `ipfProbabilistic` and `ipfProximity` create models based on the training data and parameters provided. These models can then be evaluated using the `ipfEstimate` function, that internally detects the algorithm to apply based on the model that receives as parameter.

The package also includes data from the IPIN2016[6] Tutorial data set. In the `ipftrain` data frame there are $n = 927$ observations, including the RSSI values for $q = 168$ wireless access points, the location, expressed in Cartesian coordinates, for the observation (x, y), and some other variables, as timestamps for the measurements or an identifier for the user who took the survey. The `ipftest` data frame contains $m = 702$ observations with the same structure, for testing and validation purposes. The fingerprints included in both data sets where taken in the same building and the same floor. The `ipfpwap` data frame contains the position of 39 of the WAPs included in the `ipftrain` and `ipftest` data sets. The unknown positions of the remaining WAPs are stored as `NA`. The characteristics of these data sets attributes are:

- `RSSI values`: Columns from 1 to 168. The values represent the strength of the received signal expressed in decibels, on a scale that ranges from -30dBm to -97dBm in the training set, and from -31dBm to -99dBm in the test set. The closer the value to zero, the stronger the signal.
- `position`: Columns 169 (X) and 170 (Y). The position given in Cartesian coordinates, with its origin in the same corridor where the data was acquired.
- `user id`: A numeric value from 1 to 8 to represent each of the 8 users that acquired the train data set. The test dataset was acquired by a different user, represented by the value 0.
- `timestamp`: The UNIX time stamp of the observation, in seconds.

There are some other publicly available indoor location data sets that have been used to develop and test this package and that are not included for size reasons, as the UJIIndoorLoc Data Set (Torres-Sospedra et al., 2015a) or the Tampere University data set (Cramariuc et al., 2016a).

The theoretical foundations of the algorithms and its uses are discussed in detail in Section Positioning algorithms. A description of the functions `ipfKnn`, `ipfProximity`, `ipfProbabilistic` and `ipfEstimate` is given while presenting some simulations to show how these algorithms can be useful in practice.

Data wrangling

An RSSI fingerprint is a vector composed of signal strength measurements from all the emitters received by a client device at a particular point, and can be measured in any unit of power. It is often expressed in decibels (dBm), or as percentage values between 1-100, and can be a negative or a positive value. Typically this values are stored as negative figures, where the strongest signals are closer to zero.

Some algorithms are sensitive to the scale of the data. For example, Neural Networks generally work better (**?**) with data scaled to a range between [0, 1] or [−1, 1], since unscaled data may slow down the learning process and the convergence of the network parameters and, in some cases, prevent the network from effectively learning the problem. Thus, the first step before the data can be fed to a positioning algorithm may involve some kind of transformation, depending on the characteristics of the original data.

[6] http://www3.uah.es/ipin2016/

The data sets included in this package represent the RSSI data from a set of wireless access points as negative integer numbers from -99 (weakest detected signal) to -30 (strongest detected signal). When the RSSI of a WAP is not available, the value used is `NA`. This convention may be inconvenient for some calculations. For example, a similarity measure between two fingerprints as the euclidean distance will only take into account those WAPs that have been detected in both observations, causing a loss of information that otherwise could be utilized.

The ipft package contains some functions to manipulate and wrangle raw fingerprint data. The `ipfTransform` function mutates the given fingerprint data into a new data set with a specified range for the RSSI signals. The signature of the function is:

```
ipfTransform <- function(data, outRange = c(0, 1), outNoRSSI = 0, inRange = NULL,
                         inNoRSSI = 0, trans = "scale", alpha = 24)
```

where:

- `data`: The input data set with the original RSSI fingerprints.
- `outRange`: A numeric vector with two values indicating the desired range of the output data.
- `outNoRSSI`: The desired value for not detected beacons in the output data.
- `inRange`: A numeric vector with two values indicating the range of signal strength values in the input data. If this parameter is not provided, the function will infer it from the provided data.
- `inNoRSSI`: The value given to a not detected beacon in the original data.
- `trans`: The transformation to perform over the RSSI data, either 'scale' or 'exponential'.
- `alpha`: The α parameter for the exponential transformation.

The *scale* transformation scales the input data values to a range specified by the user. The feature scaling is performed according to Equation 1:

$$\rho_{h,i}^{out} = \begin{cases} a + b \cdot \rho_{h,i}^{in}, & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ outNoRSSI, & otherwise \end{cases} \tag{1}$$

$$b = \frac{outMin - outMax}{inMin - inMax}$$
$$a = outMin - inMin \cdot b$$

where:

- $\rho_{h,i}^{out}$ and $\rho_{h,i}^{in}$ are the output and input RSSI values, respectively, for the h^{th} beacon from the i^{th} observation
- $outMax$ and $outMin$ are the maximum and minimum values, respectively, specified for the output by the `outRange` parameter.
- $inMax$ and $inMin$ are the maximum and minimum values, respectively, of the input data.
- $outNoRSSI$ and $inNoRSSI$ are the values assigned in the fingerprint to represent a not detected beacon for the output and input data, respectively, specified by the parameters `outNoRSSI` and `inNoRSSI`.

The *exponential* transformation (Torres-Sospedra et al., 2015b) changes the data according to the next equation:

$$\rho_{h,i}^{out} = \begin{cases} \exp(\frac{pos(\rho_{h,i}^{in})}{\alpha}), & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ outNoRSSI, & otherwise \end{cases}$$

$$pos(\rho_{h,i}^{in}) = \begin{cases} \rho_{h,i}^{in} - inMin, & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ 0, & otherwise \end{cases}$$

where α is a parameter for the exponential transformation. The authors establish α as a case-based parameter, and find that 24 is a good value for RSSI fingerprinting data, but they did not study the effects of α in the transformed data.

The following code scales the `ipftrain` and `ipftest` data sets RSSI data, stored in the columns 1:168, to a positive range of values, from 0 to 1, with NA representing a not detected WAP. As a not detected WAP is represented by a `NA` value in the original data, this has to be indicated to the function so it can transform these values to the desired output:

```
trainRSSI <- ipfTransform(ipftrain[, 1:168], outRange = c(0.1, 1), inNoRSSI = NA,
                          outNoRSSI = NA)
testRSSI <- ipfTransform(ipftest[, 1:168], outRange = c(0.1, 1), inNoRSSI = NA,
                         outNoRSSI = NA)
```

The ipfTransform function returns a new data set with the same structure (vector, matrix or data frame) as the input.

Positioning algorithms

This section describes three positioning algorithms implemented in the ipft package. The examples illustrating each description are based on the data previously scaled in Section Data wrangling .

The ipfKnn function.

The ipfKnn and ipfEstimate functions implement a version of the knn algorithm to select the k nearest neighbors (the k more similar vectors from the training set) to a given RSSI vector. Many different distance metrics (Torres-Sospedra et al., 2015b) can be used to compare two RSSI vectors and measure how 'near' or similar they are.

The distance metrics implemented in the package include some typical functions, as the L^1 norm, or manhattan distance, or the L^2, or euclidean distance. The L^u norm between two fingerprints with indices a and b is defined as follows:

$$L^u = \left(\sum_{h=1}^{q} |(\rho_{h,a} - \rho_{h,b}|^u \right)^{1/u}$$

The package also implements some fingerprinting specific distance estimation functions such as LDG and PLGD. The LGD between two RSSI vectors λ_i^{tr} and λ_j^{ts} of longitude q is given by:

$$LGD(\lambda_i^{tr}, \lambda_j^{ts}) = - \sum_{h=1}^{q} \log \ \max(G(\rho_{h,i}^{tr}, \ \rho_{h,j}^{ts}), \ \epsilon)$$

where ϵ is a parameter to avoid logarithm of zero, as well as having one beacon RSSI value influence the LGD only above a certain threshold. $G(\rho_{h,i}^{tr}, \ \rho_{h,j}^{ts})$ represents the Gaussian similarity between $\rho_{h,i}^{tr}$ and $\rho_{h,j}^{ts}$, defined as

$$\mathrm{G}(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\rho_{h,i}^{tr} - \rho_{h,j}^{ts})^2}{2\sigma^2} \right), & \text{if } \rho_{h,i}^{tr} \neq 0 \,\text{and}\, \rho_{h,j}^{ts} \neq 0 \\ 0, & otherwise \end{cases}$$

The σ^2 parameter represents the shadowing variance (Shrestha et al., 2013). Values for σ in the range between 4 and 10 dBm are usually good for indoor scenarios (Lohan et al., 2014).

The PLGD between two RSSI vectors λ_i^{tr} and λ_j^{ts} of longitude q is given as:

$$PLGD(\lambda_i^{tr}, \lambda_j^{ts}) = LGD(\lambda_i^{tr}, \lambda_j^{ts}) + \alpha(\phi(\lambda_i^{tr}, \lambda_j^{ts}) + \phi(\lambda_j^{ts}, \lambda_i^{tr}))$$

where $\phi(\lambda_i^{tr}, \lambda_j^{ts})$ is a penalty function for the beacons that are visible in the i^{th} training fingerprint but not in the j^{th} test fingerprint, $\phi(\lambda_j^{ts}, \lambda_i^{tr})$ is a penalty function for the beacons that are visible in the j^{th} test fingerprint but not in the i^{th} training fingerprint, and are defined as follows:

$$\phi(\lambda_i^{tr}, \lambda_j^{ts}) = \sum_{h=1}^{q} T_{max} - \rho_{h,i}^{tr}, \ \text{for} \ \ 0 < \rho_{h,i}^{tr} \leqslant T_{max} \text{ and } r_i = 0)$$

$$\phi(\lambda_j^{ts}, \lambda_i^{tr}) = \sum_{h=1}^{q} T_{max} - \rho_{h,j}^{ts}, \ \text{for} \ \ 0 < \rho_{h,j}^{ts} \leqslant T_{max} \text{ and } r_j = 0)$$

T_{max} is an upper threshold for the strength of the signal, and α is a scaling factor.

The similarity measurement method can be chosen by means of the parameter method, or by providing a custom function (parameters FUN and ...). The signature of the ipfKnn function is:

```
ipfKnn <- function(train_fgp, train_pos, k = 3, method = 'euclidean',
                   weights = 'distance', norm = 2, sd = 5, epsilon = 1e-3,
                   alpha = 1, threshold = 20, FUN = NULL, ...)
```

where:

- `train_fgp`: A data frame of n rows and q columns containing the fingerprint vectors of the training set.
- `train_pos`: A data frame of n rows and l columns containing the positions of the training observations.
- `k`: The k parameter of the *knn* algorithm, the number of nearest neighbors to consider.
- `method`: The distance metric to be used by the algorithm. The implemented options are 'euclidean', 'manhatan', 'norm', 'LGD' and 'PLGD'
- `weights`: The weight function to be used by the algorithm. The implemented options are 'distance' and 'uniform'. The default 'distance' function calculate the weights from the distances as:

  $$w_{j,t} = \frac{1}{(1 + d_{j,t})\mathcal{W}_j}$$

  where $w_{j,t}$ is the weight assigned to the t^{th} ($t \in [1..k]$) neighbor of the j^{th} ($j \in [1..m]$) test observation, $d_{j,t}$ is the distance in the feature (RSSI) space between the t^{th} neighbor and the j^{th} test fingerprint, and $\mathcal{W}_j$ is a term used to normalize the values so that the total sum of the k weights is 1.

  The 'uniform' function assigns the same weight value to each neighbor:

  $$w_{j,t} = \frac{1}{k}$$

- `norm,sd,epsilon,alpha,threshold`: Parameters for the 'norm', 'LGD' and 'PLGD' methods.
- `FUN`: An alternative function provided by the user to compute the distance.
- `...`: Additional parameters for the function FUN.

For a training data set of n RSSI vectors (a data frame or a matrix named `tr_fingerprints`) and a data set of n position vectors (a data frame or a matrix named `tr_positions`), the code for fitting a knn model with a k value of 4 and the manhattan distance as the distance measurement method is:

```
knnModel <- ipfKnn(tr_fingerprints, tr_positions, k = 4, method = 'manhattan')
```

This function returns an S3 object of class `ipftModel` containing the following properties:

- `params`: A list with the parameters passed to the function.
- `data`: A list with the fingerprints and the location data of the radio map.

To estimate the position of a new fingerprint, the `ipfEstimate` function makes use of the previously obtained model. An `ipfModel` object holds the data model needed by the `ipfEstimate` function to apply the selected algorithm and returns an estimation of the test fingerprints positions. The signature of `ipfEstimate` is:

```
ipfEstimate <- function(ipfmodel, test_fgp, test_pos = NULL)
```

where:

- `ipfmodel`: An S3 object of class `ipfModel`.
- `test_fgp`: A data frame of m rows and q columns containing the fingerprints of the test set.
- `test_pos`: An optional parameter containing a data frame of m rows and l columns with the position of the test observations.

The `ipfEstimate` function returns an S3 object of the class `ipfEstimation` with the following elements:

- `location`: A $m \times l$ matrix with the predicted position for each observation in the `test` data set.
- `errors`: If the actual location of the test observations is passed in parameter `test_pos`, and the data that represents the position is numeric, this property returns a numeric vector of length n with the errors, calculated as the *euclidean* distances between the actual and the predicted locations.

- `confusion`: If the actual location of the test observations is passed in parameter `test_pos`, and the data that represents the position is a factor, the estimation of the actual position is performed as a classification task, and this property returns a confusion matrix summarizing the results of this classification.
- `neighbors`: A $m \times k$ matrix with the indices of the k selected neighbors for each observation in the `test` data set.
- `weights`: A $m \times k$ matrix containing the weights assigned by the algorithm to the selected neighbors.

The following `R` code shows an example of the usage of the `ipfKnn` function with the data set included in the package. This example takes the data previously scaled and generates a positioning model from the input data `trainRSSI` (the radio map) that is stored in `knnModel`. Then, the model is passed to the `ipfEstimate` function, along with the test data, to get an estimation of the position of the 702 test observations:

```
tr_fingerprints <- trainRSSI[, 1:168]
tr_positions    <- ipftrain[, 169:170]
knnModel        <- ipfKnn(tr_fingerprints, tr_positions, k = 7, method = "euclidean")
ts_fingerprints <- testRSSI[, 1:168]
ts_positions    <- ipftest[, 169:170]
knnEstimation   <- ipfEstimate(knnModel, ts_fingerprints, ts_positions)
```

Since the position of the test observations is known, the mean error for the 702 test observations can be calculated as follows:

```
> mean(knnEstimation$errors)
[1] 3.302739
```

The mean positioning error is one of the most common evaluation metrics used in indoor positioning (Liu et al., 2007) to assess the system's accuracy. This metric corresponds to the average Euclidean distance between the estimated locations and the true locations. As positions in the `ipftrain` and `ipftest` are expressed in meters, this metric represents the average error in meters for this scenario.

The neighbors selected from the training data set for the 6 first test fingerprints are:

```
> head(knnEstimation$neighbors)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   71  176  126  125  127  771  130
[2,]   71  176  126  125  127  771  130
[3,]  465  914  915  913  217   77  218
[4,]  465  914  915  176  913  461  217
[5,]  176  126  125  771  130  127  914
[6,]   77  914  915  217  176  465  218
```

where each row of the output corresponds to the indices of the $k = 7$ more similar vectors from the training data set to the i^{th} vector of the `test` data set.

As an example of how to use `ipfKnn` with a custom function, the next code shows the definition of a *C++* function that implements a modified version of the manhattan distance. The function needs at least two parameters, the two matrices representing the training and test data sets. A third parameter is here introduced to represent a penalization value. This function penalizes the computed distance between two RSSI measurements when one of the beacons is not detected (represented by the value $\varnothing$), by multiplying the resulting distance by a factor F. Given two fingerprints λ_i^{tr} and λ_j^{ts} of length q, the *myD* distance is:

$$\mathrm{myD}(\lambda_i^{tr}, \lambda_j^{ts}) = \sum_{h=1}^{q} \mathrm{myd}(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}),$$

where

$$\mathrm{myd}(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}) = \begin{cases} |\rho_{h,i}^{tr} - \rho_{h,j}^{ts}|, & \text{if } \rho_{h,i}^{tr} \neq \varnothing \, \text{and} \, \rho_{h,j}^{ts} \neq \varnothing \\ |\rho_{h,i}^{tr} - \rho_{h,j}^{ts}|F, & \textit{otherwise} \end{cases}$$

The following code implements the `myD` function and shows an example of its usage with `ipfKnn`, as well as the results obtained. The function is coded in *C++* to improve its performance when using large data sets, although the method also accepts custom plain R functions. The myD function assumes that the fingerprints are in a positive range:

```
library('ipft')
library('Rcpp')
cppFunction('
  NumericMatrix myD(NumericMatrix train, NumericMatrix test, double F = 2.0) {
    NumericMatrix distanceMatrix(test.nrow(), train.nrow());
    double d = 0, pv = 0, rssi1 = 0, rssi2 = 0;
    for (int itrain = 0; itrain < train.nrow(); itrain++) {
      for (int itest = 0; itest < test.nrow(); itest++) {
        d = 0;
        for (int i = 0; i < train.ncol(); i++) {
          rssi1 = R_IsNA(train(itrain, i))? 0 : train(itrain, i);
          rssi2 = R_IsNA(test(itest, i))? 0 : test(itest, i);
          pv = (rssi1 != 0 && rssi2 != 0)? 1 : F;
          d = d + std::abs(rssi1 - rssi2) * pv;
        }
        distanceMatrix(itest, itrain) = d;
      }
    }
    return distanceMatrix;
  }'
)
customModel      <- ipfKnn(tr_fingerprints, tr_positions, k = 1, FUN = myD, F = 0.25)
customEstimation <- ipfEstimate(customModel, ts_fingerprints, ts_positions)

> head(customEstimation$neighbors)
     [,1]
[1,]  773
[2,]  773
[3,]  776
[4,]  773
[5,]  130
[6,]  130
```

The previous code outputs the selected neighbors for the first 6 observations in the test data set. As the ts_positions data frame contains the actual location of the observations, the absolute error committed by the model is returned in the ipfEstimation object:

```
> head(customEstimation$errors)
[1] 5.708275 5.708275 5.708275 5.708275 3.380000 3.380000
```

```
And the mean error with this custom similarity function is:
```

```
> mean(customEstimation$errors)
[1] 3.297342
```

An ipfEstimation object can be used directly to plot the Empirical cumulative distribution function of the error (function ipfPlotEcdf()) and the Probability density function (function ipfPlotPdf()). Figures 1 and 2 show the plots obtained from the following code:

```
> ipfPlotEcdf(customEstimation)
> ipfPlotPdf(customEstimation)
```

The plotting functions included in the package are described in detail in Section Plotting functions.

The ipfProbabilistic function.

Given the limitations of sensors accuracy (Luo and Zhan, 2014) and the irregular character of signal propagation (Ali et al., 2010), the RSSI vector stored for a particular position cannot have completely reliable and accurate information about the emitters signal strength. This uncertainty is generally modeled by a normal distribution (Haeberlen et al., 2004), but to do so many readings of the signals at the same position are needed to obtain a representative set of statistical parameters to model each RSSI present at that position.

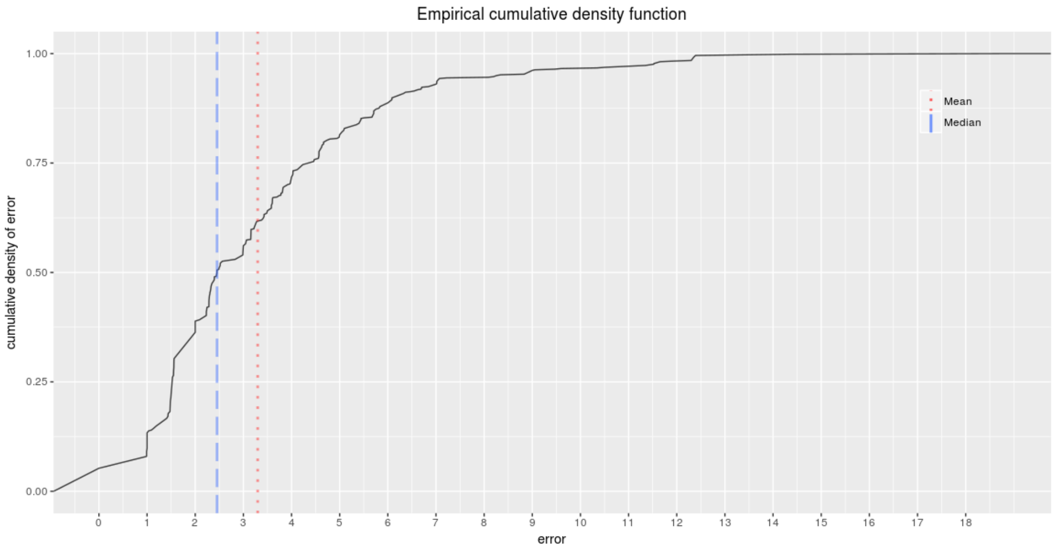


Figure 2: Funtion `ipfPlotEcdf`. Empirical cumulative distribution function of the error. The plot also shows the mean (red dotted line) and the median (blue dashed line) of the errors.

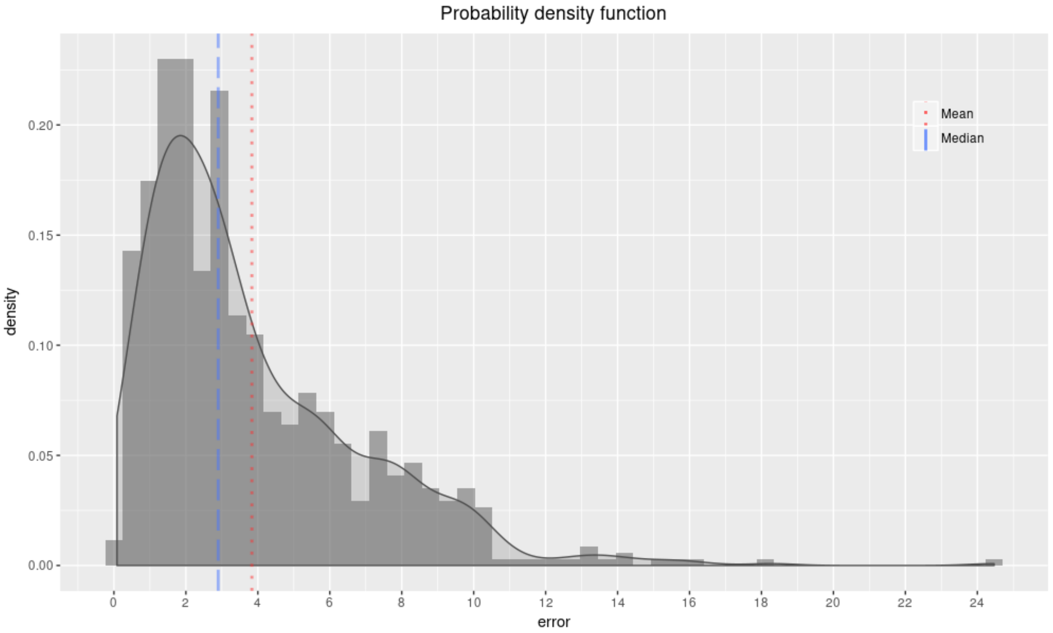


Figure 3: Funtion `ipfPlotPdf`. Probability density function. The plot shows the normalized histogram of the errors and its density function. The plot also shows the mean (red dotted line) and the median (blue dashed line) of the errors.

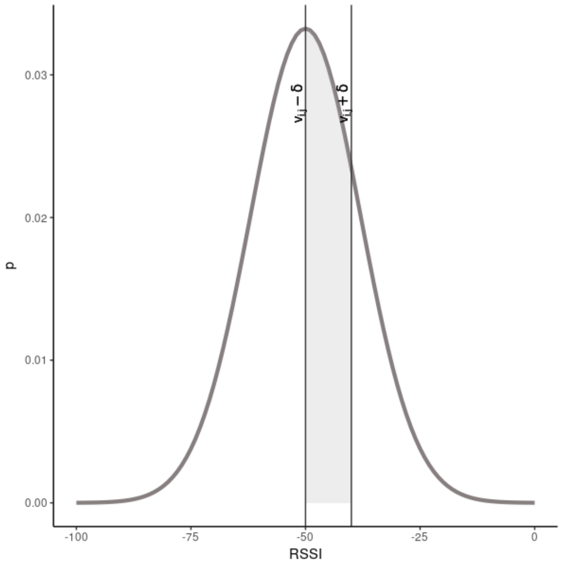


Figure 4: δ parameter for the probabilistic approach. This parameter sets the width of the discretization steps.

Thus, the initial collection of RSSI observations associated to a particular point is transformed into a pair of vectors containing the means and the standard deviations of the RSSI for each beacon, and then the complete training data is stored as a set of statistical parameters that can be used to infer the location of a test observation as the one that maximizes a probability function.

Let $\widehat{\mathcal{D}}_{train}$ be the new training set obtained from the previous procedure:

$$\widehat{\mathcal{D}}_{train} = \left\{\widehat{\mathcal{F}}_{train},\ \widehat{\mathcal{L}}_{train}\right\}$$

$$\widehat{\mathcal{F}}_{train} = \left\{\widehat{\lambda_1^{tr}}, \widehat{\lambda_2^{tr}}, ..., \widehat{\lambda_g^{tr}}\right\}$$

$$\widehat{\mathcal{L}}_{train} = \left\{\widehat{\tau_1^{tr}}, \widehat{\tau_2^{tr}}, ..., \widehat{\tau_g^{tr}}\right\}$$

where $\widehat{\mathcal{F}}_{train}$ is the set of statistical parameters obtained from the fingerprints of the training set, g is the number of groups of fingerprints with the same associated position, and $\widehat{\mathcal{L}}_{train}$ is the set of positions associated to each group. Each one of the g observations of the training data set is now composed of a fingerprint with q values:

$$\widehat{\lambda_i^{tr}} = \left\{\theta_{1,i}^{tr}, \theta_{2,i}^{tr}, ..., \theta_{q,i}^{tr}\right\},\ i \in [1, ..., g]$$

$$\theta_{h,i}^{tr} \sim \mathcal{N}(\mu_{h,i}, \sigma_{h,i}^2)$$

where $\mu_{h,i}$ and $\sigma_{h,i}^2$ are the mean and the variance, respectively, of the h^{th} RSSI of the i^{th} group of original fingerprints.

Let $\rho_{h,j}^{ts}$ be the h^{th} RSSI measurement of the j^{th} test fingerprint (λ_j^{ts}), and let $\mu_{h,i}$ and $\sigma_{h,i}^2$ be the mean and the standard deviation of the h^{th} beacon distribution obtained for the i^{th} position from the training set. The probability $p_{h,j}^{(i)}$, of observing $\rho_{h,j}^{ts}$ at the i^{th} position is:

$$p_{h,j}^{(i)} = \int_{\rho_{h,j}^{ts} - \delta}^{\rho_{h,j}^{ts} + \delta} \frac{1}{\sigma_{h,i}\sqrt{2\pi}}\ \mathrm{e}^{-\frac{x-\mu_{h,i}}{2\sigma_{h,i}^2}}\ dx$$

where δ is a parameter to allow the discretization of the normal distribution (Figure 4).

The set of all probabilities $p_{h,j}^{(i)},\ h \in [1, ..., q]$ obtained for a given test observation j, expresses the similarity between the observation measurement and the training data for a particular location. An evaluation of the total similarity for every location can be computed as a function of these individual probabilities, like its sum or its product. In the **ipft** package, this algorithm is implemented by the `ipfProbabilistic` and `ipfEstimate` functions, and by default uses the sum of probabilities as default operator to evaluate the similarity:

$$\psi_j^{(i)} = \sum_{h=1}^{p} p_{h,j}^{(i)}$$

where $\psi_j^{(i)}$ is the similarity between the j^{th} test observation and the i^{th} distribution from the training data set. The function to evaluate the similarity can be passed to `ipfProbabilistic` as a parameter.

As well as the `ipfKnn` and `ipfProximity` funtions, `ipfProbabilistic` returns a `ipfModel` object with the same data structure seen in Section The `ipfKnn` function, but with the difference that now the `data` property returns the probabilistic parameters that define the fitted distributions for every group of fingerprints on the training set. The clustering or grouping of the training data is performed by default over the location data provided by the user, but this behavior can be customized by passing a parameter with the columns over which to group the data, or by passing the group indices directly. The `ipft` package implements two functions (`ipfGroup()` and `ipfCluster()`) to perform clustering tasks. These functions are described in Section Data clustering.

The signature of the `ipfProbabilistic` function is:

```
ipfProbabilistic <- function(train_fgp, train_pos, group_cols = NULL, groups = NULL,
                             k = 3, FUN = sum, delta = 1, ...)
```

where `train_fgp`, `train_pos` and `k` have the same meaning and structure as described in Section The `ipfKnn` function, and, given n observations in the training set:

- `groups`: is a numeric vector of length n, containing the index of the group assigned to each observation of the training set. This parameter is optional.
- `group_cols`: is a character vector with the names of the columns to use as criteria to form groups of fingerprints. This parameter is optional.
- `FUN`: is a function to estimate a similarity measure from the calculated probabilities.
- `delta`: is a parameter to specify the interval around the test RSSI value to take into account when determining the probability.
- `...`: are additional parameters for `FUN`.

The following code shows how to use the `ipfProbabilistic` function to obtain a probabilistic model from the `ipftrain` and `ipftest` data sets. The default behavior of `ipfProbabilistic` groups the training data attending at the position of each observation, in this case, its x and y coordinates:

```
> probModel <- ipfProbabilistic(tr_fingerprints, tr_positions, k = 7, delta = 10)
> head(probModel$data$positions)
     X     Y
1 -0.6 24.42
2 -0.6 27.42
3  0.0  0.00
4  0.4  0.00
5  0.4  3.38
6  0.4  6.81
```

Now the `ipfModel$data` property returns a list with 3 elements:

- `means`: a data frame with the means for every beacon and every group of fingerprints.
- `sds`: a data frame with the standard deviations for every beacon and every group of fingerprints.
- `positions`: a data frame with the position of each group of fingerprints.

To obtain an estimation from this model, the same code used in section The `ipfKnn` function can be used to produce the estimated locations:

```
> ts_fingerprints <- ipftest[, 1:168]
> ts_positions    <- ipftest[, 169:170]
> probEstimation  <- ipfEstimate(probModel, ts_fingerprints, ts_positions)
```

and their errors and its mean value:

```
> mean(probEstimation$errors)
[1] 6.069336
```

An alternative function can be passed to **ipfProbabilistic**. The following code uses the maximum value of the probabilities as the similarity measure, and passes a parameter to remove NAs from the data[7]:

```
> probModel      <- ipfProbabilistic(tr_fingerprints, tr_positions, k = 9, delta = 10,
+                                     FUN = max, na.rm = TRUE)
> probEstimation <- ipfEstimate(probModel, ts_fingerprints, ts_positions)
> mean(probEstimation$errors)
[1] 8.652321
```

The ipfProximity function.

When the location of the access points is known, it's possible to estimate the position of a fingerprint using the log-distance path loss model (Seybold, J.S., 2005). Given a set of q beacons, and a fingerprint vector $\lambda = \{\rho_1, \rho_2, ..., \rho_q\}$ of length q, this model is expressed as:

$$\rho_h = P_{1m,h} - 10\alpha \log_{10} d_h - \gamma, \quad h \in [1, 2, ..., q]$$

where ρ_h is the value of the received signal from the h^{th} beacon, d_h is the distance from the observation to the beacon, $P_{1m,h}$ is the received power at 1 meter from the emitter, α is the path loss exponent, and $\gamma \sim \mathcal{N}(0, \sigma_\gamma^2)$ represents a zero mean Gaussian noise that models the random shadowing effects of the environment.

The estimator of the distance between the emitting beacon and the position where the signal is received is:

$$\hat{d}_h = 10^{\frac{\rho_h - P_{1m,h}}{10\alpha}}$$

This estimation follows a log-normal distribution that is:

$$\ln \hat{d}_h \sim \mathcal{N}(\ln d_h, \sigma_d^2)$$

where $\sigma_d = (\sigma_\gamma ln10)/(10\alpha)$.

The mean and the variance of the distribution are:

$$E[\hat{d}_h] = d_h \ e^{\sigma_d^2/2}$$
$$Var[\hat{d}_h] = d_h^2 \ e^{\sigma_d^2} \ (e^{\sigma_d^2} - 1)$$

Note that the variance grows quadratically with the distance, making the estimation less reliable as the distance becomes larger. Therefore, the distances estimated from different beacons will have different accuracies. To take this into account, the algorithm estimates the position of a fingerprint as a minimization problem of the overall squared error of the estimated distances. The objective function to minimize is:

$$\underset{\tau}{min} J = \sum_{h=1}^{p} \omega_h (\hat{d}_h - \|s_h - \tau\|)^2$$

where τ is the position that minimizes the function, that is, the estimated position, q is the number of beacons present in the fingerprint, and $\omega_h = 1/Var[\hat{d}_h]$ are the weights.

The functions **ipfProximity** and **ipfEstimate** implement this design, and uses the Broyden-Fletcher-Goldfard-Shano algorithm (BFGS) (Broyden, 1969; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970), a quasi-Newton method, to minimize the previous function to make an estimation of the fingerprint position. The accuracy of the estimation is strongly dependent on the reliability of the emitters positions. When these positions are unknown, they can be estimated with the function **ipfEstimateBeaconPositions**. Section Beacon position estimation details the implementation and usage of this function. The **ipfProximity** function returns an **ipfModel** object with the data needed by the **ipfEstimate** function to estimate a fingerprint position.

The signature of the **ipfProximity** function is:

[7] The ipfProbabilistic function takes into account the NAs contained in the data when using the default function (sum), but the user needs to manage this situation when a custom function is provided. In this example, the data is not previously transformed, is passed as it is, with NAs for not detected WAPs, to illustrate this situation.

```
ipfProximity <- function(bpos, rssirange = c(-100, 0), norssi = NA, alpha = 5,
                         wapPow1 = -30)
```

where:

- `bpos`: a matrix or a data frame containing the position of the beacons, in the same order as they appear in fingerprints.
- `rssirange`: the range of the RSSI data in the fingerprints.
- `norssi`: the value used to represent a not detected beacon.
- `alpha`: the path loss exponent (α).
- `wapPow1`: a numeric vector with the received power at one meter distance from the beacon ($P_{1m,h}$). If only one value is supplied, it will be assigned to all beacons.

In the following example, the goal is to estimate the position of the 702 fingerprints included in the test set, using the known position of the WAPs and the log-distance path loss model. The `ipfpwap` dataset contains the location of 39 of the 168 wireless access points of the `ipftrain` and `ipftest` data sets. The `ipfProximity` function returns a model that is used to estimate the position of the fingerprints. As the real position of the test fingerprints is known, this information can be also passed to the `ipfEstimate` function. Thus, the returned `ipfEstimation` object will contain, along with the estimated positions, the associated errors:

```
> proxModel      <- ipfProximity(ipfpwap, alpha = 4, rssirange = c(-100, 0),
+                                norssi = NA, wapPow1 = -32)
> fingerprints   <- ipftest[, 1:168]
> positions      <- ipftest[, 169:170]
> proxEstimation <- ipfEstimate(proxModel, ipftest[, 1:168], ipftest[, 169:170])
> mean(proxEstimation$errors)
[1] 8.0444
```

Positioning algorithms comparison

In a classical fingerprint-based positioning system, the radio map is constructed in accordance to the positioning algorithm to be used in the online phase. The knn algorithm follows a deterministic approach that performs well in most cases, while the probabilistic method is based on the assumption that there is enough training data for each particular position to obtain reliable parameters to model a distribution for each signal at each survey location. As regards to the proximity algorithm, it is based on two assumptions; first, the ability to realistically simulate the propagation model of the signal, and second, the known positions of the emitter beacons. These conditions are not met in many scenarios, where changes in occupation, for example, modify the propagation model and thus the performance of the positioning system.

To illustrate the previous considerations, Table 1 shows the mean and the quartile errors in meters for the implemented algorithms, computed using the dataset included in the package. In this particular case, given the characteristics of the training data, knn performs better than the rest.

		Quartile error (m)				
algorithm	mean error (m)	0%	25%	50%	75%	100%
knn	3.3027	0.15172	1.46891	2.61281	4.08992	19.84650
probabilistic	6.0693	0.14289	3.26988	5.63051	8.19933	17.93031
proximity	8.0444	2.49865	5.71055	7.42602	9.88427	20.12029

Table 1: Comparison of the algorithms' accuracy on the dataset included in the package

To compare the performance of the proposed implementation of the previous positioning algorithms, we ran a benchmark test of 1000 iterations on each function, using the dataset included in the package. The results for the model fitting functions are shown in Table 2. As it can be seen, the proximity and knn algorithms are the fastest, as expected, since their model fitting process basically consists in storing the training data for later processing during the estimation stage. In contrast, the probabilistic algorith has to fit a normal distribution for each signal received at each position, and thus, it takes longer to complete the process.

function	elapsed (sec)	relative
ipfKnn	0.031	1.409
ipfProbabilistic	1035.446	47065.727
ipfProximity	0.022	1.000

Table 2: Performance comparison of the model building functions

The outcomes are different when considering the results for the estimation function (Table 3). The position estimation for the probabilistic algorithm is faster that the rest. For the knn algorithm, the estimation process could be improved using clustering techniques to avoid comparing the test fingerprint with all the instances in the training set. With regards to the estimation process for the proximity algorithm, the fact that the result is computed by solving an unconstrained nonlinear optimization through an iterative method highly penalyzes its performance.

model	function	elapsed (sec)	relative
knn	ipfEstimate	2508.079	2.998
probabilistic	ipfEstimate	836.651	1.000
proximity	ipfEstimate	28259.110	33.776

Table 3: Performance comparison of the estimation functions on each model

Beacon position estimation

If the actual position of the beacons is unknown, it can be estimated in many ways from the RSSI data. Two basic methods for estimation of the beacons location have been included in the `ipft` package through the `ipfEstimateBeaconPositions` function. The 'centroid' and the 'weighted centroid' methods.

Both methods use the fingerprint data to guess the position of the beacons. Let q be the number of beacons and $\tau^{\mathcal{B}}$ be the set of beacons locations:

$$\tau^{\mathcal{B}} = \left\{ \nu_{1,h}^{\mathcal{B}}, \nu_{2,h}^{\mathcal{B}}, \nu_{3,h}^{\mathcal{B}} \right\}, \ h \in [1, 2, ..., q]$$

the position of the h^{th} beacon is given by:

$$\tau_h^{\mathcal{B}} = \left\{ \sum_{i=1}^{n} \omega_i \nu_{1,i}^{tr}, \ \sum_{i=1}^{n} \omega_i \nu_{2,i}^{tr}, \ \sum_{i=1}^{n} \omega_i \nu_{3,i}^{tr} \right\}$$

where n is the number of fingerprints in the training set. The value of ω_i is:

$$\omega_i = \frac{1}{n}$$

for the 'centroid' method and:

$$\omega_i = \frac{\rho^{tr}_{h,i}}{\sum_{l=1}^{n} \rho^{tr}_{h,l}}$$

for the 'weighted centroid' method. Since the biggest weights have to be assigned to the strongest RSSI values, the fingerprint vector values should be positive, or at least, positively correlated to the beacon received intensity. This is checked by the function implementation so the input data is internally transformed to a positive range when needed.

This is the signature of the `ipfEstimateBeaconPositions` function:

```
ipfEstimateBeaconPositions <- function(fingerprints, positions, method = 'wcentroid',
                                        rssirange = c(-100, 0), norssi = NA)
```

where:

- `fingerprints`: is a data frame with the fingerprint vectors as rows.
- `positions`: a data frame with the position of the fingerprints.
- `method`: the method to use by the algorithm, either 'centroid' or 'wcentroid'.
- `rssirange`: the range of the signal strength values of the fingerprints.
- `norssi`: the value assigned in the fingerprints to a non detected beacon.

The following code uses the function `ipfEstimateBeaconPositions` with the 'weighted centroid' method to estimate the position of the wireless access points, under the assumption that this position is unknown. Finally, the function `ipfProximity` estimates the positions of the first 6 test fingerprints:

```
> bc_positions   <- ipfEstimateBeaconPositions(ts_fingerprints, ts_positions,
                                               method = 'wcentroid')
> proxModel      <- ipfProximity(bc_positions, rssirange = c(0.1, 1),
+                                norssi = NA)
> proxEstimation <- ipfEstimate(proxModel, fingerprints[1:6,],
+                               positions[1:6,])
> proxEstimation$location
        V1       V2
1 1.686950 12.02117
2 1.686950 12.02117
3 1.654255 10.91767
4 1.682121 10.96035
5 1.711448 10.88966
6 1.695007 10.09507
```

Data clustering

Clustering techniques can be used with the aim of enhancing localization performance and reducing computational overhead (Cramariuc et al., 2016b). The **ipft** package includes some functions for cluster analysis and grouping of the fingerprinting and location data. These functions can be used to create or detect clusters based on the position of the observations, on its signal levels, or on any other criteria that might be useful to group the data by. Performing RSSI clustering before the positioning process groups a large number of reference points into various clusters that can be used to perform first-level classification. This allows to assess the testing point location by using only the fingerprints in the matched cluster rather than the whole radio map. Furthermore, given the amplitude atenuation that building partitions cause to electromagnetic signals, clusters usually can be related to physical spaces such as buildings, floors or even rooms.

The main function for clustering tasks is `ipfCluster`. The more basic usage of the function takes the provided data and uses the k-means algorithm to classify it into k disjoint sets of observations, by selecting a set of k cluster centers to minimize the sum of the squared distances between the data vectors and their corresponding centers.

The k-means clustering procedure begins with an initial set of randomly selected centers, and iteratively tries to minimize the sum of the squared distances. This makes the algorithm very sensitive to the arbitrary selection of initial centers, and introduces variability in the results obtained from one execution to another. Besides, the number of clusters has to be established beforehand, and that may be inconvenient in some scenarios.

The signature of the `ipftCluster` function is:

```
ipfCluster <- function(data, method = 'k-means', k = NULL, grid = NULL, ...)
```

where

- `data`: is a data frame with the data to cluster. When using the *k-means* method, the data frame must not contain any `NA` values.
- `method`: the algorithm used to create clusters. The implemented algorithms are 'k-means' for k-means algorithm, 'grid' for clustering based on spatial grid partition, and 'AP' for affinity propagation algorithm.
- `k`: a numeric parameter for k-means algorithm.
- `grid`: a numeric vector with the size of the grid for the grid algorithm.

When using the default k-means algorithm, the function behaves as a wrapper around the k-means function of the **stats** package, and therefore, the usage can be further customized by passing extra parameters, as the number of iterations or the algorithm to be used ("Hartigan-Wong" is the default).

The following example will find $k = 30$ clusters of similar fingerprints in the `ipftrain` dataset. First the data set of fingerprints is transformed to eliminate the `NA` values that represent a not detected beacon. Then, the data is passed to the `ipfCluster` function to find the 30 clusters using the 'MacQueen' algorithm:

```
> set.seed(1)
> cl_fingerprints <- ipfTransform(tr_fingerprints, inNoRSSI = NA, outNoRSSI = 0)
> clusterData     <- ipfCluster(cl_fingerprints, k = 30, iter.max = 20,
+                               algorithm = "MacQueen")
> head(clusterData$clusters)
[1] 3 3 3 3 3 3
```

The outcome of the `ipfCluster` function is a list containing the indices of the k clusters and its centroids. Given the previous example, `clusterData$centers` will return the k centroids, and `clusterData$clusters` will return the cluster index $i \in [1, .., k]$ for every observation in `ipftrain`.

The `ipfCluster` function includes an implementation of the affinity propagation (AP) algorithm (Frey and Dueck, 2007) that can be used to estimate the number of distinct clusters present in the radio map. AP does not require the number of clusters to be determined before running it. It finds members of the input set, known as 'exemplars', that are representative of clusters by creating the centers and the corresponding clusters based on the constant exchanging of reading similarities between the observations. This message-passing process continues until a good set of centers and corresponding clusters emerges.

The following code uses AP clustering to find groups of similar RSSI vectors from the `ipftrain` data set. With no further parametrization, it will classify the RSSI data into 43 distinct clusters:

```
> clusterData     <- ipfCluster(tr_fingerprints, method = 'AP')
> dim(clusterData$centers)
[1]  43 168
```

Now, `clusterData$centers` holds the 43 'exemplars', those RSSI vectors from the radio map that are representative of a cluster, and `clusterData$clusters` contains the indices that link every observation of the data set with its assigned cluster.

To perform a more simple grouping based on a precise set of variables, the `ipfGroup` function provides a method to group the data by column name. The function signature is:

```
ipfGroup <- function(data, ...)
```

where

- `data`: is a data frame with the data to group.
- `...`: The variables to group the data by.

The ipfGroup function returns a numeric vector with the same length as the number of observations contained in the data data frame, containing the index of the group assigned to each observation. The following example groups the data according to the position of the observations, that in the ipftrain and ipftest datasets are represented by the columns 'X' and 'Y':

```
> groups <- ipfGroup(ipftrain, X, Y)
> head(groups)
[1]  4  4  4  4 22 22
> length(unique(groups))
[1]  41
```

Plotting functions

Indoor positioning generally involves statistical analysis of datasets, and the **ipft** provides some useful functions to produce graphs for exploring data. All the graphic functions included in the package are built upon the **ggplot2** package (Wickham, 2011), and return a ggplot object that can be plotted or further personalized with custom labels, theme, etc.

The ipfPlotPdf and the ipfPlotEcdf have already been introduced in Section The ipfKnn function. These functions will plot the probability density function and the empirical cumulative distribution function, respectively. Both functions take an ipfEstimation object to produce the plot, while the axis labels and plot tittle can be also supplied by the parameters xlab, ylab and tittle. Their respective signatures are:

```
ipfPlotPdf <- function(estimation, xlab = 'error', ylab = 'density',
                       title = 'Probability density function')

ipfPlotEcdf <- function(estimation, xlab = 'error',
                        ylab = 'cumulative density of error',
                        title = 'Empirical cumulative density function')
```

The function ipfPlotLocation will produce a plot of the location of the data. The following code shows its signature and presents an example of its use. The example calls the function with parameter plabel set to TRUE, to plot labels identifying each location, and reverseAxis set to TRUE to swap the axis. It also modifies the resulting object by changing the default **ggplot2** theme to the white one. The result is shown in Figure 5.

```
ipfPlotLocation <- function(positions, plabel = FALSE, reverseAxis = FALSE,
                            xlab = NULL, ylab = NULL, title = '')

library(ggplot2)
ipfPlotLocation(ipftrain[, 169:170], plabel = TRUE, reverseAxis = TRUE) + theme_bw()
```

The function ipfPlotEstimation plots the estimated position of the test observations based on an ipfModel object and an ipfEstimation object, as well as the actual position (parameter testpos), if known, and the position of the k selected fingerprints from the training set used to guess its location (parameter showneighbors). The green dots indicate the actual position of the observations, while the black dots indicate the estimated ones. The blue lines connect the estimated positions with the k neighbors from which the location has been estimated, and the red arrows connect the actual position of the fingerprint with the estimated one. The following code shows the function signature and provides an example of its usage. The result plot is shown in Figure 6:

```
ipfPlotEstimation <- function(model, estimation, testpos = NULL, observations = c(1),
                              reverseAxis = FALSE, showneighbors = FALSE,
                              showLabels = FALSE, xlab = NULL, ylab = NULL,
                              title = '')

library(ggplot2)
probModel <- ipfProbabilistic(ipftrain[, 1:168], ipftrain[, 169:170])
probEst   <- ipfEstimate(probModel, ipftest[, 1:168], ipftest[, 169:170])
ipfPlotEstimation(probModel, probEst, ipftest[, 169:170],
                  observations = c(61:62, 81:82), reverseAxis = TRUE,
                  showneighbors = TRUE, showLabels = TRUE) + theme_bw()
```

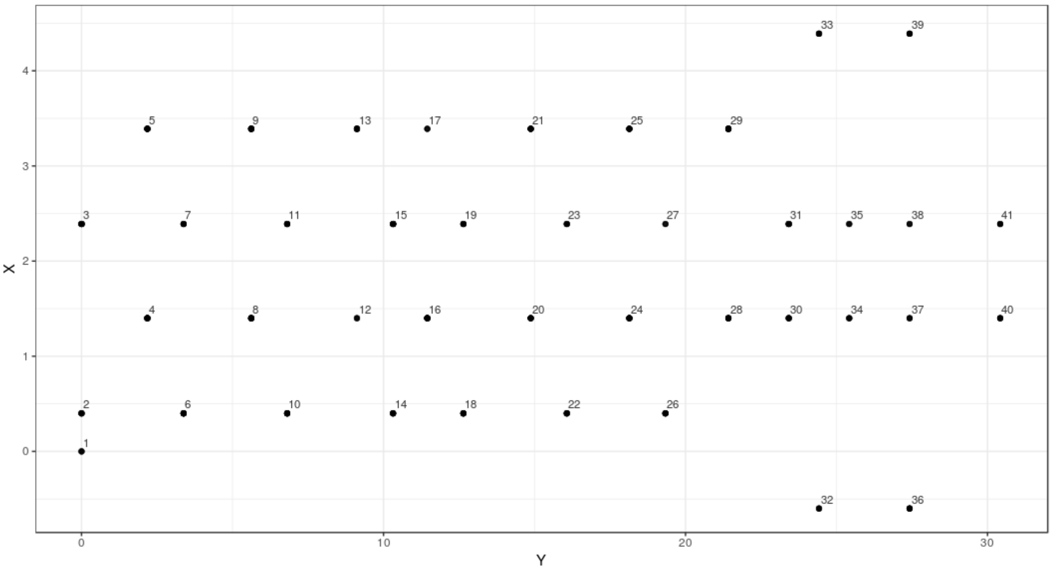


Figure 5: Location of fingerprints included in the **ipftrain** data frame. The labels indicate the group indices.

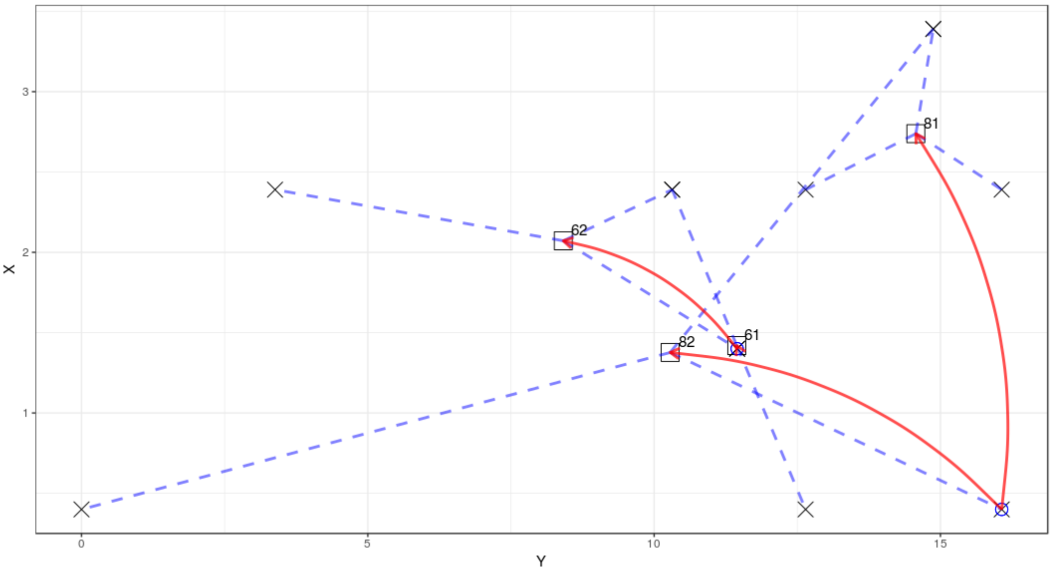


Figure 6: Estimated and actual positions of test observations 61, 62, 81 and 82 from the **ipftrain** data set. The circles indicate the actual positions of the observations. The squares show the estimated positions. The red arrows connect the actual positions with the estimated ones. The dashed lines connect the estimated positions with the k neighbors from which the location has been estimated, represented by the crosses.

Summary

In this paper, the package ipft is presented. The main goal of the package is to provide researchers with a set of functions to manipulate, cluster, transform, create models and make estimations using indoor localization fingerprinting data. This package enables researchers to use a well established set of algorithms and tools to manipulate and model RSSI fingerprint data sets, and also allows them to customize the included algorithms with personalized parameters and functions to adapt the working mode to their particular research interests.

In this work some of the fundamental algorithms used in indoor fingerprinting localization techniques have been formally presented and illustrated, while detailed examples and information about its usage and implementation have been provided.

Future work

This package is an ongoing work, and future versions will implement new algorithms and tools with the aim of providing a base framework for researchers, and become a reference library for fingerprinting-based indoor positioning research.

In particular, future lines of work should consider the implementation of deep learning based algorithms. Many deep learning techniques can be exploited to try to obtain better positioning performance. Recurrent neural networks could be used to learn not only spatial but also temporal patterns of the received signals. Deep autoencoders can be implemented as a way to encode fingerprints and reduce their dimensionality to a few number of significant features. Their variational and generative extensions can be of use to better model the stochastic nature of RSSI data. These models can also be applied to generate new training data for deep learning-based clasisifiers, increasing the robustness of positioning systems and trying to address problems caused by heterogeneity of devices.

Acknowledgements

The authors would like to thank the two anonymous reviewers for providing useful feedback that helped to improve the paper.

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness through the "Proyectos I + D Excelencia" programme (TIN2015-70202-P) and by Jaume I University "Research promotion plan 2017" programme (UJI-B2017-45).

Bibliography

E. Aarts and R. Wichert. Ambient intelligence. In *Technology Guide*, pages 244–249. Springer-Verlag, 2009. URL https://doi.org/10.1007/978-3-540-88546-7_47. [p67]

A. H. Ali, M. R. A. Razak, M. Hidayab, S. A. Azman, M. Z. M. Jasmin, and M. A. Zainol. Investigation of Indoor WIFI Radio Signal Propagation. In *Proceedings of the Symposium on Industrial Electronics and Applications, (ISIEA'10)*, pages 117–119, 2010. URL https://doi.org/10.1109/isiea.2010.5679486. [p76]

C. Broyden. A new double-rank minimisation algorithm. preliminary report. In *Notices of the American Mathematical Society*, volume 16, page 670. American Mathematical Society 201 Charles ST, Providence, RI 02940-2213, 1969. [p80]

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. URL https://doi.org/10.1109/tit.1967.1053964. [p70]

A. Cramariuc, H. Huttunen, and E. S. Lohan. Clustering benefits in mobile-centric wifi positioning in multi-floor buildings. In *2016 International Conference on Localization and GNSS (ICL-GNSS)*, pages 1–6. IEEE, 2016a. URL https://doi.org/10.1109/icl-gnss.2016.7533846. [p71]

A. Cramariuc, H. Huttunen, and E. S. Lohan. Clustering Benefits in Mobile-Centric WiFi Positioning in Multi-Floor Buildings. In *Proceedings of the 6th International Conference on Localization and GNSS (ICL-GNSS'16)*, pages 1–6, 2016b. URL https://doi.org/10.1109/icl-gnss.2016.7533846. [p70, 83]

R. Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970. [p80]

B. J. Frey and D. Dueck. Clustering by Passing Messages between Data Points. *Science*, 315(5814): 972–976, 2007. URL https://doi.org/10.1126/science.1136800. [p84]

T. Gigl, G. J. M. Janssen, V. Dizdarevic, K. Witrisal, and Z. Irahhauten. Analysis of a uwb indoor positioning system based on received signal strength. In *Proceedings of the 4th Workshop on Positioning, Navigation and Communication (PNC'07)*, pages 97–101, 2007. URL https://doi.org/10.1109/wpnc.2007.353618. [p68]

D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970. URL https://doi.org/10.1090/s0025-5718-1970-0258249-6. [p80]

A. Haeberlen, E. Flannery, A. M. Ladd, A. Rudys, D. S. Wallach, and L. E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom'04)*, pages 70–84, 2004. URL https://doi.org/10.1145/1023720.1023728. [p70, 76]

A. B. Harbicht, T. Castro-Santos, W. R. Ardren, D. Gorsky, and D. J. Fraser. Novel, continuous monitoring of fine-scale movement using fixed-position radiotelemetry arrays and random forest location fingerprinting. *Methods in Ecology and Evolution*, 8(7):850–859, 2017. URL https://doi.org/10.1111/2041-210x.12745. [p67]

S. He and S. H. G. Chan. Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons. *IEEE Communications Surveys & Tutorials*, 18(1):466–490, 2016. URL https://doi.org/10.1109/comst.2015.2464084. [p67]

N. E. Klepeis, W. C. Nelson, W. R. Ott, J. P. Robinson, A. M. Tsang, P. Switzer, J. V. Behar, S. C. Hern, and W. H. Engelmann. The national human activity pattern survey (nhaps): a resource for assessing exposure to environmental pollutants. *Journal Of Exposure Analysis And Environmental Epidemiology*, 11:231 EP –, 2001. URL https://doi.org/10.1038/sj.jea.7500165. [p67]

B. Li, J. Salter, A. Dempster, and C. Rizos. Indoor positioning techniques based on wireless lan. In *Proceedings of the 1st IEEE International Conference on Wireless Broadband and Ultra Wide-Band Communications (AusWireless'06)*, pages 13–16, 2006. URL https://opus.lib.uts.edu.au/bitstream/2100/170/1/113_Li.pdf. [p68]

H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 37(6):1067–1080, 2007. URL https://doi.org/10.1109/tsmcc.2007.905750. [p67, 75]

Y. Liu, H. Du, and Y. Xu. The research and design of the indoor location system based on rfid. In *Proceedings of the 4th International Symposium on Computational Intelligence and Design (ISCID'11)*, pages 87–90, 2011. URL https://doi.org/10.1109/iscid.2011.123. [p68]

E. S. Lohan, K. Koski, J. Talvitie, and L. Ukkonen. WLAN and RFID Propagation Channels for Hybrid Indoor Positioning. In *Proceedings of the 4th International Conference on Localization and GNSS, (ICL-GNSS'14)*, 2014. URL https://doi.org/10.1109/icl-gnss.2014.6934184. [p73]

J. Luo and X. Zhan. Characterization of Smart Phone Received Signal Strength Indication for WLAN Indoor Positioning Accuracy Improvement. *Journal of Networks*, 9(3):739–746, 2014. URL https://doi.org/10.4304/jnw.9.3.739-746. [p76]

A. Popleteev, V. Osmani, O. Mayora, and A. Matic. Indoor localization using audio features of fm radio signals. In *International and Interdisciplinary Conference on Modeling and Using Context*, pages 246–249. Springer, 2011. URL https://doi.org/10.1007/978-3-642-24279-3_26. [p67]

Y. Quan, L. Lau, F. Jing, Q. Nie, A. Wen, and S.-Y. Cho. Analysis and machine-learning based detection of outlier measurements of ultra-wideband in an obstructed environment. In *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 997–1000. IEEE, 2017. URL https://doi.org/10.1109/indin.2017.8104909. [p67]

Research and markets. Indoor location market by component, deployment mode, application, vertical and region - global forecast to 2022. *Research and markets*, 2017. URL https://www.researchandmarkets.com/reports/4416241/indoor-location-market-by-component-deployment. [p67]

T. Roos, P. Myllymäki, H. Tirri, P. Misikangas, and J. Sievänen. A Probabilistic Approach to WLAN User Location Estimation. *International Journal of Wireless Information Networks*, 9(3): 155–164, 2002. URL https://doi.org/10.1023/a:1016003126882. [p70]

E. Sansano. *ipft: Indoor Positioning Fingerprinting Toolset*, 2017. URL https://cran.r-project.org/web/packages/ipft/index.html. [p67]

Seybold, J.S. *Introduction to RF Propagation.* John Wiley & Sons, 2005. [p71, 80]

D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970. URL https://doi.org/10.1090/s0025-5718-1970-0274029-x. [p80]

S. Shrestha, J. Talvitie, and E. S. Lohan. On the Fingerprints Dynamics in WLAN Indoor Localization. In *Proceedings of the 13th International Conference on ITS Telecommunications (ITST'13)*, pages 122–126, 2013. URL https://doi.org/10.1109/itst.2013.6685532. [p73]

J. Torres-Sospedra, R. Montoliu, A. Martinez-Uso, J. P. Avariento, T. J. Arnau, M. Benedito-Bordonau, and J. Huerta. UJIIndoorLoc: A New Multi-Building and Multi-Floor Database for WLAN Fingerprint-Based Indoor Localization Problems. In *Proceedings of the 5th International Conference on Indoor Positioning and Indoor Navigation (IPIN'14)*, pages 261–270, 2015a. URL https://doi.org/10.1109/ipin.2014.7275492. [p71]

J. Torres-Sospedra, R. Montoliu, S. Trilles, Óscar Belmonte, and J. Huerta. Comprehensive Analysis of Distance and Similarity Measures for Wi-Fi Fingerprinting Indoor Positioning Systems. *Expert Systems with Applications*, 42(23):9263–9278, 2015b. URL https://doi.org/10.1016/j.eswa.2015.08.013. [p70, 72, 73]

Y. Wang, X. Yang, Y. Zhao, Y. Liu, and L. Cuthbert. Bluetooth positioning using rssi and triangulation methods. In *Proceedings of the 10th IEEE Consumer Communications and Networking Conference, (CCNC'13)*, pages 837–842, 2013. URL https://doi.org/10.1109/ccnc.2013.6488558. [p68]

W. Werner, J. Rabaey, and E. H. L. Aarts, editors. *Ambient Intelligence.* Springer-Verlag, 2005. ISBN 978-3-540-27139-0. URL https://doi.org/10.1007/b138670. [p67]

H. Wickham. ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(2):180–185, 2011. URL https://doi.org/10.1002/wics.147. [p85]

J. Xiao, Z. Zhou, Y. Yi, and L. M. Ni. A Survey on Wireless Indoor Localization from the Device Perspective. *ACM Computing Surveys*, 49(2):1–31, 2016. URL https://doi.org/10.1145/2933232. [p68]

C. Yang, T. Nguyen, and E. Blasch. Mobile Positioning via Fusion of Mixed Signals of Opportunity. *IEEE Aerospace and Electronic Systems Magazine*, 29(4):34–46, 2014. URL https://doi.org/10.1109/maes.2013.130105. [p69]

J. yub Lee, C. hwan Yoon, H. Park, and J. So. Analysis of location estimation algorithms for wifi fingerprint-based indoor localization. In *Proceedings of the 2nd International Conference on Software Technology (SoftTech'13)*, pages 89–92, 2013. [p67]

Emilio Sansano
Institute of New Imaging Technologies
Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12017 Castelló de la Plana
Spain
esansano@uji.es

Raúl Montoliu
Institute of New Imaging Technologies
Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12017 Castelló de la Plana
Spain
montoliu@uji.es

Óscar Belmonte
Institute of New Imaging Technologies

Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12017 Castelló de la Plana
Spain
oscar.belmonte@uji.es

Joaquín Torres-Sospedra
Institute of New Imaging Technologies
Universitat Jaume I
Av. de Vicent Sos Baynat, s/n 12017 Castelló de la Plana
Spain
jtorres@uji.es

What's for dynr: A Package for Linear and Nonlinear Dynamic Modeling in R

by Lu Ou[+], *Michael D. Hunter*[+], *and Sy-Miin Chow*

Abstract Intensive longitudinal data in the behavioral sciences are often noisy, multivariate in nature, and may involve multiple units undergoing regime switches by showing discontinuities interspersed with continuous dynamics. Despite increasing interest in using linear and nonlinear differential/difference equation models with regime switches, there has been a scarcity of software packages that are fast and freely accessible. We have created an **R** package called **dynr** that can handle a broad class of linear and nonlinear discrete- and continuous-time models, with regime-switching properties and linear Gaussian measurement functions, in **C**, while maintaining simple and easy-to-learn model specification functions in **R**. We present the mathematical and computational bases used by the **dynr R** package, and present two illustrative examples to demonstrate the unique features of **dynr**.

Introduction

The past several decades have seen a significant rise in the prevalence of intensive longitudinal data (ILD), particularly in the social and behavioral sciences (Bolger and Laurenceau, 2013; Byrom and Tiplady, 2010; Stone et al., 2008). Differential equation and difference equation models in the form of state-space models have been one of the most dominant tools for representing the dynamics of ILD in disciplines such as the physical sciences, econometrics, engineering, and ecology. In parallel, some computational advances have been proposed in estimating *regime-switching* models — namely, models positing how otherwise continuous dynamic processes may undergo discontinuous changes through categorical but unobserved phases known as "regimes" (Kim and Nelson, 1999; Hamilton, 1989; Muthén and Asparouhov, 2011; Chow et al., 2013, 2015; Dolan, 2009). Throughout, we use the terms *regimes* and *classes* interchangeably to denote unobserved unit- and time-specific indicator variables that serve to group portions of repeated measures into phases with homogeneous dynamics or measurement properties.

Examples of regime-switching phenomena from psychology includes Piaget's (1969) theory of human cognitive development and related extensions (Dolan et al., 2004; van der Maas and Molenaar, 1992; Hosenfeld, 1997); Kohlberg's (Kohlberg and Kramer, 1969) conceptualization of stagewise development in moral reasoning; Van Dijk and Van Geert's (2007) findings on discrete shifts in early language development; as well as Fukuda and Ishihara's (1997) work on the discontinuous changes in infant sleep and wakefulness rhythm during the first six months of life. Related to, but distinct from, hidden Markov models (Elliott et al., 1995; Visser, 2007), regime-switching differential and difference equation models allow researchers to specify targeted differential or difference functions to describe the continuous changes that occur within regimes. Ample work exists on fitting these models (Hamilton, 1989; Dolan, 2009; Yang and Chow, 2010; Chow et al., 2013; Chow and Zhang, 2013; Chow et al., 2015; Muthén and Asparouhov, 2011; Tong and Lim, 1980; Tiao and Tsay, 1994), but readily accessible software suited for handling such models with ILD are lacking.

Several programs and packages exist for fitting differential equation, difference equation, and hidden Markov models. However, each program has certain limitations that **dynr** (Ou et al., 2018) aims to overcome. Speaking broadly, the largest differences between **dynr** and other packages are threefold: (1) **dynr** readily allows for multi-unit models, (2) **dynr** allows for nonlinear discrete-time and continuous-time dynamics, and (3) **dynr** allows for regime switching throughout every part of the model. Many **R** packages exist for univariate and multivariate time series. CRAN lists hundreds of packages in its task view for *TimeSeries* (Hyndman, 2016), a complete review of which is well-beyond the scope of this work. However, generally these packages lack facilities for fitting time series from multiple units. Likewise there are very few software utilities designed for nonlinear dynamics or regime switching (see Table 1 for an overview). Petris and Petrone (2011) reviewed three packages for linear state-space models: **dlm** (Petris, 2010, 2014), **KFAS** (Helske, 2017a,b), and **dse** (Gilbert, 2006 or later, 2015). These are among the state of the art for state-space modeling in **R**. Although **KFAS** can accommodate in its measurement model all densities within the exponential family, the corresponding dynamic model is required to be linear. In addition to these **R** packages, the **OpenMx** 2.0 release (Neale et al., 2016; Boker et al., 2017) has maximum likelihood time-varying linear discrete- and continuous-time state-space modeling (Hunter, 2017). Likewise, the MKFM6 program (Dolan, 2005) implements methods of Harvey (1989) for time-invariant linear state-space

[+] These two authors contributed equally to the work.

models. SsfPack (Koopman et al., 1999) implements the methods of Durbin and Koopman (2001) for linear state-space modeling and Markov chain Monte Carlo methods for nonlinear modeling, but it is primarily restricted to single-unit time series without regime switching. The **ctsem** package (Driver et al., 2017b,a) has utilities for linear state-space modeling of multiple units in continuous time, but lacks functionality for nonlinear models or regime switching. **MATLAB** (The MathWorks, Inc., 2016) has numerous extensions for time series and state-space modeling (Grewal and Andrews, 2008), but lacks the ability to include regime switching and multiple units. Some **R** packages that handle regime switching are only designed for hidden Markov models, for example, **depmixS4** (Visser and Speekenbrink, 2016, 2010) and **RHmm** (Taramasco and Bauer, 2012), while the others are only for specific Markov-switching discrete-time time-series models, including **MSwM** (Sanchez-Espigares and Lopez-Moreno, 2014) for univariate autoregressive models, **MSBVAR** (Brandt, 2016) for vector autoregressive models, and **MSGARCH** (Ardia et al., 2017) for generalized autoregressive conditional heteroskedasticity models. The **pomp** package (King et al., 2016, 2018) lists among its features hidden Markov models and state-space models, both of which can be discrete- or continuous-time, non-Gaussian, and nonlinear. However, **pomp** does not currently support regime-switching functionality beyond the regime switching found in hidden Markov modeling. Helske (2017a) included a review of numerous other packages for non-Gaussian time series models which generally do not involve latent variables.

Overall, developments in fitting differential/difference equation models that evidence discontinuities in dynamics are still nascent. Despite some of the above-mentioned advances in computational algorithms, there is currently no readily available software package that allows researchers to fit differential/difference equations with regime-switching properties. As stated previously, currently available computational programs for dynamic modeling are limited in one of several ways: (1) they are restricted to handling only linear models within regimes such as the package **OpenMx**, (2) they can only handle very specific forms of nonlinear relations among latent variables, (3) they are computationally slow, (4) they do not allow for stochastic qualitative shifts in the dynamics over time, or (5) they require that the user write complex compiled code to enhance computational speed at the cost of high user burden. Efficient and user-friendly computer software needs to be developed to overcome these restrictions so the estimation of dynamic models can become more applicable by researchers.

We present an **R** package, **dynr**, that allows users to fit both linear and nonlinear differential and difference equation models with regime-switching properties. All computations are performed quickly and efficiently in **C**, but are tied to a user interface in the familiar **R** language. Specifically, for a very broad class of linear and nonlinear differential/difference equation models with linear Gaussian measurement functions, **dynr** provides **R** helper functions that write appropriate **C** code based on user input in **R** into a local (potentially temporary) **C** file, which is then compiled on user's end with a call to an **R** function in **dynr**. The **C** function pointers are passed to the back-end for computation of a negative log-likelihood function, which is numerically optimized also in **C** using the optimization routine SLSQP (Kraft, 1988, 1994) for parameter estimation. During the process, the user never has to write or even see the **C** code that underlies **dynr** and yet, the computations are performed entirely in **C**, with no interchanges between **R** and **C** to reduce memory copying and optimize speed. This removes some of the barriers to dynamic modeling, opening it as a possibility to a broader class of users, while retaining the flexibility of specifying targeted model-specific functions in **C** for users wishing to pursue models that are not yet supported in the **R** interface.

In the remaining sections, we will first present the mathematical and computational bases of the **dynr R** package, and then demonstrate the interface of **dynr** for modeling multivariate observations with Gaussian measurement errors using two ILD modeling examples from the social and behavioral sciences. Key features of the **dynr** package we seek to highlight include: (1) **dynr** fits discrete- and continuous-time dynamic models to multivariate longitudinal/time-series data; (2) **dynr** deals with dynamic models with regime-switching properties; (3) for improved speed, **dynr** computes and optimizes negative log-likelihood function values in **C**; (4) **dynr** handles linear and nonlinear dynamic models with an easy-to-use interface that includes a matrix form (for linear dynamic models only) and formula form (for linear as well as nonlinear models); (5) **dynr** removes the burden on the user to perform analytic differentiation in fitting nonlinear differential/difference equation models by providing the user with **R**'s symbolic differentiation; and (6) **dynr** provides ready-to-present results through LaTeX equations and plots.

General modeling framework

At a basic level, our general modeling framework comprises a dynamic model and a measurement model. The former describes the ways in which the latent variables change over time, whereas the latter portrays the relationships between the observed variables and latent variables at a specific time.

The dynamic model for a particular regime in continuous-time assumes the following form:

$$d\boldsymbol{\eta}_i(t) = \boldsymbol{f}_{S_i(t)}\left(\boldsymbol{\eta}_i(t), t, \boldsymbol{x}_i(t)\right) dt + d\boldsymbol{w}_i(t), \tag{1}$$

where i indexes the smallest independent unit of analysis, t indexes time, $\boldsymbol{\eta}_i(t)$ is the $r \times 1$ vector of latent variables at time t, $\boldsymbol{x}_i(t)$ is the vector of covariates at time t, and $\boldsymbol{f}_{S_i(t)}(.)$ is the vector of (possibly nonlinear) dynamic functions which depend on the latent regime indicator, $S_i(t)$. The left-hand side of Equation 1, $d\boldsymbol{\eta}_i(t)$, gives the differential of the vector of continuous latent variables, $\boldsymbol{\eta}_i(t)$, and $\boldsymbol{f}_{S_i(t)}(.)$ is called the *drift* function. Added to these deterministic changes induced by the drift function is $\boldsymbol{w}_i(t)$, an r-dimensional Wiener process. The differentials of the Wiener processes have zero means and covariance matrix, $\mathbf{Q}_{S_i(t)}$, called the *diffusion* matrix. When the dynamic model consists only of linear functions, Equation 1 reduces to:

$$d\boldsymbol{\eta_i}(t) = \left(\boldsymbol{\alpha}_{S_i(t)} + \boldsymbol{F}_{S_i(t)}\boldsymbol{\eta_i}(t) + \boldsymbol{B}_{S_i(t)}\boldsymbol{x}_i(t)\right) dt + d\boldsymbol{w}_i(t). \tag{2}$$

where the general function $\boldsymbol{f}_{S_i(t)}()$ is replaced with a linear function consisting of (1) an intercept term $\boldsymbol{\alpha}_{S_i(t)}$, (2) linear dynamics in a matrix $\boldsymbol{F}_{S_i(t)}$, and (3) linear covariate regression effects $\boldsymbol{B}_{S_i(t)}$.

For discrete-time processes, we adopt a dynamic model in state-space form (Durbin and Koopman, 2001) as

$$\boldsymbol{\eta_i}(t_{i,j+1}) = \boldsymbol{f}_{S_i(t_{i,j})}\left(\boldsymbol{\eta_i}(t_{i,j}), t_{i,j}, \boldsymbol{x}_i(t_{i,j+1})\right) + \boldsymbol{w}_i(t_{i,j+1}), \tag{3}$$

now postulated to unfold at discrete time points indexed by sequential positive integers, $t_{i,j}$, $j = 1, 2, \cdots$. In this case, $\boldsymbol{w}_i(t_{i,j})$ denotes a vector of Gaussian distributed process noise with covariance matrix, $\mathbf{Q}_{S_i(t_{i,j})}$. We have intentionally kept notation similar between discrete- and continuous-time models to facilitate their linkage. **dynr** allows for an easy transition between these two frameworks with a binary flag. In a similar vein, we refer to $\boldsymbol{f}_{S_i(t)}(.)$ in both Equations 1 and 3 broadly as the *dynamic functions.* The same structure as Equation 2 is possible in discrete time as the linear analog of Equation 3,

$$\boldsymbol{\eta_i}(t_{i,j+1}) = \boldsymbol{\alpha}_{S_i(t_{i,j})} + \boldsymbol{F}_{S_i(t_{i,j})}\boldsymbol{\eta_i}(t_{i,j}) + \boldsymbol{B}_{S_i(t_{i,j})}\boldsymbol{x}_i(t_{i,j+1}) + \boldsymbol{w}_i(t_{i,j+1}). \tag{4}$$

In both the discrete- and continuous-time cases, the initial conditions for the dynamic functions are defined explicitly to be the latent variables at a unit-specific first observed time point, $t_{i,1}$, denoted as $\boldsymbol{\eta}_i(t_{i,1})$, and are specified to be normally distributed with means $\boldsymbol{\mu}_{\boldsymbol{\eta}_1}$ and covariance matrix, $\boldsymbol{\Sigma}_{\boldsymbol{\eta}_1}$:

$$\boldsymbol{\eta}_i(t_{i,1}) \sim N\left(\boldsymbol{\mu}_{\boldsymbol{\eta}_1}, \boldsymbol{\Sigma}_{\boldsymbol{\eta}_1}\right). \tag{5}$$

Likewise for both discrete- and continuous-time models, we assume that observations only occur at selected, discrete time points. Thus, we have a discrete-time measurement model in which $\boldsymbol{\eta}_i(t_{i,j})$ at discrete time point $t_{i,j}$ is indicated by a $p \times 1$ vector of manifest observations, $\boldsymbol{y}_i(t_{i,j})$. Continuous-time processes allow unequal time intervals for these observations. Missing data may be present under either specification. The vector of manifest observations is linked to the latent variables as

$$\boldsymbol{y}_i(t_{i,j}) = \boldsymbol{\tau}_{S_i(t_{i,j})} + \boldsymbol{\Lambda}_{S_i(t_{i,j})}\boldsymbol{\eta}_i(t_{i,j}) + \mathbf{A}_{S_i(t_{i,j})}\boldsymbol{x}_i(t_{i,j}) + \boldsymbol{\epsilon}_i(t_{i,j}), \quad \boldsymbol{\epsilon}_i(t_{i,j}) \sim N\left(\mathbf{0}, \mathbf{R}_{S_i(t_{i,j})}\right), \tag{6}$$

where $\boldsymbol{\tau}_{S_i(t_{i,j})}$ is a $p \times 1$ vector of intercepts, $\mathbf{A}_{S_i(t_{i,j})}$ is a matrix of regression weights for the covariates, $\boldsymbol{\Lambda}_{S_i(t_{i,j})}$ is a $p \times r$ factor loadings matrix that links the observed variables to the latent variables, and $\boldsymbol{\epsilon}_i(t_{i,j})$ is a $p \times 1$ vector of measurement errors assumed to be serially uncorrelated over time and normally distributed with zero means and covariance matrix, $\mathbf{R}_{S_i(t_{i,j})}$. Of course, all parts of the measurement model may be regime-dependent.

The subscript $S_i(t)$ in Equations 1–6 indicates that these functions and matrices may depend on $S_i(t)$, the operating regime. To make inferences on $S_i(t_{i,j})$, we initialize the categorical latent variable $S_i(t_{i,j})$ on the first occasion and then provide a model for how $S_i(t_{i,j})$ changes over time. The initial regime probabilities for $S_i(t_{i,1})$ are represented using a multinomial regression model as

$$\Pr\left(S_i(t_{i,1}) = m|\boldsymbol{x}_i(t_{i,1})\right) \triangleq \pi_{m,i1} = \frac{\exp(a_m + \mathbf{b}_m^T\boldsymbol{x}_i(t_{i,1}))}{\sum_{k=1}^{M}\exp(a_k + \mathbf{b}_k^T\boldsymbol{x}_i(t_{i,1}))}, \tag{7}$$

where M denotes the total number of regimes, a_m is the logit intercept for the mth regime and $\mathbf{b}_m$ is a $n_b \times 1$ vector of regression slopes linked to a vector of covariates that explain between-unit differences in initial log-odds (LO). For identification, a_m and all entries in $\mathbf{b}_m$ are set to zero for some regime, m.

We use a first-order Markov process to define how the classes change over time in a transition probability matrix, which contains all possible transitions from one regime to another. In the matrix, the rows index the previous regime at time $t_{i,j-1}$ and the columns index the current regime at time $t_{i,j}$. The rows of this matrix sum to 1 because the probability of transitioning from a particular state to any other state must be 1. This transition matrix may also depend on covariates. Thus, a multinomial logistic regression equation is assumed to govern the probabilities of transitions between regimes as:

$$\Pr\left(S_i(t_{i,j}) = m | S_i(t_{i,j-1}) = l, \boldsymbol{x}_i(t_{i,j})\right) \triangleq \pi_{lm,it} = \frac{\exp(c_{lm} + \mathbf{d}_{lm}^T \boldsymbol{x}_i(t_{i,j}))}{\sum_{k=1}^{M} \exp(c_{lk} + \mathbf{d}_{lk}^T \boldsymbol{x}_i(t_{i,j}))}, \tag{8}$$

where $\pi_{lm,it}$ denotes unit i's probability of transitioning from class l at time $t_{i,j-1}$ to class m at time $t_{i,j}$, c_{lm} denotes the logit intercept for the transition probability, and $\mathbf{d}_{lm}$ is a $n_d \times 1$ vector of logit slopes summarizing the effects of the covariates in $\boldsymbol{x}_i(t_{i,j})$ on that transition probability. One regime, again, has to be specified as the reference regime by fixing all LO parameters, including c_{lm} and all elements in $\mathbf{d}_{lm}^T$ for some regime m, to zero for identification purposes.

To summarize, the model depicted in Equations 1 – 8 may take on the form of various linear or nonlinear dynamic models in continuous or discrete time. Moreover, these dynamic models may have regime-switching properties. Systematic between-unit differences stem primarily from changes in the unit- and time-specific regime, $S_i(t_{i,j})$, and the corresponding changes in the dynamic and measurement models over units and occasions.

Estimation procedures

In this section, we outline the procedures implemented in **dynr** for estimating the model shown in Equations 1 – 8. An overview of the estimation procedures involved, the different special cases handled by **dynr**, and the software packages that can handle these special cases are summarized in Table 1.

Discrete-time models

Broadly speaking, the estimation procedures implemented in **dynr** are based on the Kalman filter (KF; Kalman, 1960), its various continuous-time and nonlinear extensions, and the Kim filter (Anderson and Moore, 1979; Bar-Shalom et al., 2001; Kim and Nelson, 1999; Yang and Chow, 2010; Chow and Zhang, 2013; Kulikov and Kulikova, 2014; Kulikova and Kulikov, 2014; Chow et al., 2018). The Kim filter, designed to extend the Kalman filter to handle regime-switching state-space models, was proposed by Kim and Nelson (1999) and extended by Chow and Zhang (2013) to allow for nonlinear dynamic functions. In **dynr**, models are allowed to (1) be in discrete or continuous time, (2) be single regime or regime switching, (3) have linear or nonlinear dynamics, (4) involve stochastic or deterministic dynamics, and (5) have one or more units. All combinations of these variations are possible in **dynr**, creating 32 different kinds of models.

In the case of linear discrete-time dynamics without regime-switching, the model reduces to a linear state-space model, and we apply the Kalman filter to estimate the latent variable values and obtain other by-products for parameter optimization. At each time point, the KF consists of two steps. In the first step, the dynamics are used to make a prediction for the latent state at the next time point conditional on the observed measurements up to time $t_{i,j-1}$, creating a predicted mean $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j-1}) = E(\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(t_{i,j-1}))$ and covariance matrix for the latent state $\mathbf{P}_i(t_{i,j}|t_{i,j-1}) = Cov[\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(t_{i,j-1})]$, where $\mathbf{Y}_i(t_{i,j-1})$ includes manifest observations from time $t_{i,1}$ up to time $t_{i,j-1}$. In the second step, the prediction is updated based on the measurement model (Equation 6) and the new measurements, yielding $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j}) = E(\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(t_{i,j}))$ and associated covariance matrix, $\mathbf{P}_i(t_{i,j}|t_{i,j}) = Cov[\boldsymbol{\eta}_{it}|\mathbf{Y}_i(t_{i,j})]$. Assuming that the measurement and process noise components are normally distributed and that the measurement equation is linear, as in Equation 6, the prediction errors, $\mathbf{Y}_i(t_{i,j}) - E(\mathbf{Y}_i(t_{i,j})|\mathbf{Y}_i(t_{i,j}))$, are multivariate normally distributed. Thus, these by-products of the KF can be used to construct a log-likelihood function known as the *prediction error decomposition* function (De Jong, 1988; Harvey, 1989; Hamilton, 1994; Chow et al., 2010). This log-likelihood function is optimized to yield maximum-likelihood (ML) estimates of all the time-invariant parameters, as well as to construct information criterion (IC) measures (Chow and Zhang, 2013; Harvey, 1989) such as the Akaike Information Criterion (AIC; Akaike, 1973) and Bayesian Information Criterion (BIC; Schwarz, 1978). Standard errors of the parameter estimates are obtained by taking the square root of the diagonal elements of the inverse of the negative numerical Hessian matrix of the prediction error decomposition function at the point of convergence.

At convergence, other products from the linear KF include updated latent states, $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j})$, and the updated latent covariance matrices, $\mathbf{P}_i(t_{i,j}|t_{i,j})$. In the social and behavioral sciences, the entire time series of observations has often been collected prior to model fitting. In such cases, we use the fixed interval smoother (Anderson and Moore, 1979; Ansley and Kohn, 1985) to refine the latent variable estimates, yielding the smoothed latent variable estimates, $\hat{\boldsymbol{\eta}}_i(t_{i,j}|T_i) = E(\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(T_i))$, and associated covariance matrices, $\mathbf{P}_i(t_{i,j}|T_i)$.

When the dynamic model takes on the form of a nonlinear state-space model with differentiable dynamic functions, the linear KF is replaced with the extended Kalman filter (EKF; Anderson and Moore, 1979; Bar-Shalom et al., 2001) so that the nonlinear dynamic functions are "linearized" or approximated by the first-order Taylor series. Then, a log-likelihood function can be constructed in similar form to the linear state-space prediction error decomposition. However, the corresponding parameter estimates are only "approximate" ML estimates due to the truncation errors in the EKF. The feasibility of this approach has been demonstrated by Chow et al. (2007).

When a linear state-space model is used as the dynamic model but it is characterized by regime-switching properties, **dynr** uses an extension of the KF, known as the Kim filter, and the related Kim smoother (Kim and Nelson, 1999; Yang and Chow, 2010). The Kim filter combines the KF, the Hamilton filter (Hamilton, 1989) that yields filtered state probabilities, and a collapsing procedure to avoid the need to store M^2 new values of $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j})^{l,m} \triangleq \mathbb{E}[\boldsymbol{\eta}_i(t_{i,j})|S_i(t_{i,j-1}) = l, S_i(t_{i,j}) = m, \mathbf{Y}_i(t_{i,j})]$, as well as $\mathbf{P}_i(t_{i,j}|t_{i,j})^{l,m} \triangleq \text{Cov}[\boldsymbol{\eta}_i(t_{i,j})|S_i(t_{i,j-1}) = l, S_i(t_{i,j}) = m, \mathbf{Y}_i(t_{i,j})]$ with each additional time point. The collapsing procedure averages the estimates over the previous regime l so only the marginal estimates, $\hat{\boldsymbol{\eta}}_i(t_{i,j}|t_{i,j})^m = E[\boldsymbol{\eta}_i(t_{i,j})|S_i(t_{i,j}) = m, \mathbf{Y}_i(t_{i,j})])$, and the associated covariance matrix, $\mathbf{P}_i(t_{i,j}|t_{i,j})^m$, need to be stored at each time step. To handle cases in which nonlinearities are present in Equation 3, a method proposed by Chow and Zhang (2013), called the extended Kim filter, is used for estimation instead. The extended Kim filter replaces the KF portion of the Kim filter with the EKF.

Continuous-time models

Finally, when the dynamics are in continuous time—whether composed of linear or nonlinear dynamic functions—the resultant estimation procedures are the continuous-discrete extended Kalman filter (CDEKF; Bar-Shalom et al., 2001; Kulikov and Kulikova, 2014; Kulikova and Kulikov, 2014). The CDEKF handles a single-regime special case of the general model shown in Equations 1–6.

For continuous processes in the form of Equation 1, let $\hat{\boldsymbol{\eta}}_i(t) = E(\boldsymbol{\eta}_i(t)|\mathbf{Y}_i(t_{i,j-1}))$ and $\mathbf{P}_i(t) = Cov[\boldsymbol{\eta}_i(t)|\mathbf{Y}_i(t_{i,j-1})]$ denote the mean and covariance matrix of the latent variables, respectively, at time t in the interval $[t_{i,j-1}, t_{i,j}]$. In the CDEKF framework, the prediction step of the KF is replaced by solving a set of ordinary differential equations (ODEs) at time $t_{i,j}$, given the initial conditions at time $t_{i,j-1}$: $\hat{\boldsymbol{\eta}}_i(t_{i,j-1}) = \hat{\boldsymbol{\eta}}_i(t_{i,j-1}|t_{i,j-1})$ and $\mathbf{P}_i(t_{i,j-1}) = \mathbf{P}_i(t_{i,j-1}|t_{i,j-1})$. This set of ODEs is obtained by only retaining the first term, $\boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right)$, in the Taylor series expansion of $\boldsymbol{f}_{S_i(t)}\left(\boldsymbol{\eta}_i(t), t, \boldsymbol{x}_i(t)\right)$ around the expectation $\hat{\boldsymbol{\eta}}_i(t)$, and is shown below:

$$\frac{d\hat{\boldsymbol{\eta}}_i(t)}{dt} = \boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right), \tag{9}$$

$$\frac{d\mathbf{P}_i(t)}{dt} = \frac{\partial \boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right)}{\partial \hat{\boldsymbol{\eta}}_i(t)}\mathbf{P}(t) + \mathbf{P}(t)\left(\frac{\partial \boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right)}{\partial \hat{\boldsymbol{\eta}}_i(t)}\right)^\top + \mathbf{Q}_{S_i(t)}, \tag{10}$$

where $\frac{\partial \boldsymbol{f}_{S_i(t)}(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t))}{\partial \hat{\boldsymbol{\eta}}_i(t)}$ is the Jacobian matrix of $\boldsymbol{f}_{S_i(t)}\left(\hat{\boldsymbol{\eta}}_i(t), t, \boldsymbol{x}_i(t)\right)$ with respect to $\hat{\boldsymbol{\eta}}_i(t)$ at time t. Kulikov and Kulikova (2014, Kulikova and Kulikov 2014) suggested solving for equations 9 and 10 using adaptive ODE solvers. We adopt an approximate numerical solution — the fourth-order Runge-Kutta (Press et al., 2002) method — to solve Equations 9 and 10. In cases where the hypothesized continuous-time dynamics are linear, explicit analytic solutions exist and there is no need to use numerical solvers. However, in our simulation work, estimating known special cases of linear stochastic differential equation models using numerical solvers yielded both comparable estimates and computational time to estimating the same models using their known solutions. Thus, for generality, we utilize numerical solvers in solving both linear and nonlinear differential equations in **dynr**.

As in the case involving nonlinear discrete-time dynamic models, parameter estimates obtained from optimizing the log-likelihood function constructed from by-products of the CDEKF are also approximate ML estimates; however, the approximations now stem both from the truncation errors from the first-order Taylor series in the CDEKF, as well as the numerical solution of Equations 9 and 10.

In cases involving regime-switching ordinary or stochastic differential equations, the algorithms for estimating regime-switching continuous-time models are essentially estimation procedures that combine the CDEKF and part of the Kim filter designed to handle estimation of the regime-switching portion of the model. The resultant procedure, referred to herein as *continuous-discrete extended Kim filter*, is summarized in Chow et al. (2018).

Steps for preparing and "cooking" a model

The theme around the naming convention exploits the pronunciation of the package name: **dynr** is pronounced the same as "dinner". Therefore, the names of functions and methods are specifically designed to relate to things done surrounding dinner, such as gathering ingredients such as the data, preparing recipes, cooking, which involves combining ingredients according to a "modeling" recipe and applies heat, and serving the finished product.

The general procedure for using the **dynr** package can be summarized in five steps. First, data are gathered and identified with the `dynr.data()` function. Second, *recipes* are prepared. To each part of a model there is a corresponding `prep.*()` recipe function. Each of these functions creates an object of class `"dynrRecipe"`. Each `prep.*()` function creates an object of class `"dynr*"` which is in turn a subclass of `"dynrRecipe"`. These recipe functions include:

1. The `prep.measurement()` function defines the measurement part of the model, that is, how latent variables and exogenous covariates map onto the observed variables.
2. The `prep.matrixDynamics()` and `prep.formulaDynamics()` functions define the dynamics of the model with either a strictly linear, matrix interface or with a possibly nonlinear formula interface, respectively.
3. The `prep.initial()` function defines the initial conditions of the model. The initial conditions are used by the recursive algorithms as the starting point for latent variable estimates. As such, the `prep.initial()` function describes the initial mean vector and covariance matrix of the latent variables, assumed to be multivariate normally distributed.
4. The `prep.noise()` function defines the covariance structure for both the measurement (or observation) noise and the dynamic (or latent) noise.
5. The `prep.regimes()` function provides the regime switching structure of the model. Single-regime models do not require a `"dynrRegimes"` object.

Once the data and recipes are prepared, the third step mixes the data and recipes together into a model object of class `"dynrModel"` with the `dynr.model()` function. Fourth, the model is cooked with `dynr.cook()` to estimate the free parameters and standard errors. Fifth and finally, results are served in summary tables using `summary()`, LaTeX equations using `printex()`, and plots of trajectories and equations using `plot()`, `dynr.ggplot()`, `autoplot()`, and `plotFormula()`.

We will demonstrate the interface of **dynr** using two examples: (1) a linear state-space example with regime-switching based on Yang and Chow (2010) and (2) a regime-switching extension of the predator-prey model (Lotka, 1925; Volterra, 1926).

Example 1: Regime-switching linear state-space model

Facial electromyography (EMG) has been used in the behavioral sciences as one possible indicator of human emotions (Schwartz, 1975; Cacioppo and Petty, 1981; Cacioppo et al., 1986; Dimberg et al., 2000). A time series of EMG data contains bursts of electrical activity that are typically magnified when an individual is under emotion induction. Yang and Chow (2010) proposed using a regime-switching linear state-space model in which the individual may transition between regimes with and without facial EMG activation. As such, heterogeneities in the dynamic patterns and variance of EMG data are also accounted for through the incorporation of these latent regimes. Model fitting was previously performed at the individual level. Data from the participant shown in Figure 1(A) are made available as part of the demonstrative examples in **dynr**. A complete modeling script for this example is available as a demo in **dynr** and can be found by calling `file.edit(system.file("demo","RSLinearDiscreteYang.R",package = "dynr"))`, and a full explanation is included as a package vignette called 'LinearDiscreteTimeModels'.

Here we present selected segments of code to showcase how a linear state-space model with regime-switching can be specified in **dynr**. The model of interest is the final model selected for this participant by Yang and Chow (2010):

$$y_i(t_{i,j}) = \mu_{yS_i(t_{i,j})} + \beta_{S_i(t_{i,j})}\text{Self-report}(t_{i,j}) + \eta_i(t_{i,j}), \tag{11}$$

$$\eta_i(t_{i,j+1}) = \phi_{S_i(t_{i,j})}\eta_i(t_{i,j}) + \zeta_i(t_{i,j+1}), \tag{12}$$

in which we allowed the intercept, $\mu_{yS_i(t_{i,j})}$; the regression slope, $\beta_{S_i(t_{i,j})}$; and the autoregression coefficient, $\phi_{S_i(t_{i,j})}$, to be regime-dependent. By allowing $\phi_{S_i(t_{i,j})}$ to be regime-specific, we indirectly allowed the total variance of the latent component, $\eta_i(t_{i,j+1})$, to be heterogeneous across the deactivation and activation stages, in spite of requiring the dynamic noise variance, $E(\zeta_i(t)^2)$, to be constant across regimes.

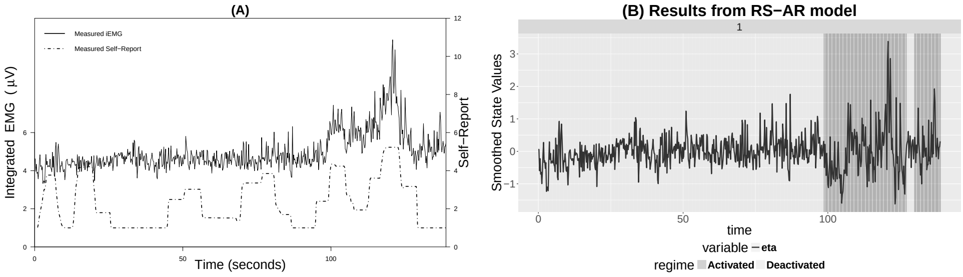


Figure 1: (A) A plot of integrated electromyography (iEMG) and self–report affect ratings for one participant with a time interval of 0.2 seconds between two adjacent observations. Self–report = self–report affect ratings; iEMG = integrated EMG signals. (B) An automatic plot of the smoothed state estimates for the regime-switching linear state-space model.

The first step in **dynr** modeling is to structure the data. This is done with the `dynr.data()` function.

```
require("dynr")
data("EMG")
EMGdata <- dynr.data(EMG, id = 'id', time = 'time',
  observed = 'iEMG', covariates = 'SelfReport')
```

The first argument of this function is either a `"ts"` class object of single-unit time series or a `"data.frame"` object structured in a long relational format with different measurement occasions from the same unit appearing as different rows in the data frame. When a `"ts"` class object is passed to `dynr.data()`, no other inputs are needed. Otherwise, the `id` argument needs the name of the variable that distinguishes units, allowing multiple replicated time series to be analyzed together. The `time` argument needs the name of the variable that indicates unit-specific measurement occasions. If a discrete-time model is desired, the time variable should contain sequential positive integers. If the measurement occasions for a unit are sequential but not consecutive, `NA`s will be inserted automatically to create equally spaced data. If a continuous-time model is being specified, the time variable can contain unit-specific increasing sequences of irregularly spaced real numbers. In this particular example, a discrete-time model is used. The `observed` and `covariates` arguments are vectors of the names of the observed variables and covariates in the data.

The next step in **dynr** modeling is to build the recipes for the various parts of a model. The recipes are created with `prep.*()` functions.

The dynamic functions in Equations 1 and 3, can be specified using either `prep.formulaDynamics()` or `prep.matrixDynamics()`. In this example, the dynamics as in Equation 12 are linear and discrete-time, so we can describe the dynamics in terms of Equation 4 as

$$\eta_i(t_{i,j+1}) = \underbrace{0}_{\boldsymbol{\alpha}_{S_i(t_{i,j})}} + \underbrace{\phi_{S_i(t_{i,j})}}_{\boldsymbol{F}_{S_i(t_{i,j})}} \eta_i(t_{i,j}) + \underbrace{0}_{\boldsymbol{B}_{S_i(t_{i,j})}} \boldsymbol{x}_i(t_{i,j}) + \underbrace{\zeta_i(t_{i,j+1})}_{\boldsymbol{w}_i(t_{i,j+1})}. \tag{13}$$

The `prep.matrixDynamics()` function allows the user to specify the structures of the intercept vector $\boldsymbol{\alpha}_{S_i(t_{i,j})}$, through `values.int` and `params.int`; the covariate regression matrix $\boldsymbol{B}_{S_i(t_{i,j})}$, through `values.exo` and `params.exo`; and the one-step-ahead transition matrix $\boldsymbol{F}_{S_i(t_{i,j})}$, through `values.dyn` and `params.dyn`. We illustrate this function below. The `values.dyn` argument gives a list of matrices for the starting values of $\boldsymbol{F}_{S_i(t_{i,j})}$. The `params.dyn` argument names the free parameters. These are the ϕ_{S_t} in Equation 12. The `isContinuousTime` argument switches between continuous-time modeling and discrete-time modeling. The arguments corresponding to the intercepts (`values.int` and `params.int`) and the covariate effects (`values.exo` and `params.exo`) are omitted to leave these matrices as zeros.

```
recDyn <- prep.matrixDynamics(values.dyn = list(matrix(0.1, 1, 1), matrix(0.5, 1, 1)),
  params.dyn = list(matrix('phi_1', 1, 1), matrix('phi_2', 1, 1)),
  isContinuousTime = FALSE)
```

The noise recipe is created with `prep.noise()`. The noise recipe is stored in the `recNoise` object, an abbreviation for "recipe noise". The latent noise covariance matrix is a 1 × 1 matrix with a free parameter called `dynNoise`, short for "dynamic noise". The observed noise covariance matrix is also a 1 × 1 matrix, but has the measurement noise variance fixed to zero using the special keyword `fixed`.

```
recNoise <- prep.noise(values.latent = matrix(1, 1, 1),
  params.latent = matrix('dynNoise', 1, 1),
  values.observed = matrix(0, 1, 1), params.observed = matrix('fixed', 1, 1))
```

The `prep.regimes()` function specifies the structure of the regime time evolution shown in Equation 8. In this example, we do not have any covariates in the regime-switching (RS) functions. The problem then reduces to the specification of a 2 × 2 transition log-odds (LO) matrix. We provide starting values that imply persisting in the same regime is more likely than transitioning to another regime, and set the second regime LO to zero for identification, making it the reference regime. The first column of the transition LO matrix, is populated with the starting values of: (1) `c11` $= 0.7$, corresponding to $exp(0.7) = 2.01$ times greater LO of staying within the Deactivated regime as transitioning to the Activated regime; and (2) `c21` $= -1$, corresponding to $exp(-1) = 0.37$ times lower LO of transitioning to the Deactivated regime.

```
recReg <- prep.regimes(values = matrix(c(0.7, -1, 0, 0), 2, 2),
  params = matrix(c('c11', 'c21', 'fixed', 'fixed'), 2, 2))
```

In essence, the above code creates the following transition probability matrix:

$$\left.\begin{array}{rcc} & Deactivated_{t_{i,j+1}} & Activated_{t_{i,j+1}} \\ \begin{array}{r} Deactivated_{t_{i,j}} \\ Activated_{t_{i,j}} \end{array} & \multicolumn{2}{c}{\left(\begin{array}{cc} \frac{exp(c_{11})}{exp(c_{11})+exp(0)} & \frac{exp(0)}{exp(c_{11})+exp(0)} \\ \frac{exp(c_{21})}{exp(c_{21})+exp(0)} & \frac{exp(0)}{exp(c_{21})+exp(0)} \end{array}\right)} \end{array}\right|_{c_{11}=.7 \quad c_{12}=-1} = \begin{array}{rcc} & D_{t_{i,j+1}} & A_{t_{i,j+1}} \\ \begin{array}{r} D_{t_{i,j}} \\ A_{t_{i,j}} \end{array} & \multicolumn{2}{c}{\left(\begin{array}{cc} 0.668 & 0.332 \\ 0.269 & 0.731 \end{array}\right)} \end{array}. \tag{14}$$

In many situations it is useful to specify the structure of the transition LO matrix in deviation form — that is, to express the LO intercepts in all but the reference regime as deviations from the LO intercept in the reference regime. The package vignette illustrates this by invoking the `deviation` argument of `prep.regimes()`.

After the recipes for all parts of the model are defined, the `dynr.model()` function creates the model and stores it in the `"dynrModel"` object. Each recipe object created by `prep.*()` and the data prepared by `dynr.data()` are given to this function. The `dynr.model()` function always requires `dynamics`, `measurement`, `noise`, `initial`, and `data`. When there are multiple regimes, the `regimes` argument should also be provided. When parameters are subject to transformation functions, a `transform` argument can be added, which will be discussed in the second example. The `dynr.model()` function combines information from the recipes and data to write the text for a **C** function. This text is written to a file optionally named by the `outfile` argument, so that the user can inspect or modify the generated **C** code. The default `outfile` is a temporary file returned by `tempfile()`.

```
rsmod <- dynr.model(dynamics = recDyn, measurement = recMeas,
  noise = recNoise, initial = recIni, regimes = recReg,
  data = EMGdata, outfile = "RSLinearDiscreteYang.c")
yum <- dynr.cook(rsmod)
```

In the last line above, the model is "cooked" with the `dynr.cook()` function to estimate the free parameters and their standard errors. When cooking, the **C** code in the `outfile` is compiled and dynamically linked to the rest of the compiled **dynr** code. If the **C** functions have previously been compiled then the user can prevent re-compilation by setting `compileLib = FALSE` in the `"dynrModel"` object given to `dynr.cook()`. After compilation the **C** code is executed to optimize the free parameters while calling the dynamically linked **C** functions that were created from the user-specified recipes. In this way, **dynr** provides an **R** interface for dynamical systems modeling while maintaining much of the speed associated with **C**.

The final step associated with **dynr** modeling is serving results (a `"dynrCook"` object) after the model has been cooked. To this end, several standard, popular S3 methods are defined for the `"dynrCook"` class, including `coef()`, `confint()`, `deviance()`, `logLik()`, `AIC()`, `BIC()`, `names()`, `nobs()`, `summary()`, and `vcov()`. These methods perform the same tasks as their counterparts for regression models in **R**. Additionally, **dynr** provides a few other model-serving functions illustrated here: `summary()`, `plot()`, `dynr.ggplot()` (or `autoplot()`), `plotFormula()`, and `printex()`. The summary() method provides a table of free parameter names, estimates, standard errors, t-values, and Wald-type confidence intervals.

```
summary(yum)

Coefficients:
         Estimate Std. Error t value ci.lower ci.upper Pr(>|t|)
phi_1     0.26608    0.04953   5.372  0.16900  0.36315 5.33e-08 ***
phi_2     0.47395    0.04425  10.711  0.38722  0.56068  < 2e-16 ***
beta_2    0.46449    0.04394  10.571  0.37837  0.55061  < 2e-16 ***
mu_1      4.55354    0.02782 163.658  4.49901  4.60807  < 2e-16 ***
mu_2      4.74770    0.14250  33.318  4.46842  5.02699  < 2e-16 ***
dynNoise  0.20896    0.01129  18.504  0.18683  0.23110  < 2e-16 ***
c11       5.50199    0.70939   7.756  4.11160  6.89237  < 2e-16 ***
c21      -5.16170    1.00424  -5.140 -7.12998 -3.19342 1.79e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log-likelihood value at convergence = 1002.52
AIC = 1018.52
BIC = 1054.87
```

These parameter estimates, standard errors, and likelihood values closely mirror those reported in Yang and Chow (2010, p. 755-756). In the Deactivated regime, the autoregressive parameter (phi_1) and the intercept (mu_1) are lower than those in the Activated regime. So, neighboring EMG measurements are more closely related in the Activated regime and the overall level is slightly higher. This matches very well with the idea that the Activated regime consists of bursts of facial muscular activities and an elevated emotional state. Similarly, the effect of the self-reported emotional level is positive in the Activated regime and fixed to zero in the Deactivated regime, as the freely estimated value was close to zero with a nonsignificant t-value. So, in the Deactivated regime the self-reported emotional level and the facial muscular activity decouple. The dynamic noise parameter gives a sense of the size of the intrinsic unmeasured disturbances that act on the system. These forces perturb the system with a typical magnitude of a little less than half a point on the EMG scale seen in Figure 1(A). Lastly, the log-odds parameters (c11 and c21) can be turned into the transition probability matrix yielding

$$\begin{array}{r} \\ \mathit{Deactivated}_{t_{i,j}} \\ \mathit{Activated}_{t_{i,j}} \end{array} \begin{array}{c} \begin{array}{cc} \mathit{Deactivated}_{t_{i,j+1}} & \mathit{Activated}_{t_{i,j+1}} \end{array} \\ \left(\begin{array}{cc} 0.9959 & 0.0041 \\ 0.0057 & 0.9943 \end{array} \right). \end{array} \tag{15}$$

which implies that both the Deactivated and the Activated regimes are strongly persistent with high self-transistion probabilities. Next we consider some of the visualization options for serving a model.

The default plot() method is used to visualize the time series in a collection of plots: (1) a plot of time series created by dynr.ggplot() (or autoplot()), (2) a histogram of predicted regimes, and (3) a plot of equations created by plotFormula().

```
plot(yum, dynrModel = rsmod, style = 1, textsize = 5)
```

The dynr.ggplot() (or autoplot()) method creates a plot of the smoothed state estimates with the predicted regimes. It needs the result object and model object as inputs, and allows for plotting (1) user-selected smoothed state variables by default or (2) user-selected observed-versus-predicted values by setting style = 2. An illustrative plot is created from the code below and shown in Figure 1(B).

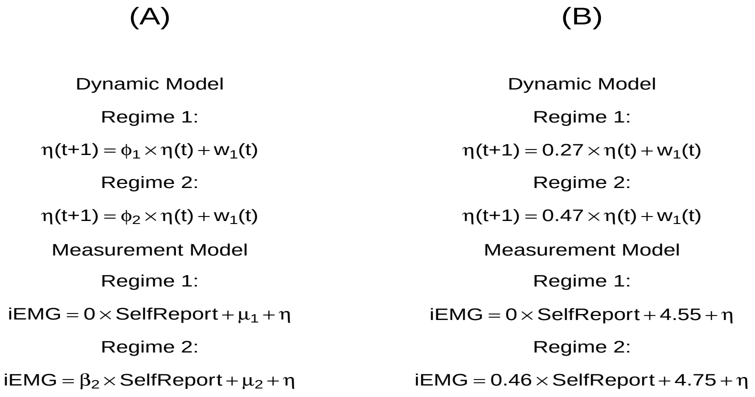
```
dynr.ggplot(yum, dynrModel = rsmod, style = 1,
  names.regime = c("Deactivated", "Activated"),
  title = "(B) Results from RS-AR model", numSubjDemo = 1,
  shape.values = 1, text = element_text(size = 24), is.bw = TRUE)
```

This shows that for the first 99 seconds the participant is in the Deactivated regime, with their latent state $\eta_i(t_{i,j+1})$ varying according to the lower autocorrelation model and having no relation to the variation in the self-reported emotional data in Figure 1(A). Then the participant switches to the Activated regime and their latent state becomes more strongly autocorrelated and coupled to the self-report data. There follows a brief period in the Deactivated regime around time=130 seconds with a subsequent return to the Activated regime for the remainder of the observation. Of course, note that Figure 1(A) shows the observed EMG data whereas Figure 1(B) shows the latent state which is related to the observed data by Equation 11.

The `plotFormula()` method can be used to display model equations on **R** plots. Equations can be viewed in several ways with different inputs to the `ParameterAs` argument: (1) with free parameter names, for example, returned by `names(rsmod)`, as illustrated in Figure 2(A); (2) with parameter starting values; or (3) after estimation with fitted parameter values, for example, returned by `coef(yum)`, as in Figure 2(B). The `plotFormula()` method does not require the user to install LaTeX facilities and compile LaTeX code in a separate step, and hence are convenient to use. To maximize the readability of the equations, it is only shown here using equations for the dynamic and measurement models, which can be obtained by respectively setting the `printDyn` and `printMeas` arguments to true.

```
plotFormula(dynrModel = rsmod, ParameterAs = names(rsmod),
  printDyn = TRUE, printMeas = TRUE) + ggtitle("(A)") +
  theme(plot.title = element_text(hjust = 0.5, vjust = 0.01, size = 16))

plotFormula(dynrModel = rsmod, ParameterAs = coef(yum),
  printDyn = TRUE, printMeas = TRUE) + ggtitle("(B)") +
  theme(plot.title = element_text(hjust = 0.5, vjust = 0.01, size = 16))
```

We can see that the equations in Figure 2(A) are precisely those from Equations 11 and 12 which we used to define the model except that we have fixed β_1 to zero. If these equations did not match, it may indicate that we made a mistake in our model specification.



(A)

Dynamic Model

Regime 1:

$\eta(t+1) = \phi_1 \times \eta(t) + w_1(t)$

Regime 2:

$\eta(t+1) = \phi_2 \times \eta(t) + w_1(t)$

Measurement Model

Regime 1:

$\text{iEMG} = 0 \times \text{SelfReport} + \mu_1 + \eta$

Regime 2:

$\text{iEMG} = \beta_2 \times \text{SelfReport} + \mu_2 + \eta$

(B)

Dynamic Model

Regime 1:

$\eta(t+1) = 0.27 \times \eta(t) + w_1(t)$

Regime 2:

$\eta(t+1) = 0.47 \times \eta(t) + w_1(t)$

Measurement Model

Regime 1:

$\text{iEMG} = 0 \times \text{SelfReport} + 4.55 + \eta$

Regime 2:

$\text{iEMG} = 0.46 \times \text{SelfReport} + 4.75 + \eta$

Figure 2: Automatic plots of model equations with (A) parameter names and (B) estimated parameters for the regime-switching linear state-space model.

Finally, for LaTeX users, the `printex()` method helps generate equations for the model in LaTeX form.

```
printex(rsmod, ParameterAs = names(rsmod), printInit = TRUE, printRS = TRUE,
  outFile = "RSLinearDiscreteYang.tex")
```

The `ParameterAs` argument functions the same as that in the `plotFormula()` method. Here we have specified to use the names of the free parameters. In this case, the initial conditions and regime-switching functions are included in the equations, as indicated by the `printInit` and `printRS` arguments being set to `TRUE`. The LaTeX code for the equations is written to 'RSLinearDiscreteYang.tex', which the user can then work with and modify as they wish. Of course, this function is designed more as a convenience feature for users who already use LaTeX and requires all the LaTeX-related facilities on the user's computer.

This example has used real EMG data from a previous study (Yang and Chow, 2010) to illustrate many parts of the user-interface for **dynr**. Of particular note are the various "serving" functions which allow users to both verify their model and examine their results in presentation-ready formats. In the next example, we will use simulated data to further illustrate features of **dynr**, especially the nonlinear formula interface for dynamics.

Example 2: Nonlinear continuous-time models

In the study of human dynamics many processes are characterized by changes that are dependent on interactions with other processes producing dynamics with nonlinearities. Nonlinear ordinary differential equations have been used to model, among other phenomena, ovulatory regulation (Boker et al., 2014), circadian rhythms (Brown and Luithardt, 1999), cerebral development (Thatcher, 1998), substance use (Boker and Graham, 1998), cognitive aging (Chow and Nesselroade, 2004), parent-child interactions (Thomas and Martin, 1976), couple dynamics (Chow et al., 2007; Gottman, 2002), and sudden transitions in attitudes (van der Maas et al., 2003).

Single-regime nonlinear continuous-time model

In addition to the linear/matrix dynamics interface, **dynr** also provides users with a formula interface to accommodate nonlinear as well as linear dynamic functions. To illustrate the use of the formula interface in **dynr**, we use a benchmark nonlinear ordinary differential equation model, the predator-prey model (Lotka, 1925; Volterra, 1926; Hofbauer and Sigmund, 1988). One can find the complete demo scripts in **dynr**, using `file.edit(system.file("demo","NonlinearODE.R",package = "dynr"))` and `file.edit(system.file("demo","RSNonlinearODE.R",package = "dynr"))`, and related explanation in the package vignette 'NonlinearContinuousTimeModels'.

The predator-prey model is a classic model for representing the nonlinear dynamics of interacting populations. The most often cited behavior of the predator-prey system while in a particular parameter range is ongoing nonlinear oscillations in the predator and prey populations with a phase lag between them. The utility of the predator-prey model extends far beyond the area of population dynamics. Direct applications or extensions of this predator-prey system include the epidemic models of the onset of social activities (EMOSA) used to study the spread of smoking, drinking, delinquency, and sexual behaviors among adolescents (Rodgers and Rowe, 1993; Rodgers et al., 1998); the cognitive aging model (Chow and Nesselroade, 2004); and the model of couples' affect dynamics (Chow et al., 2007).

Written as a differential equation, the predator-prey model is expressed as:

$$d(prey(t)) = (a\ prey(t) - b\ prey(t)\ predator(t))\,dt, \tag{16}$$

$$d(predator(t)) = (-c\ predator(t) + d\ prey(t)\ predator(t))\,dt, \tag{17}$$

where the parameters a, b, c, d are all nonnegative. These equations make up the continuous-time dynamics, Equation 1, for this system. Examining the prey equation (Equation 16), the prey population would increase exponentially without bound if there were zero predators. Similarly, examining the predator equation (Equation 17), if the prey population was zero, then the predator population would decrease exponentially to zero. For demonstration purposes, we have included with the **dynr** package a set of simulated data generated with true parameter values: $a = 2, b = 1, c = 4, d = 1, e = .25, f = 5$.

Using the formula interface in **dynr**, which supports all native mathematical functions available in **R**, the predator-prey model can be specified as:

```
preyFormula <- prey ~ a * prey - b * prey * predator
predFormula <- predator ~ - c * predator + d * prey * predator
ppFormula <- list(preyFormula, predFormula)
ppDynamics <- prep.formulaDynamics(formula = ppFormula,
  startval = c(a = 2.1, c = 0.8, b = 1.9, d = 1.1), isContinuousTime = TRUE)
```

The first argument of the `prep.formulaDynamics()` function is `formula`. More specifically, this is a list of formulas. Each element in the list is a single, univariate, formula that defines a differential (if `isContinuousTime` = TRUE) or difference (if `isContinuousTime` = FALSE) equation. There should be one formula for every latent variable, in the order in which the latent variables are specified by using the `state.names` argument in `prep.measurement()`. The left-hand side of each formula is either the one-step-ahead projection or the differential of the latent variable: namely, the left-hand side of Equations 1 and 3, respectively. In both cases, users only need to specify the names of the latent variables that match the specification in `prep.measurement()` on the left-hand side of the formulas. The right-hand side of each formula gives a linear or nonlinear function that may involve free or fixed parameters, numerical constants, exogenous covariates, and other arithmetic/mathematical functions that define the dynamics of the latent variables. The `startval` argument is a named vector giving the names of the free parameters and their starting values. Just as in the `prep.matrixDynamics()` function, the `isContinuousTime` argument is a binary flag that switches between continuous- and discrete-time modeling. The rest of **dynr** code for fitting the predator-prey model can be specified in similar ways to the code shown in Example 1 and is omitted here for space constraints. A fully functional demo script can be found in **dynr**, using `file.edit(system.file("demo","NonlinearODE.R",package = "dynr"))`, and further comments are included as a package vignette.

With the formula interface, **dynr** uses the `D()` function to symbolically differentiate the formulas provided. Hence, **dynr** uses the analytic Jacobian of the dynamics in its extended Kalman filter, greatly increasing its speed and accuracy. The `D()` function can handle the differentiation of functions involving parentheses, arithmetic operators, for instance, +, −, ∗, /, and ˆ, and numerous mathematical functions such as `exp()`, `log()`, `sin()`, `cos()`, `tan()`, `sinh()`, `cosh()`, `sqrt()`, `pnorm()`, `dnorm()`, `asin()`, `acos()`, `atan()`, and `gamma()`. Thus, for a very large class of nonlinear functions, the user is spared from supplying the analytic Jacobian of the dynamic functions. However, symbolic differentiation will not work for all formulas. For instance, formulas involving the absolute value function cannot be symbolically differentiated. For formulas that cannot be differentiated symbolically, the user must provide the analytic first derivatives through the `jacobian` argument. One can use `file.edit(system.file("demo","RSNonlinearDiscrete.R",package = "dynr"))` to find an example. An explanation is also included as a package vignette.

Regime-switching extension

Just as with the `prep.matrixDynamics()`, the formula interface also allows for regime-switching functionality. Consider an extension of the classical predator-prey model that lets the prey and predator interaction follow seasonal patterns. In the Summer regime, we have the predator-prey model as previously described, but in the Winter regime we now have a predator-prey model characterized by within-species competition and limiting growth/decay. In this competitive predator-prey model, the two populations do not grow/decline exponentially without bound in absence of the other, but rather, they grow logistically up to some finite carrying capacity. This logistic growth adds to the between-species interactions with the other population. This model can be specified as:

```
cPreyF <- prey ~ a * prey - e * prey ^ 2 - b * prey * predator
cPredF <- predator ~ f * predator - c * predator ^ 2 + d * prey * predator
cpFormula <- list(cPreyF, cPredF)
```

where the predator and prey equations are combined and supplied as a list.

To specify the regime-switching predator-prey model, we combine the classical predator-prey model and the predator-prey model with within-species competition into a list of lists. Then we provide this list to the usual `prep.formulaDynamics()` function as the `formula` argument.

```
rsFormula <- list(ppFormula, cpFormula)
dynm <- prep.formulaDynamics(formula = rsFormula,
   startval = c(a = 2.1, c = 3, b = 1.2, d = 1.2, e = 1, f = 2),
   isContinuousTime = TRUE)
```

Many dynamic models only lead to permissible values in particular parameter ranges. As such, we often need to add box constraints to model parameters. This is accomplished by setting bounds on the parameters as shown in the next section. An alternative in **dynr** is to apply unconstrained optimization to a transformed set of parameters. This latter strategy uses `prep.tfun()`. For example, the $a - f$ parameters should take on positive values. Thus, we may choose to optimize their log-transformed values and exponentiate the unconstrained parameter values during likelihood evaluations to ensure that their values are always positive. To achieve this, we supply a list of transformation formulas to the `formula.trans` argument in the prep.tfun() function as follows:

```
tformList <- list(a ~ exp(a), b ~ exp(b), c ~ exp(c),
  d ~ exp(d), e ~ exp(e), f ~ exp(f))
tformInvList <- list(a ~ log(a), b ~ log(b), c ~ log(c),
  d ~ log(d), e ~ log(e), f ~ log(f))
trans <- prep.tfun(formula.trans = tformList, formula.inv = tformInvList)
```

In cases involving transformation functions, the delta method is used to yield standard error estimates for the parameters on the constrained scales. If the starting values of certain parameters are indicated on a constrained scale, the `formula.inv` argument should then give a list of inverse transformation formulas.

In our hypothetical example, we have discussed how the weather condition may govern the regime switching processes. Specifically, we assume a covariate `cond` (with a value of 0 indicating the warmer weather and 1 indicating the colder weather) has an effect on the regime-switching transition probabilities. Then, we can specify the logistic regression model by

```
regimes <- prep.regimes(
  values = matrix(c(0, 0, -1, 1.5, 0, 0, -1, 1.5), nrow = 2, ncol = 4, byrow = TRUE),
  params = matrix(c("fixed", "fixed", "int_1", "slp_1",
  "fixed", "fixed", "int_2", "slp_2"), nrow = 2, ncol = 4, byrow = TRUE),
  covariates = "cond")
```

In essence, the above code creates a matrix in the following form:

$$\left(\begin{array}{c} c_{11} = 0 \; d_{11} = 0 \; c_{12} = int_1 = -1 \; d_{12} = slp_1 = 1.5 \\ \hdashline c_{21} = 0 \; d_{21} = 0 \; c_{22} = int_2 = -1 \; d_{22} = slp_2 = 1.5 \end{array}\right), \tag{18}$$

which in turn creates the following transition probability matrix:

$$\begin{array}{ccc} & Summer_{t_{i,j+1}} & Winter_{t_{i,j+1}} \\ Summer_{t_{i,j}} & \frac{exp(0+0\times cond)}{exp(0+0\times cond)+exp(int_1+slp_1\times cond)} & \frac{exp(int_1+slp_1\times cond)}{exp(0+0\times cond)+exp(int_1+slp_1\times cond)} \\ Winter_{t_{i,j}} & \frac{exp(0+0\times cond)}{exp(0+0\times cond)+exp(int_2+slp_2\times cond)} & \frac{exp(int_2+slp_2\times cond)}{exp(0)+exp(int_2+slp_2\times cond)} \end{array}. \tag{19}$$

Here we consider the Summer regime as the reference regime, so the first two columns of the transition LO matrix (Equation 18) are fixed at zero. The third and fourth columns of the transition LO matrix respectively correspond to the regression intercepts and slopes associated with the covariate, whose starting values are respectively set at -1 and 1.5. With this set of starting values, the transition probability from any regime to the Summer regime is 0.73 when `cond` $= 0$, and 0.38 when `cond` $= 1$. The negative intercept implies that in warmer days (`cond` $= 0$), there is a greater chance of the process transitioning into the Summer regime, and the regression slope greater than the absolute value of the intercept suggests that in colder days (`cond` $= 1$), the transition into the Winter regime is more likely.

We fitted the specified model to the simulated data. Figure 3 is created from the `dynr.ggplot()` (or `autoplot()`) method with `style = 2`, and shows that the predicted trajectories match with the observed values and alternate between different regimes.

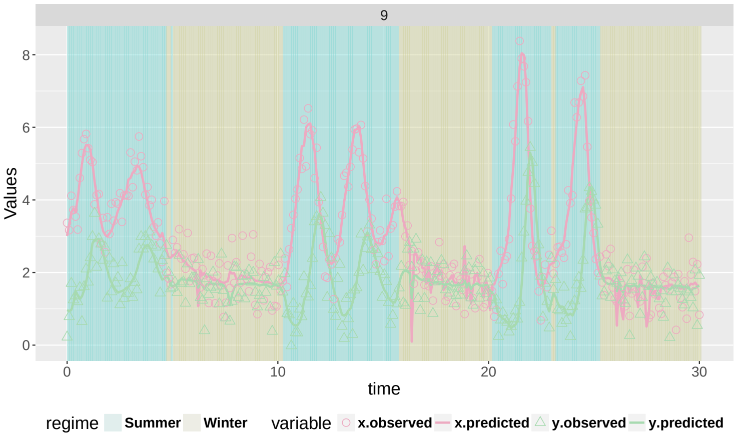


Figure 3: Built-in plotting feature for the predicted trajectories with observed values for the regime-switching nonlinear ODE model.

Other miscellaneous control options

In parameter estimation, **dynr** utilizes a sequential quadratic programming algorithm (Kraft, 1988, 1994) available from an open-source library for nonlinear optimization — NLOPT (Johnson, 2008). By default, we do not set boundaries on the free parameters. However, one can set the upper and lower bounds by respectively modifying the `ub` and `lb` slots of the model object. An example is given below to constrain the `int_1` and `int_2` parameters to be between -10 and 0, while limiting `slp_1` and `slp_2` to be between 0 to 10:

```
model2.2$ub[ c("int_1", "int_2", "slp_1", "slp_2") ] <- c(0, 0, 10, 10)
model2.2$lb[ c("int_1", "int_2", "slp_1", "slp_2") ] <- c(-10, -10, 0, 0)
```

Similarly, the stopping criteria of the optimization can be modified through the `options` slot of the `"dynrModel"` object, which is a list consisting of the relative tolerance on optimization parameters xtol_rel; the stopping threshold of the objective value stopval; the absolute and relative tolerance on function value, ftol_abs and ftol_rel; the maximum number of function evaluations maxeval; and the maximum optimization time maxtime.

If there is no need to re-compile the **C** functions in a call to dynr.cook(), the user can change the compileLib slot of the "dynrModel" object from default true to false. The output of the estimation function, dynr.cook(), is an object of class "dynrCook". It not only includes estimation results displayed with summary(), but also contains information on posterior regime probabilities in the pr_t_given_T slot, smoothed state estimates $\hat{\boldsymbol{\eta}}_i(t_{i,j}|T_i) = E(\boldsymbol{\eta}_i(t_{i,j})|\mathbf{Y}_i(T_i))$ of the latent variables in the eta_smooth_final slot, and smoothed error covariance matrices $\mathbf{P}_i(t_{i,j}|T_i)$ of the latent variables in the error_cov_smooth_final slot, at all available time points. They can be retrieved by using the $ operator.

Discussion and conclusions

This paper has introduced the **dynr** package that attempts to carefully balance intuitive usability with flexibility in the specification to satisfy the need of the broad social and behavioral science community. **dynr** offers linear and nonlinear time series methods for latent variables in both the traditional discrete-time models and in the hybrid continuous-time models that have discrete measurements with continuous underlying processes. Moreover, regime-switching can be layered on top of any aspect of these models.

Even though **dynr** can specify some models that other programs cannot, all of the features of other programs that exist for time series modeling are not subsets of **dynr**. For example, **KFAS** allows for nonlinear measurement (Helske, 2017a) which is not currently possible in **dynr**. Moreover, **SsfPack** has nonlinear measurement capabilities along with many MCMC methods that **dynr** lacks (Koopman et al., 1999). The **pomp** package has also implemented several algorithms absent in **dynr**, including MCMC methods, Bayesian methods, particle filtering, as well as ensemble filtering and forecasting. However, to our knowledge, no other software allows for regime-switching nonlinear dynamics with latent variables.

The **dynr** package highlighted the use of *recipe* objects to prepare components of the model. The recipes divide the full model into meaningful conceptual chunks for ease of specification and interactive inspection. The recipes seamlessly handle various bookkeeping tasks like the creation and management of the free parameter vector and how free parameters map onto model components. This is in contrast to several other packages offload this management on the user, often writing their own functions in the process. In addition to sparing the user sundry bothersome tasks, the recipes allow for interactive error checking and model verification using standard commands that should already be familiar to users of **R**. The contents of each recipe can be printed in the **R** console, letting the user verify that the recipe they intended to specify was actually created. Along this vein, plotFormula() allows the user to see nicely formatted equations for their models directly in **R**, and printex() outputs LaTeX equations for their models which can be typeset immediately or modified for inclusion in manuscripts, presentations, and reports.

The **dynr** package critically depends on several data structures and methods from the GNU Scientific Library (GSL; GSL Project Contributors, 2010) for fast and accurate scientific computing, and consequently requires the user to install GSL on their system. We wanted to allow users the flexibility of specifying their own models, while not sacrificing computational speed that would be influenced by frequent interchanges between **R** and **C** functions. Thus, **dynr** requires that users generate and compile the **C** code "on the fly", and pass the **C** function pointers to the back-end directly. Hence, **dynr** has a nontrivial set-up cost as compared to other **R** packages. However, to alleviate this burden we have written an installation and configuration guide as a vignette labeled 'InstallationGuideForUsers'. We generally find set-up of **dynr** to be similar to that of other packages that allow "on the fly" compilation and ready C interfaces like **Rcpp** (Eddelbuettel, 2013; Eddelbuettel et al., 2018) and **RcppGSL** (Eddelbuettel and Francois, 2018).

Alternative computational strategies tended to worsen performance, increase user burden for model specification, or simply trade one difficult configuration task for another. In **dynr** the user only needs to specify a possibly nonlinear model of interest using standard **R** syntax. By contrast with **Rcpp**/**RcppGSL** the user would have to write **C** functions and hand differentiate their nonlinear dynamics functions: an error-prone process with a much steeper learning curve that acts as a deterrent to adoption, particularly to many researchers in the social and behavioral sciences. Additionally, we have found that automatic generation of a model specification file coded in **C** provides more sophisticated users with the opportunity to define modeling variations directly in **C** that are not already supported by the **R** interface functions.

Currently **dynr** only allows nonlinearity in the dynamics but not the measurement model to capitalize on the availability of a Gaussian approximate log-likelihood function for fast parameter estimation. Future extensions will incorporate Markov chain Monte Carlo (MCMC) techniques (Chow et al., 2011; Durbin and Koopman, 2001; Kim and Nelson, 1999; Lu et al., 2015) and pertinent frequentist-based estimation techniques (Fahrmeir and Tutz, 1994) to accommodate a broader class of measurement models consisting of nonlinear functions and non-Gaussian densities. In addition, several other extensions are being pursued and implemented in the **dynr** package. For example, **dynr** currently handles missingness in the dependent variables via full-information maximum likelihood but does not allow for missingness in the covariates. Future plans include interfacing **dynr** with **R** packages such as **mice** (van Buuren and Groothuis-Oudshoorn, 2011, 2017) to handle missingness in the covariates and/or dependent variables via multiple imputation. Further, models with nonlinearities at the dynamic level currently are not supported by well-established fit indices. Although **dynr** provides AIC (Akaike, 1973) and BIC (Schwarz, 1978) for model comparison purposes, the tenability of using these criteria when nonlinearities at the dynamic level are present and the optimized log-likelihood function involves approximations and truncation errors is yet to be investigated. Finally, even though difference and differential equations have served as and remain one of the most popular modeling tools across myriad scientific disciplines, their use is still nascent in many social and behavioral sciences. Tools to aid model developments and explorations are important extensions to enable and promote modeling efforts utilizing difference/differential equations (Chow et al., 2016; Ramsay et al., 2009). Fortunately, several existing packages in **R** offer many of the functionalities to support these modeling endeavors and may be used in conjunction or interfaced in the future with **dynr** for these purposes.

Bibliography

H. Akaike. Information theory and an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki, editors, *Second International Symposium on Information Theory*, pages 267–281. Akademiai Kiado, Budapest, 1973. [p94, 105]

B. D. O. Anderson and J. B. Moore. *Optimal Filtering*. Prentice Hall, Englewood Cliffs, NJ, 1979. [p94, 95, 111]

C. F. Ansley and R. Kohn. Estimation, filtering and smoothing in state space models with incompletely specified initial conditions. *The Annals of Statistics*, 13:1286–1316, 1985. [p95]

D. Ardia, K. Bluteau, K. Boudt, L. Catania, B. Peterson, and D.-A. Trottier. *Markov-Switching GARCH Models in* ***R****: The MSGARCH*, 2017. R package version 1.3. [p92]

Y. Bar-Shalom, X. R. Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. John Wiley & Sons, New York, NY, 2001. [p94, 95, 111]

S. M. Boker and J. Graham. A dynamical systems analysis of adolescent substance abuse. *Multivariate Behavioral Research*, 33(4):479–507, 1998. URL https://doi.org/10.1207/s15327906mbr3304_3. [p101]

S. M. Boker, M. C. Neale, and K. L. Klump. A differenial equations model for the ovarian hormone cycle. In P. C. M. Molenaar, R. M. Lerner, and K. M. Newell, editors, *Handbook of Developmental Systems Theory and Methodology*, pages 369–391. Guilford Press, New York, NY, 2014. [p101]

S. M. Boker, M. C. Neale, H. H. Maes, M. Spiegel, T. R. Brick, R. Estabrook, T. C. Bates, R. J. Gore, M. D. Hunter, J. N. Pritikin, M. Zahery, and R. M. Kirkpatrick. *OpenMx: Extended Structural Equation Modelling*, 2017. URL https://CRAN.R-project.org/package=OpenMx. R package version 2.8.3. [p91]

N. Bolger and J.-P. Laurenceau. *Intensive Longitudinal Methods: An Introduction to Diary and Experience Sampling Research*. Guilford Press, New York, NY, 2013. [p91]

P. Brandt. *MSBVAR: Markov-Switching, Bayesian, Vector Autoregression Models*, 2016. URL https://CRAN.R-project.org/package=MSBVAR. **R** package version 0.9-3. [p92]

E. N. Brown and H. Luithardt. Statistical model building and model criticism for human circadian data. *Journal of Biological Rhythms*, 14:609–616, 1999. URL https://doi.org/10.1177/074873099129000975. [p101]

B. Byrom and B. Tiplady. *ePRO: Electronic Solutions for Patient-Reported Data*. Gower, Farnham, England, 2010. [p91]

J. T. Cacioppo and R. E. Petty. Electromyograms as measures of extent and affectivity of information processing. *American Psychologist*, 36:441–456, 1981. [p96]

J. T. Cacioppo, R. E. Petty, M. E. Losch, and H. S. Kim. Electromyographic activity over facial muscle regions can differentiate the valence and intensity of affective reactions. *Journal of Personality and Social Psychology*, 50(2):260–268, 1986. [p96]

S.-M. Chow and J. R. Nesselroade. General slowing or decreased inhibition? Mathematical models of age differences in cognitive functioning. *Journals of Gerontology B*, 59(3):101–109, 2004. [p101]

S.-M. Chow and G. Zhang. Nonlinear regime-switching state-space (RSSS) models. *Psychometrika*, 78(4):740–768, 2013. URL https://doi.org/10.1007/s11336-013-9330-8. [p91, 94, 95, 111]

S.-M. Chow, E. Ferrer, and J. R. Nesselroade. An unscented Kalman filter approach to the estimation of nonlinear dynamical systems models. *Multivariate Behavioral Research*, 42(2):283–321, 2007. URL https://doi.org/10.1080/00273170701360423. [p95, 101]

S.-M. Chow, M.-H. R. Ho, E. J. Hamaker, and C. V. Dolan. Equivalences and differences between structural equation and state-space modeling frameworks. *Structural Equation Modeling*, 17: 303–332, 2010. URL https://doi.org/10.1080/10705511003661553. [p94]

S.-M. Chow, N. Tang, Y. Yuan, X. Song, and H. Zhu. Bayesian estimation of semiparametric nonlinear dynamic factor analysis models using the Dirichlet process prior. *British Journal of Mathematical and Statistical Psychology*, 64(1):69–106, 2011. URL https://doi.org/10.1348/000711010x497262. [p105]

S.-M. Chow, K. J. Grimm, F. Guillaume, C. V. Dolan, and J. J. McArdle. Regime-switching bivariate dual change score model. *Multivariate Behavioral Research*, 48(4):463–502, 2013. URL https://doi.org/10.1080/00273171.2013.787870. [p91]

S.-M. Chow, K. Witkiewitz, R. P. P. P. Grasman, and S. A. Maisto. The cusp catastrophe model as cross-sectional and longitudinal mixture structural equation models. *Psychological Methods*, 20: 142–164, 2015. URL https://doi.org/10.1037/a0038962. [p91]

S.-M. Chow, J. J. Bendezú, P. M. Cole, and N. Ram. A comparison of two-stage approaches for fitting nonlinear ordinary differential equation (ode) models with mixed effects. *Multivariate Behavioral Research*, 51(2–3):154–184, 2016. URL https://doi.org/10.1080/00273171.2015.1123138. [p105]

S.-M. Chow, L. Ou, A. Ciptadi, E. Prince, D. You, M. D. Hunter, J. M. Rehg, A. Rozga, and D. S. Messinger. Representing sudden shifts in intensive dyadic interaction data using differential equation models with regime switching. *Psychometrika*, 83(2):476–510, 2018. URL https://doi.org/10.1007/s11336-018-9605-1. [p94, 96, 111]

P. De Jong. The likelihood for a state space model. *Biometrika*, 75(1):165–169, 1988. URL https://doi.org/10.2307/2336450. [p94]

U. Dimberg, M. Thunberg, and K. Elmehed. Unconscious facial reactions to emotional facial expressions. *Psychological Science*, 11(1):86–89, 2000. URL https://doi.org/10.1111/1467-9280.00221. [p96]

C. V. Dolan. *MKFM6: Multi-Group, Multi-Subject Stationary Time Series Modeling Based on the Kalman Filter*, 2005. URL http://users/fmg.uva.nl/cdolan/. [p91]

C. V. Dolan. Structural equation mixture modeling. In R. E. Millsap and A. Maydeu-Olivares, editors, *The SAGE Handbook of Quantitative Methods in Psychology*, pages 568–592. Sage, Thousand Oaks, CA, 2009. [p91]

C. V. Dolan, B. R. Jansen, and H. L. J. Van der Maas. Constrained and unconstrained multivariate normal finite mixture modeling of piagetian data. *Multivariate Behavioral Research*, 39(1):69–98, 2004. URL https://doi.org/10.1207/s15327906mbr3901_3. [p91]

C. Driver, M. Voelkle, and H. Oud. *Ctsem: Continuous Time Structural Equation Modelling*, 2017a. URL https://CRAN.R-project.org/package=ctsem. R package version 2.5.0. [p92]

C. C. Driver, J. H. L. Oud, and M. C. Voelkle. Continuous time structural equation modelling with **R** package ctsem. *Journal of Statistical Software*, 2017b. URL https://doi.org/10.18637/jss.v077.i05. [p92]

J. Durbin and S. J. Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, Oxford, United Kingdom, 2001. [p92, 93, 105]

D. Eddelbuettel and R. Francois. Rcpp: Seamless **R** and **C++** integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL `https://doi.org/10.18637/jss.v040.i08`. [p104, 212]

D. Eddelbuettel and R. Francois. *RcppGSL: 'Rcpp' Integration for 'GNU GSL' Vectors and Matrices*, 2018. URL `https://CRAN.R-project.org/package=RcppGSL`. R package version 0.3.6. [p104]

D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2018. URL `https://CRAN.R-project.org/package=Rcpp`. R package version 1.0.0. [p104]

R. J. Elliott, L. Aggoun, and J. B. Moore. *Hidden Markov Models: Estimation and Control*. Springer-Verlag, New York, 1995. [p91]

L. Fahrmeir and G. Tutz. *Multivariate Statistical Modelling Based on Generalized Linear Models*. Springer-Verlag, New York, NY, 1994. [p105]

K. Fukuda and K. Ishihara. Development of human sleep and wakefulness rhythm during the first six months of life: Discontinuous changes at the 7th and 12th week after birth. *Biological Rhythm Research*, 28:94–103, 1997. URL `https://doi.org/10.1076/brhm.28.3.5.94.13132`. [p91]

P. Gilbert. *Dse: Dynamic Systems Estimation (Time Series Package)*, 2015. URL `https://CRAN.R-project.org/package=dse`. R package version 2015.12-1. [p91]

P. D. Gilbert. *Brief User's Guide: Dynamic Systems Estimation*, 2006 or later. URL `http://cran.r-project.org/web/packages/dse/vignettes/Guide.pdf`. [p91]

J. M. Gottman. *The Mathematics of Marriage: Dynamic Nonlinear Models*. The MIT Press, Cambridge, MA, 2002. [p101]

M. S. Grewal and A. P. Andrews. *Kalman Filtering: Theory and Practice Using **MATLAB***. John Wiley & Sons, Hoboken, NJ, 3rd edition, 2008. [p92]

GSL Project Contributors. *GSL - GNU Scientific Library - GNU Project - Free Software Foundation (FSF)*, 2010. URL `http://www.gnu.org/software/gsl/`. [p104]

J. D. Hamilton. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica*, 57:357–384, 1989. URL `https://doi.org/10.2307/1912559`. [p91, 95]

J. D. Hamilton. *Time Series Analysis*. Princeton University Press, Princeton, NJ, 1994. [p94]

A. C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, Cambridge, United Kingdom, 1989. [p91, 94]

J. Helske. KFAS: Exponential family state space models in **R**. *Journal of Statistical Software*, 2017a. URL `https://doi.org/10.18637/jss.v078.i10`. [p91, 92, 104]

J. Helske. *KFAS: Kalman Filter and Smoother for Exponential Family State Space Models*, 2017b. URL `https://CRAN.R-project.org/package=KFAS`. R package version 1.2.9. [p91]

J. Hofbauer and K. Sigmund. *The Theory of Evolution and Dynamical Systems: Mathematical Aspects of Selection (London Mathematical Society Student Texts)*. Cambridge University Press, 1988. ISBN 0521358388. URL `http://www.worldcat.org/isbn/0521358388`. [p101]

B. Hosenfeld. Indicators of discontinuous change in the development of analogical reasoning. *Journal of Experimental Child Psychology*, 64:367–395, 1997. URL `https://doi.org/10.1006/jecp.1996.2351`. [p91]

M. D. Hunter. State space modeling in an open source, modular, structural equation modeling environment. *Structural Equation Modeling: A Multidisciplinary Journal*, pages 1–18, 2017. URL `https://doi.org/10.1080/10705511.2017.1369354`. [p91]

R. J. Hyndman. CRAN task view: Time series analysis. Online, 2016. URL `https://CRAN.R-project.org/view=TimeSeries`. Accessed on October 09, 2016. [p91]

S. G. Johnson. *The NLopt Nonlinear-Optimization Package*, 2008. URL `http://ab-initio.mit.edu/nlopt`. [p103]

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. URL https://doi.org/10.1115/1.3662552. [p94, 111]

C.-J. Kim and C. R. Nelson. *State-Space Models with Regime Switching: Classical and Gibbs-Sampling Approaches with Applications.* MIT Press, Cambridge, MA, 1999. [p91, 94, 95, 105, 111]

A. A. King, D. Nguyen, and E. L. Ionides. Statistical inference for partially observed Markov processes via the R package pomp. *Journal of Statistical Software*, 69(12):1–43, 2016. URL https://doi.org/10.18637/jss.v069.i12. [p92]

A. A. King, E. L. Ionides, and C. Breto. *Pomp: Statistical Inference for Partially Observed Markov Processes*, 2018. URL https://CRAN.R-project.org/package=pomp. R package version 1.18. [p92]

L. Kohlberg and R. Kramer. Continuities and discontinuities in childhood and adult moral development. *Human development*, 12(2):93–120, 1969. URL https://doi.org/10.1159/000270857. [p91]

S. J. Koopman, N. Shephard, and J. A. Doornik. Statistical algorithms for models in state space using SsfPack 2.2. *Econometrics Journal*, 2(1):113–166, 1999. URL https://doi.org/10.1111/1368-423x.00023. [p92, 104]

D. Kraft. A software package for sequential quadratic programming. Technical Report 88-28, DFVLR-FB, Oberpfaffenhofen, Germany, 1988. [p92, 103]

D. Kraft. Algorithm 733: TOMP — **Fortran** Modules for Optimal Control Calculations. *ACM Transactions on Mathematical Software*, 20(3):262–281, 1994. URL https://doi.org/10.1145/192115.192124. [p92, 103]

G. Y. Kulikov and M. V. Kulikova. Accurate numerical implementation of the continuous-discrete extended kalman filter. *IEEE Transactions on Automatic Control*, 59(1), 2014. URL https://doi.org/10.1109/tac.2013.2272136. [p94, 95]

M. V. Kulikova and G. Y. Kulikov. Adaptive ODE Solvers in Extended Kalman Filtering Algorithms. *Journal of Computational and Applied Mathematics*, 262:205–216, 2014. URL https://doi.org/10.1016/j.cam.2013.09.064. [p94, 95]

A. J. Lotka. *Elements of Physical Biology.* Williams & Wilkins, Baltimore, MD, 1925. [p96, 101]

Z.-H. Lu, S.-M. Chow, A. Sherwood, and H. Zhu. Bayesian analysis of ambulatory cardiovascular dynamics with application to irregularly spaced sparse data. *Annals of Applied Statistics*, 9: 1601–1620, 2015. URL https://doi.org/10.1214/15-aoas846. [p105]

B. O. Muthén and T. Asparouhov. LTA in **Mplus**: Transition probabilities influenced by covariates. **Mplus** Web Notes: No. 13., 2011. URL http://www.statmodel.com/examples/{LTA}webnote.pdf. [p91]

M. C. Neale, M. D. Hunter, J. N. Pritikin, M. Zahery, T. R. Brick, R. M. Kirkpatrick, R. Estabrook, T. C. Bates, H. H. Maes, and S. M. Boker. OpenMx 2.0: Extended structural equation and statistical modeling. *Psychometrika*, 80(2):535–549, 2016. URL https://doi.org/10.1007/s11336-014-9435-8. [p91]

L. Ou, M. D. Hunter, and S.-M. Chow. *Dynr: Dynamic Modeling in R*, 2018. R package version 0.1.13-4. [p91]

G. Petris. An **R** package for dynamic linear models. *Journal of Statistical Software*, 36(12):1–16, 2010. URL https://doi.org/10.18637/jss.v036.i12. [p91]

G. Petris. *Dlm: Bayesian and Likelihood Analysis of Dynamic Linear Models*, 2014. URL https://CRAN.R-project.org/package=dlm. R package version 1.1-4. [p91]

G. Petris and S. Petrone. State space models in **R**. *Journal of Statistical Software*, 41(4):1–25, 2011. URL https://doi.org/10.18637/jss.v041.i04. [p91]

J. Piaget and B. Inhelder. *The Psychology of the Child.* Basic Books, New York, NY, 1969. [p91]

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in **C**.* Cambridge University Press, Cambridge, 2002. [p95]

J. O. Ramsay, G. Hooker, and S. Graves. *Functional Data Analysis with **R** and **MATLAB***. Springer-Verlag, New York, NY, 2009. [p105]

J. L. Rodgers and D. C. Rowe. Social contagion and adolescent sexual behavior: A developmental EMOSA model. *Psychological Review*, 100(3):479–510, 1993. URL `https://doi.org/10.1037/0033-295x.100.3.479`. [p101]

J. L. Rodgers, D. C. Rowe, and M. Buster. Social contagion, adolescent sexual behavior, and pregnancy: a nonlinear dynamic EMOSA model. *Developmental Psychology*, 34(5):1096–1113, 1998. URL `https://doi.org/10.1037/0012-1649.34.5.1096`. [p101]

J. A. Sanchez-Espigares and A. Lopez-Moreno. *MSwM: Fitting Markov Switching Models*, 2014. URL `https://CRAN.R-project.org/package=MSwM`. **R** package version 1.2. [p92]

G. E. Schwartz. Biofeedback, self-regulation, and the patterning of physiological processes. *American Scientist*, 63:314–324, 1975. [p96]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. [p94, 105]

A. Stone, S. Shiffman, A. Atienza, and L. Nebeling. *The Science of Real-Time Data Capture: Self-Reports in Health Research.* Oxford University Press, NY, 2008. [p91]

O. Taramasco and S. Bauer. *Hidden Markov Models Simulations and Estimations*, 2012. **R** package version 2.0.2. [p92]

R. W. Thatcher. A predator-prey model of human cerebral development. In K. M. Newell and P. C. M. Molenaar, editors, *Applications of Nonlinear Dynamics to Developmental Process Modeling*, pages 87–128. Lawrence Erlbaum, Mahwah, NJ, 1998. [p101]

The MathWorks, Inc. ***MATLAB** Version 9.1 (R2016b).* The MathWorks, Inc., Natick, MA, 2016. [p92]

E. A. Thomas and J. A. Martin. Analyses of parent-infant interaction. *Psychological Review*, 83(2): 141–156, 1976. URL `https://doi.org/10.1037/0033-295x.83.2.141`. [p101]

G. C. Tiao and R. S. Tsay. Some advances in non-linear and adaptive modelling in time series. *Journal of Forecasting*, 13:109–131, 1994. URL `https://doi.org/10.1002/for.3980130206`. [p91]

H. Tong and K. S. Lim. Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society B*, 42:245–292, 1980. URL `https://doi.org/10.1142/9789812836281_0002`. [p91]

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`. [p105]

S. van Buuren and K. Groothuis-Oudshoorn. *Mice: Multivariate Imputation by Chained Equations*, 2017. URL `https://CRAN.R-project.org/package=mice`. R package version 2.46.0. [p105]

H. L. J. van der Maas and P. C. M. Molenaar. Stagewise cognitive development: An application of catastrophe theory. *Psychological Review*, 99(3):395–417, 1992. [p91]

H. L. J. van der Maas, R. Kolstein, and J. van der Pligt. Sudden transitions in attitudes. *Sociological Methods & Research*, 32(125–152), 2003. URL `https://doi.org/10.1177/0049124103253773`. [p101]

M. van Dijk and P. van Geert. Wobbles, humps and sudden jumps: A case study of continuity, discontinuity and variability in early language development. *Infant and Child Development*, 16(1): 7–33, 2007. URL `https://doi.org/10.1002/icd.506`. [p91]

I. Visser. Depmix: An R-package for fitting mixture models on mixed multivariate data with Markov dependencies. Technical report, University of Amsterdam, 2007. URL `http://cran.r-project.org`. [p91]

I. Visser and M. Speekenbrink. depmixS4: An **R** package for hidden Markov models. *Journal of Statistical Software*, 36(7):1–21, 2010. URL `https://doi.org/10.18637/jss.v036.i07`. [p92]

I. Visser and M. Speekenbrink. *depmixS4: Dependent Mixture Models - Hidden Markov Models of GLMs and Other Distributions in S4*, 2016. URL `https://CRAN.R-project.org/package=depmixS4`. R package version 1.3-3. [p92]

V. Volterra. Fluctuations in the abundance of a species considered mathematically. *Nature*, 118: 558–560, 1926. URL https://doi.org/10.1038/118558a0. [p96, 101]

M. Yang and S.-M. Chow. Using state-space model with regime switching to represent the dynamics of facial electromyography (EMG) data. *Psychometrika: Application and Case Studies*, 74(4): 744–771, 2010. URL https://doi.org/10.1007/s11336-010-9176-2. [p91, 94, 95, 96, 99, 100]

Funding for this study was provided by NSF grant SES-1357666, NIH grants R01MH61388, R01HD07699, R01GM105004, Pennsylvania State Quantitative Social Sciences Initiative, and UL TR000127 from the National Center for Advancing Translational Sciences.

Lu Ou
ACTNext
ACT, Inc.
500 Act Drive
Iowa City, IA 52244
E-mail: lu.ou@act.org

Michael D. Hunter
Georgia Institute of Technology
J.S. Coon Bldg, Room 225
648 Cherry St NW
Atlanta, GA 30313
E-mail: mhunter43@gatech.edu

Sy-Miin Chow
Department of Human Development and Family Studies
The Pennsylvania State University
420 Biobehavioral Health Building
University Park, PA 16802
E-mail: emailquc16@psu.edu

		Discrete-time	Continuous-time
Single-regime	Linear	Linear state-space model KF **dynr**, **OpenMx**, **pomp**, **KFAS**, **dlm**, **dse**, MKFM6, SsfPack, **MATLAB**	Linear SDE/ODE CDEKF **dynr**, **pomp**, **OpenMx**, **ctsem**, **MATLAB**
	Nonlinear	Nonlinear state-space model EKF **dynr**, **pomp**, SsfPack, **MATLAB**	Nonlinear SDE/ODE CDEKF **dynr**, **pomp**, **MATLAB**
Multiple-regime	Linear	RS state-space model Kim filter **dynr**, **GAUSS** code, **MATLAB**	RS SDE/ODE CD Kim filter **dynr** only
	Nonlinear	RS nonlinear state-space model Extended Kim filter **dynr** only	RS nonlinear SDE/ODE CD extended Kim filter **dynr** only

Table 1: Models, algorithms, and software for the framework of regime-switching (non)linear state space models in discrete- and continuous-time. SDE = Stochastic Differential Equation, ODE = Ordinary Differential Equation, CD = Continuous-Discrete, RS = Regime-Switching, KF = Kalman filter (Kalman, 1960), EKF = Extended Kalman filter (Anderson and Moore, 1979; Bar-Shalom et al., 2001), Kim filter = KF + Hamilton filter + Collapsing procedure (Kim and Nelson, 1999). Extended Kim filter was proposed by Chow and Zhang (2013); the CD extended Kim filter is proposed by Chow et al. (2018).

RobustGaSP: Robust Gaussian Stochastic Process Emulation in R

by Mengyang Gu, Jesus Palomo, and James O. Berger

Abstract Gaussian stochastic process (GaSP) emulation is a powerful tool for approximating computationally intensive computer models. However, estimation of parameters in the GaSP emulator is a challenging task. No closed-form estimator is available and many numerical problems arise with standard estimates, e.g., the maximum likelihood estimator. In this package, we implement a marginal posterior mode estimator, for special priors and parameterizations. This estimation method that meets the robust parameter estimation criteria was discussed in Gu et al. (2018); mathematical reasons are provided therein to explain why robust parameter estimation can greatly improve predictive performance of the emulator. In addition, inert inputs (inputs that almost have no effect on the variability of a function) can be identified from the marginal posterior mode estimation at no extra computational cost. The package also implements the parallel partial Gaussian stochastic process (PP GaSP) emulator (Gu and Berger (2016)) for the scenario where the computer model has multiple outputs on, for example, spatial-temporal coordinates. The package can be operated in a default mode, but also allows numerous user specifications, such as the capability of specifying trend functions and noise terms. Examples are studied herein to highlight the performance of the package in terms of out-of-sample prediction.

Introduction

A GaSP emulator is a fast surrogate model used to approximate the outcomes of a computer model (Sacks et al. (1989); Bayarri et al. (2007); Paulo et al. (2012); Palomo et al. (2015); Gu and Berger (2016)). The prediction accuracy of the emulator often depends strongly on the quality of the parameter estimates in the GaSP model. Although the mean and variance parameters in the GaSP model are relatively easy to deal with, estimation of parameters in the correlation functions is difficult (Kennedy and O'Hagan (2001)). Standard methods of estimating these parameters, such as maximum likelihood estimation (MLE), often produce unstable results leading to inferior prediction. As shown in (Gu et al. (2018)), the GaSP emulator is unstable when the correlation between any two different inputs are estimated to be close to one or to zero. The former case causes a near singularity when inverting the covariance matrix (this can partially be addressed by adding a small nugget (Andrianakis and Challenor (2012))), while the latter problem happens more often and has no easy fix.

There are several packages on the Comprehensive R Archive Network (CRAN, `https://CRAN.R-project.org/`) which implement the GaSP model based on the MLE, including **DiceKriging** (Roustant et al. (2012)), **GPfit** (MacDonald et al. (2015)), **mleGP** (Dancik (2013)), **spatial** (Venables and Ripley (2002)), and **fields** (Nychka et al. (2016)). In these packages, bounds on the parameters in the correlation function are typically implemented to overcome the numerical problems with the MLE estimates. Predictions are, however, often quite sensitive to the choice of bound, which is essentially arbitrary, so this is not an appealing fix to the numerical problems.

In Gu (2016), marginal posterior modes based on several objective priors are studied. It has been found that certain parameterizations result in more robust estimators than others, and, more importantly, that some parameterizations which are in common use should clearly be avoided. Marginal posterior modes with the robust parameterization are mathematically stable, as the posterior density is shown to be zero at the two problematic cases–when the correlation is nearly equal to one or to zero. This motivates the **RobustGaSP** package; examples also indicate that the package results in more accurate in out-of-sample predictions than previous packages based on the MLE. We use the **DiceKriging** package in these comparisons, because it is a state-of-the-art implementation of the MLE methodology

The **RobustGaSP** package (Gu et al. (2016)) for R builds a GaSP emulator with robust parameter estimation. It provides a default method with regard to a specific correlation function, a mean/trend function and an objective prior for the parameters. Users are allowed to specify them, for example, by using a different correlation and/or trend function, another prior distribution, or by adding a noise term with either a fixed or estimated variance. Although the main purpose of the **RobustGaSP** package is to do emulation/approximation of a complex function, this package can also be used in fitting the GaSP model for other purposes, such as nonparametric regression, modeling spatial data and so on. For computational purposes, most of the time consuming functions in the **RobustGaSP** package are implemented in C++.

We highlight several contributions of this work. First of all, to compute the derivative of the reference prior with a robust parametrization in (Gu et al. (2018)) is computationally expensive, however this information is needed to find the posterior mode by the low-storage quasi-Newton optimization method (Nocedal (1980)). We introduce a robust and computationally efficient prior, called the jointly robust prior (Gu (2018)), to approximate the reference prior in the tail rates of the posterior. This has been implemented as a default setting in the **RobustGaSP** package.

Furthermore, the use of the jointly robust prior provides a natural shrinkage for sparsity and thus can be used to identify inert/noisy inputs (if there are any), implemented in the `findInertInputs` function in the **RobustGaSP** package. A formal approach to Bayesian model selection requires a comparison of 2^p models for p variables, whereas in the **RobustGaSP** package, only the posterior mode of the full model has to be computed. Eliminating mostly inert inputs in a computer model is similar to not including regression coefficients that have a weak effect, since the noise introduced in their estimation degrades prediction. However, as the inputs have a nonlinear effect to the output, variable selection in GaSP is typically much harder than the one in the linear regression. The `findInertInputs` function in the **RobustGaSP** package can be used, as a fast pre-experimental check, to separate the influential inputs and inert inputs in highly nonlinear computer model outputs.

The **RobustGaSP** package also provides some regular model checks in fitting the emulator, while the robustness in the predictive performance is the focus in Gu et al. (2018). More specifically, the leave-one-out cross validation, standardized residuals and Normal QQ-plot of the standardized residuals are implemented and will be introduced in this work.

Lastly, some computer models have multiple outputs. For example, each run of the TITAN2D simulator produces up to 10^9 outputs of the pyroclastic flow heights over a spatial-temporal grid of coordinates (Patra et al. (2005); Bayarri et al. (2009)). The computational complexity of building a separate GaSP emulator for the output at each grid is $O(kn^3)$, where k is the number of grids and n is the number of computer model runs. The package also implements another computationally more efficient emulator, called the parallel partial Gaussian stochastic process emulator, which has the computational complexity being the maximum of $O(n^3)$ and $O(kn^2)$ (Gu and Berger (2016)). When the number of outputs in each simulation is large, the computational cost of PP GaSP is much smaller than the separate emulator of each output.

The rest of the paper is organized as follows. In the next section, we briefly review the statistical methodology of the GaSP emulator and the robust posterior mode estimation. In Section An overview of RobustGaSP , we describe the structure of the package and highlight the main functions implemented in this package. In Section Numerical examples , several numerical examples are provided to illustrate the behavior of the package under different scenarios. In Section Concluding remarks, we present conclusions and briefly discuss potential extensions. Examples will be provided throughout the paper for illustrative purposes.

The statistical framework

GaSP emulator

Prior to introducing specific functions and usage of the **RobustGaSP** package, we first review the statistical formulation of the GaSP emulator of the computer model with real-valued scalar outputs. Let $\mathbf{x} \in \mathcal{X}$ denote a p-dimensional vector of inputs for the computer model, and let $y(\mathbf{x})$ denote the resulting simulator output, which is assumed to be real-valued in this section. The simulator $y(\mathbf{x})$ is viewed as an unknown function modeled by the stationary GaSP model, meaning that for any inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ from $\mathcal{X}$, the likelihood is a multivariate normal distribution,

$$(y(\mathbf{x}_1), \ldots, y(\mathbf{x}_n))^\top \mid \boldsymbol{\mu}, \sigma^2, \mathbf{R} \sim \mathcal{MN}((\mu(\mathbf{x}_1), \ldots, \mu(\mathbf{x}_n))^\top, \sigma^2\mathbf{R})\,, \tag{1}$$

here $\mu(\cdot)$ is the mean function, σ^2 is the unknown variance parameter and $\mathbf{R}$ is the correlation matrix. The mean function is typically modeled via regression,

$$\mu(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\theta} = \sum_{t=1}^{q} h_t(\mathbf{x})\theta_t\,, \tag{2}$$

where $\mathbf{h}(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_q(\mathbf{x}))$ is a vector of specified mean basis functions and θ_t is the unknown regression parameter for basis function $h_t(\cdot)$. In the default setting of the **RobustGaSP** package, a constant basis function is used, i.e., $h(\mathbf{x}) = 1$; alternatively, a general mean structure can be specified by the user (see Section An overview of RobustGaSP for details).

The (i, j) element of $\mathbf{R}$ in (1) is modeled through a correlation function $c(\mathbf{x}_i, \mathbf{x}_j)$. The product

Matérn $\alpha = 5/2$	$\left(1 + \frac{\sqrt{5}d}{\gamma} + \frac{5d^2}{3\gamma^2}\right)\exp\left(-\frac{\sqrt{5}d}{\gamma}\right)$
Matérn $\alpha = 3/2$	$\left(1 + \frac{\sqrt{3}d}{\gamma}\right)\exp\left(-\frac{\sqrt{3}d}{\gamma}\right)$
Power exponential	$\exp\left\{-\left(\frac{d}{\gamma}\right)^{\alpha}\right\}$, $0 < \alpha \leqslant 2$

Table 1: Correlation functions currently implemented in **RobustGaSP**. Here γ is the range parameter and d is the distance between two points in each dimension. For simplicity, the subscript l in Equation (3) has been dropped.

correlation function is often assumed in the emulation of computer models (Santner et al. (2003)),

$$c(\mathbf{x}_i,\, \mathbf{x}_j) = \prod_{l=1}^{p} c_l(x_{il}, x_{jl}), \tag{3}$$

where $c_l(\cdot,\, \cdot)$ is an one-dimensional correlation function for the l^{th} coordinate of the input vector. Some frequently chosen correlation functions are implemented in the **RobustGaSP** package, listed in Table 1. In order to use the power exponential covariance function, one needs to specify the roughness parameter α_l, which is often set to be close to 2; e.g., $\alpha_l = 1.9$ is advocated in Bayarri et al. (2009), which maintains an adequate smoothness level yet avoids the numerical problems with $\alpha_l = 2$.

The Matérn correlation is commonly used in modeling spatial data (Stein (2012)) and has recently been advocated for computer model emulation (Gu et al. (2018)); one benefit is that the roughness parameter of the Matérn correlation directly controls the smoothness of the process. For example, the Matérn correlation with $\alpha_l = 5/2$ results in sample paths of the GaSP that are twice differentiable, a smoothness level that is usually desirable. Obtaining this smoothness with the more common squared exponential correlation comes at a price, however, as, for large distances, the correlation drops quickly to zero. For the Matérn correlation with $\alpha_l = 5/2$, the natural logarithm of the correlation only decreases linearly with distance, a feature which is much better for emulation of computer models. Based on these reasons, the Matérn correlation with $\alpha_l = 5/2$ is the default correlation function in **RobustGaSP**. It is also the default correlation function in some other packages, such as **DiceKriging** (Roustant et al. (2012)).

Since the simulator is expensive to run, we will at most be able to evaluate $y(\mathbf{x})$ at a set of design points. Denote the chosen design inputs as $\mathbf{x}^{\mathcal{D}} = \{\mathbf{x}_1^{\mathcal{D}}, \mathbf{x}_2^{\mathcal{D}}, ..., \mathbf{x}_n^{\mathcal{D}}\}$, where $\mathcal{D} \subset \mathcal{X}$. The resulting outcomes of the simulator are denoted as $\mathbf{y}^{\mathcal{D}} = (y_1^{\mathcal{D}}, y_2^{\mathcal{D}}, ..., y_n^{\mathcal{D}})^{\top}$. The design points are usually chosen to be “space-filling", including the uniform design and lattice designs. The Latin hypercube (LH) design is a “space-filling" design that is widely used. It is defined in a rectangle whereby each sample is the only one in each axis-aligned hyperplane containing it. LH sampling for a 1-dimensional input space is equivalent to stratified sampling, and the variance of an estimator based on stratified sampling has less variance than the random sampling scheme (Santner et al. (2003)); for a multi-dimensional input space, the projection of the LH samples on each dimension spreads out more evenly compared to simple stratified sampling. The LH design is also often used along with other constraints, e.g., the maximin Latin Hypercube maximizes the minimum Euclidean distance in the LH samples. It has been shown that the GaSP emulator based on maximin LH samples has a clear advantage compared to the uniform design in terms of prediction (see, e.g., Chen et al. (2016)). For these reasons, we recommend the use of the LH design, rather than the uniform design or lattice designs.

Robust parameter estimation

The parameters in a GaSP emulator include mean parameters, a variance parameter, and range parameters, denoted as $(\theta_1, .., \theta_q, \sigma^2, \gamma_1, ..., \gamma_p)$. The objective prior implemented in the **RobustGaSP** package has the form

$$\pi(\boldsymbol{\theta}, \sigma^2, \boldsymbol{\gamma}) \propto \frac{\pi(\boldsymbol{\gamma})}{\sigma^2}, \tag{4}$$

$\pi^R(\boldsymbol{\gamma})$	$\lvert\mathbf{I}^*(\boldsymbol{\gamma})\rvert^{1/2}$
$\pi^R(\boldsymbol{\xi})$	$\lvert\mathbf{I}^*(\boldsymbol{\xi})\rvert^{1/2}$ with $\xi_l = \log(1/\gamma_l)$, for $l = 1, ..., p$
$\pi^{JR}(\boldsymbol{\beta})$	$(\sum_{l=1}^{p} C_l\beta_l)^a exp(-b\sum_{l=1}^{p} C_l\beta_l)$, with $\beta_l = 1/\gamma_l$, for $l = 1, ..., p$

Table 2: Different priors for the parameters in the correlation function implemented in **RobustGaSP**. Here $\mathbf{I}^*(\cdot)$ is the expected Fisher information matrix, after integrating out $(\boldsymbol{\theta}, \sigma^2)$. The default choice of the prior parameters in $\pi^{JR}(\boldsymbol{\beta})$ is $a = 0.2$, $b = n^{-1/p}(a + p)$, and C_l equal to the mean of $|x^{\mathcal{D}}_{il} - x^{\mathcal{D}}_{jl}|$, for $1 \leqslant i, j \leqslant n$, $i \neq j$.

where $\pi(\boldsymbol{\gamma})$ is an objective prior for the range parameters. After integrating out $(\boldsymbol{\theta}, \sigma^2)$ by the prior in (4), the marginal likelihood is

$$\mathcal{L}(\mathbf{y}^{\mathcal{D}}|\boldsymbol{\gamma}) \propto |\mathbf{R}|^{-\frac{1}{2}} |\mathbf{h}^\top(\mathbf{x}^{\mathcal{D}})\mathbf{R}^{-1}\mathbf{h}(\mathbf{x}^{\mathcal{D}})|^{-\frac{1}{2}} \left(S^2\right)^{-(\frac{n-q}{2})}, \tag{5}$$

where $S^2 = (\mathbf{y}^{\mathcal{D}})^\top \mathbf{Q}\mathbf{y}^{\mathcal{D}}$ with $\mathbf{Q} = \mathbf{R}^{-1}\mathbf{P}$ and $\mathbf{P} = \mathbf{I}_n - \mathbf{h}(\mathbf{x}^{\mathcal{D}})\{\mathbf{h}^\top(\mathbf{x}^{\mathcal{D}})\mathbf{R}^{-1}\mathbf{h}(\mathbf{x}^{\mathcal{D}})\}^{-1}\mathbf{h}^\top(\mathbf{x}^{\mathcal{D}})\mathbf{R}^{-1}$, with $\mathbf{I}_n$ being the identity matrix of size n.

The reference prior $\pi^R(\cdot)$ and the jointly robust prior $\pi^{JR}(\cdot)$ for the range parameters with robust parameterizations implemented in the **RobustGaSP** package are listed in Table 2. Although the computational complexity of the value of the reference prior is the same as the marginal likelihood, the derivatives of the reference prior are computationally hard. The numerical derivative is thus computed in the package in finding the marginal posterior mode using the reference prior. Furthermore, the package incorporates, by default, the jointly robust prior with the prior parameters $(C_1, \ldots, C_p, a, b)$ (whose values are given in Table 2). The properties of the jointly robust prior are studied extensively in Gu (2018). The jointly robust prior approximates the reference prior reasonably well with the default prior parameters, and has a close form derivatives. The jointly robust prior is a proper prior with a closed form of the normalizing constant and the first two moments. In addition, the posterior modes of the jointly robust prior can identify the inert inputs, as discussed in Section 14.2.4.

The range parameters $(\gamma_1, ..., \gamma_p)$ are estimated by the modes of the marginal posterior distribution

$$(\hat{\gamma}_1, \ldots \hat{\gamma}_p) = \underset{\gamma_1, \ldots, \gamma_p}{argmax}(\mathcal{L}(\mathbf{y}^{\mathcal{D}}|\gamma_1, \ldots, \gamma_p)\pi(\gamma_1, \ldots, \gamma_p)). \tag{6}$$

When another parameterization is used, parameters are first estimated by the posterior mode and then transformed back to obtain $(\hat{\gamma}_1, \ldots \hat{\gamma}_p)$.

Various functions implemented in the **RobustGaSP** package can be reused in other studies. `log_marginal_lik` and `log_marginal_lik_deriv` give the natural logarithm of the marginal likelihood in (5) and its directional derivatives with regard to $\boldsymbol{\gamma}$, respectively. The reference priors $\pi^R(\boldsymbol{\gamma})$ and $\pi^R(\boldsymbol{\xi})$ are not coded separately, but `neg_log_marginal_post_ref` gives the negative values of the log marginal posterior distribution and thus one can use -`neg_log_marginal_post_ref` minus `log_marginal_lik` to get the log reference prior. The jointly robust prior $\pi^{JR}(\boldsymbol{\beta})$ and its directional derivatives with regard to $\boldsymbol{\beta}$ are coded in `log_approx_ref_prior` and `log_approx_ref_prior_deriv`, respectively. All these functions are not implemented in other packages and can be reused in other theoretical studies and applications.

Prediction

After obtaining $\hat{\boldsymbol{\gamma}}$, the predictive distribution of the GaSP emulator (after marginalizing $(\boldsymbol{\theta}, \sigma^2)$ out) at a new input point $\mathbf{x}^*$ follows a student t distribution

$$y(\mathbf{x}^*) \mid \mathbf{y}^{\mathcal{D}}, \hat{\boldsymbol{\gamma}} \sim \mathcal{T}(\hat{y}(\mathbf{x}^*), \hat{\sigma}^2 c^{**}, n-q), \tag{7}$$

with $n - q$ degrees of freedom, where

$$\begin{aligned}
\hat{y}(\mathbf{x}^*) &= \mathbf{h}(\mathbf{x}^*)\hat{\boldsymbol{\theta}} + \mathbf{r}^\top(\mathbf{x}^*)\mathbf{R}^{-1}\left(\mathbf{y}^{\mathcal{D}} - \mathbf{h}(\mathbf{x}^{\mathcal{D}})\hat{\boldsymbol{\theta}}\right), \\
\hat{\sigma}^2 &= (n-q)^{-1}\left(\mathbf{y}^{\mathcal{D}} - \mathbf{h}(\mathbf{x}^{\mathcal{D}})\hat{\boldsymbol{\theta}}\right)^T \mathbf{R}^{-1}\left(\mathbf{y}^{\mathcal{D}} - \mathbf{h}(\mathbf{x}^{\mathcal{D}})\hat{\boldsymbol{\theta}}\right), \\
c^{**} &= c(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{r}^\top(\mathbf{x}^*)\mathbf{R}^{-1}\mathbf{r}(\mathbf{x}^*) + \left(\mathbf{h}(\mathbf{x}^*) - \mathbf{h}^\top(\mathbf{x}^{\mathcal{D}})\mathbf{R}^{-1}\mathbf{r}(\mathbf{x}^*)\right)^\top \\
&\quad \times \left(\mathbf{h}^\top(\mathbf{x}^{\mathcal{D}})\mathbf{R}^{-1}\mathbf{h}(\mathbf{x}^{\mathcal{D}})\right)^{-1}\left(\mathbf{h}(\mathbf{x}^*) - \mathbf{h}^\top(\mathbf{x}^{\mathcal{D}})\mathbf{R}^{-1}\mathbf{r}(\mathbf{x}^*)\right), \qquad (8)
\end{aligned}$$

with $\hat{\boldsymbol{\theta}} = \left(\mathbf{h}^T(\mathbf{x}^{\mathcal{D}})\mathbf{R}^{-1}\,\mathbf{h}(\mathbf{x}^{\mathcal{D}})\right)^{-1}\mathbf{h}^\top(\mathbf{x}^{\mathcal{D}})\mathbf{R}^{-1}\mathbf{y}^{\mathcal{D}}$ being the generalized least squares estimator for $\boldsymbol{\theta}$ and $\mathbf{r}(\mathbf{x}^*) = (c(\mathbf{x}^*, \mathbf{x}_1^{\mathcal{D}}), \ldots, c(\mathbf{x}^*, \mathbf{x}_n^{\mathcal{D}}))^\top$.

The emulator interpolates the simulator at the design points $\mathbf{x}_i^{\mathcal{D}}$, $1 \leqslant i \leqslant n$, because when $\mathbf{x}^* = \mathbf{x}_i^{\mathcal{D}}$, one has $\mathbf{r}^\top(\mathbf{x}^*)\mathbf{R}^{-1} = \mathbf{e}_i^\top$, where $\mathbf{e}_i$ is the n dimensional vector with the i^{th} entry being 1 and the others being 0. At other inputs, the emulator not only provides a prediction of the simulator (i.e., $\hat{y}(\mathbf{x}^*)$) but also an assessment of prediction accuracy. It also incorporates the uncertainty arising from estimating $\boldsymbol{\theta}$ and σ^2 since this was developed from a Bayesian perspective.

We now provide an example in which the input has one dimension, ranging from $[0, 10]$ (Higdon and others (2002)). Estimation of the range parameters using the **RobustGaSP** package can be done through the following code:

```
R> library(RobustGaSP)
R> library(lhs)
R> set.seed(1)
R> input <- 10 * maximinLHS(n=15, k=1)
R> output <- higdon.1.data(input)
R> model <- rgasp(design = input, response = output)
R> model

Call:
rgasp(design = input, response = output)
Mean parameters:  0.03014553
Variance parameter:  0.5696874
Range parameters:  1.752277
Noise parameter:  0
```

The fourth line of the code generates 15 LH samples at $[0, 10]$ through the `maximinLHS` function of the **lhs** package (Carnell (2016)). The function `higdon.1.data` is provided within the **RobustGaSP** package which has the form $y(x) = \sin(2\pi x/10) + 0.2\sin(2\pi x/2.5)$. The third line fits a GaSP model with the robust parameter estimation by marginal posterior modes.

The `plot` function in **RobustGaSP** package implements the leave-one-out cross validation for a `"rgasp"` class after the GaSP model is built (see Figure 1 for its output):

```
R> plot(model)
```

The prediction at a set of input points can be done by the following code:

```
R> testing_input <- as.matrix(seq(0, 10, 1/50))
R> model.predict<-predict(model, testing_input)
R> names(model.predict)

[1] "mean"    "lower95" "upper95" "sd"
```

The `predict` function generates a list containing the predictive mean, lower and upper 95% quantiles and the predictive standard deviation, at each test point $\mathbf{x}^*$. The prediction and the real outputs are plotted in Figure 2; produced by the following code:

```
R> testing_output <- higdon.1.data(testing_input)
R> plot(testing_input, model.predict$mean,type='l', col='blue',
+      xlab='input', ylab='output')
R> polygon( c(testing_input,rev(testing_input)), c(model.predict$lower95,
+      rev(model.predict$upper95)), col = "grey80", border = F)
R> lines(testing_input, testing_output)
R> lines(testing_input,model.predict$mean, type='l', col='blue')
R> lines(input, output, type='p')
```

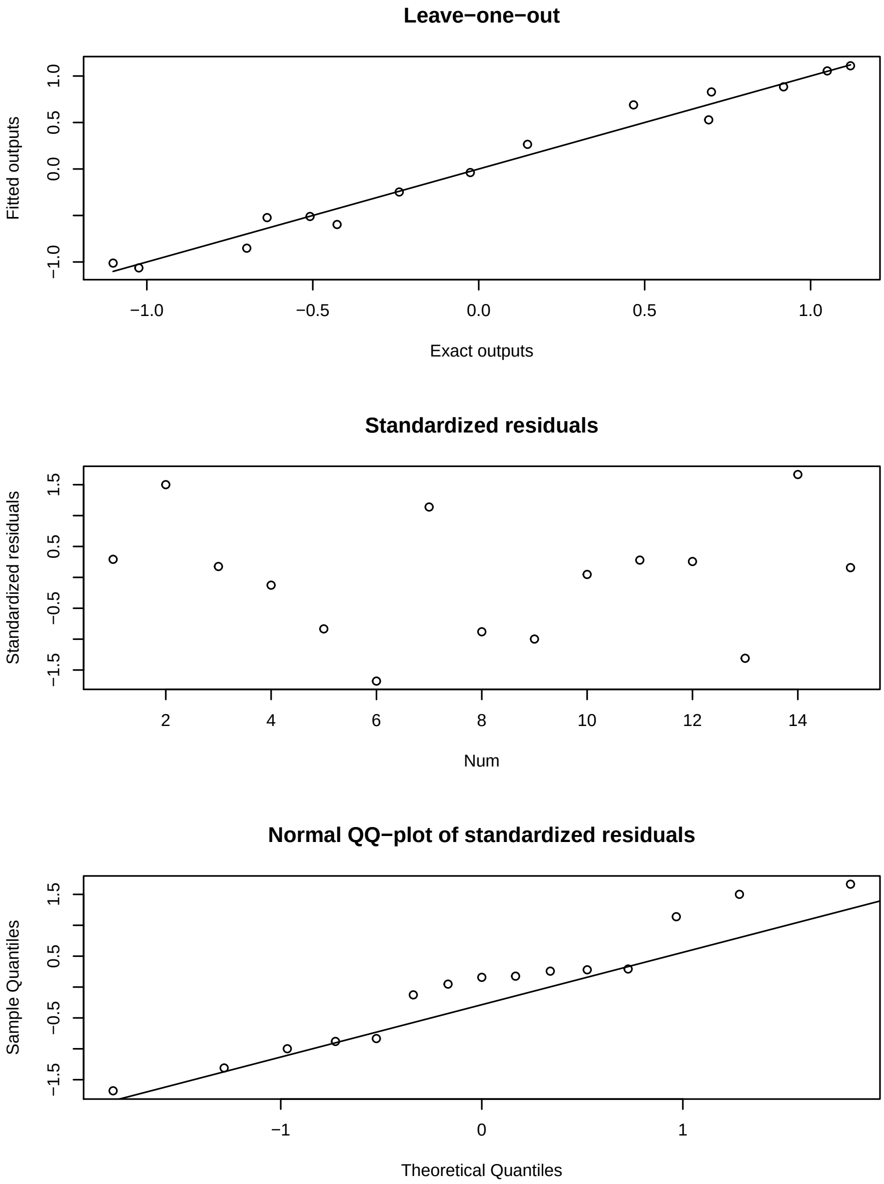


Figure 1: Leave-one-out fitted values for the GaSP model of the `higdon.1.data` function in the **RobustGaSP** package.

It is also possible to sample from the predictive distribution (which is a multivariate t distribution) using the following code:

```
R> model.sample <- simulate(model, testing_input, num_sample=10)
R> matplot(testing_input, model.sample, type='l', xlab='input', ylab='output')
R> lines(input, output,type='p')
```

The plots of 10 posterior predictive samples are shown in Figure 3.

Identification of inert inputs

Some inputs have little effect on the output of a computer model. Such inputs are called inert inputs (Linkletter et al. (2006)). To quantify the influence of a set of inputs on the variability of the outputs, functional analysis of the variance (functional ANOVA) can be used, often implemented through Sobol's Indices (Sobol' (1990); Sobol (2001)). Methods for numerical calculation of Sobol's Indices have been implemented in the **sensitivity** package (Pujol et al. (2016)) for R.

The identification of inert inputs through the posterior modes with the jointly robust prior ($\pi^{JR}(\cdot)$) for the range parameters is discussed in Gu (2018). The package discussed here implements

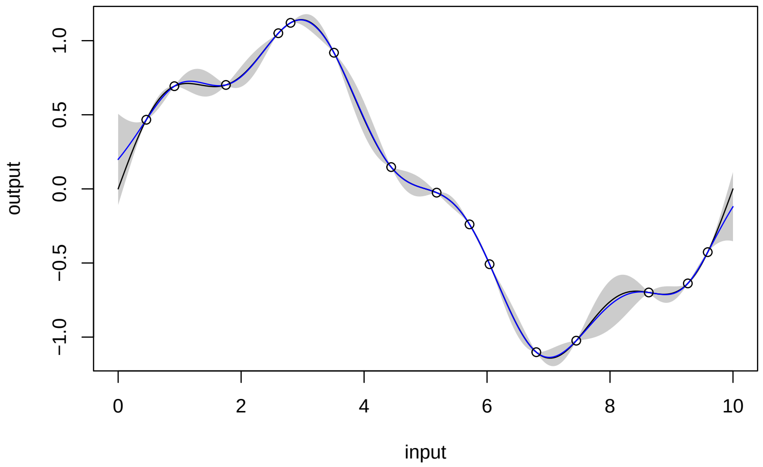


Figure 2: The predictive mean (blue curve), the 95% predictive credible interval (grey region) and the real function (black curve). The outputs at the design points are the black circles.

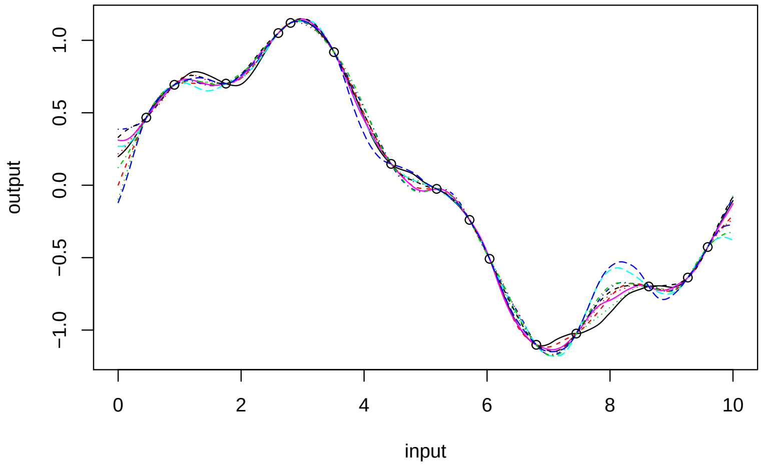


Figure 3: 10 posterior predictive samples from the **RobustGaSP**. The outputs at the design points are the black circles.

this idea, using the *estimated normalized inverse range parameters*,

$$\hat{P}_l = \frac{pC_l\hat{\beta}_l}{\sum_{i=1}^{p} C_i\hat{\beta}_i}, \tag{9}$$

for $l = 1, ..., p$. The involvement of C_l (defined in Table 2) is to account for the different scales of different inputs. The denominator $(\sum_{i=1}^{p} C_i\hat{\beta}_i)$ reflects the overall size of the estimator and $C_l\hat{\beta}_l$ gives the contribution of the l^{th} input. The average $\hat{P}_l$ is 1 and the sum of $\hat{P}_l$ is p. When $\hat{P}_l$ is very close to 0, it means the l^{th} input might be an inert input. In the **RobustGaSP** package, the default threshold is 0.1; i.e., when $\hat{P}_l < 0.1$, it is suggested to be an inert input. The threshold can also be specified by users through the argument `threshold` in the function `findInertInputs`.

For demonstration purpose, we build a GaSP emulator for the borehole experiment (Worley (1987); Morris et al. (1993); An and Owen (2001)), a well-studied computer experiment benchmark which models water flow through a borehole. The output y is the flow rate through the borehole in

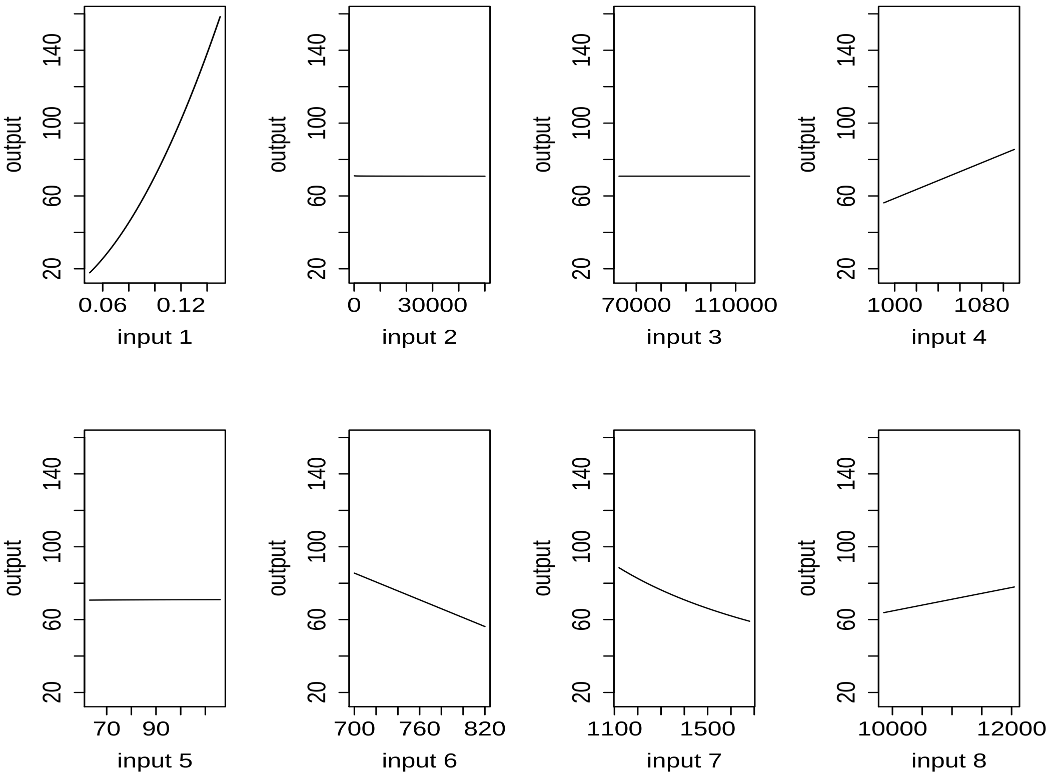


Figure 4: Values of the borehole function by varying one input at a time.

$m^3/year$ and it is determined by the equation:

$$y = \frac{2\pi T_u (H_u - H_l)}{ln(r/r_\omega)\big[1 + \frac{2LT_u}{ln(r/r_\omega)r_\omega^2 K_\omega} + \frac{T_u}{T_l}\big]},$$

where $r_\omega, r, T_u, H_u, T_l, H_l, L$ and K_ω are the 8 inputs constrained in a rectangular domain with the following ranges

$$r_\omega \in [0.05, 0.15], \quad r \in [100, 50000], \quad T_u \in [63070, 115600], \quad H_u \in [990, 1110],$$
$$T_l \in [63.1, 116], \quad H_l \in [700, 820], \quad L \in [1120, 1680], \quad K_\omega \in [9855, 12045].$$

We use 40 maximin LH samples to find inert inputs at the borehole function through the following code.

```
R> set.seed(1)
R> input <- maximinLHS(n=40, k=8)  # maximin lhd sample
R> # rescale the design to the domain of the borehole function
R> LB <- c(0.05, 100, 63070, 990, 63.1, 700, 1120, 9855)
R> UB <- c(0.15, 50000, 115600, 1110, 116, 820, 1680, 12045)
R> range <- UB - LB
R> for(i in 1:8) {
R>   input[,i] = LB[i] + range[i] * input[,i]
R> }
R> num_obs <- dim(input)[1]
R> output <- matrix(0,num_obs,1)
R> for(i in 1:num_obs) {
+   output[i] <- borehole(input[i,])
+ }
R> m <- rgasp(design = input, response = output, lower_bound=FALSE)
R> P <- findInertInputs(m)

The estimated normalized inverse range parameters are : 3.440765 8.13156e-09
```

```
4.983695e-09 0.844324 4.666519e-09 1.31081 1.903236 0.5008652
The inputs  2 3 5 are suspected to be inert inputs
```

Similar to the automatic relevance determination model in neural networks, e.g. MacKay (1996); Neal (1996), and in machine learning, e.g. Tipping (2001); Li et al. (2002), the function `findInertInputs` of the **RobustGaSP** package indicates that the $2^{nd}, 3^{rd}$, and 5^{th} inputs are suspected to be inert inputs. Figure 4 presents the plots of the borehole function when varying one input at a time. This analyzes the *local* sensitivity of an input when having the others fixed. Indeed, the output of the borehole function changes very little when the $2^{nd}, 3^{rd}$, and 5^{th} inputs vary.

Noisy outputs

The ideal situation for a computer model is that it produces noise-free data, meaning that the output will not change at the same input. However, there are several cases in which the outputs are noisy. First of all, the numerical solution of the partial differential equations of a computer model could introduce small errors. Secondly, when only a subset of inputs are analyzed, the computer model is no longer deterministic given only the subset of inputs. For example, if we only use the 5 influential inputs of the borehole function, the outcomes of this function are no longer deterministic, since the variation of the inert inputs still affects the outputs a little. Moreover, some computer models might be stochastic or have random terms in the models.

For these situations, the common adjustment is to add a noise term to account for the error, such as $\tilde{y}(\cdot) = y(\cdot) + \epsilon$, where $y(\cdot)$ is the noise-free GaSP and ϵ is an i.i.d. mean-zero Gaussian white noise (Ren et al. (2012); Gu and Berger (2016)). To allow for marginalizing out the variance parameter, the covariance function for the new process $\tilde{y}(\cdot)$ can be parameterized as follows:

$$\sigma^2 \tilde{c}(\mathbf{x}_l, \mathbf{x}_m) = \sigma^2 \{ c(\mathbf{x}_l, \mathbf{x}_m) + \eta \delta_{lm} \}, \tag{10}$$

where η is defined to be the nugget-variance ratio and δ_{lm} is a Dirac delta function when $l = m$, $\delta_{lm} = 1$. After adding the nugget, the covariance matrix becomes:

$$\sigma^2 \tilde{\mathbf{R}} = \sigma^2 (\mathbf{R} + \eta \mathbf{I}_n). \tag{11}$$

Although we call η the nugget-variance ratio parameter, the analysis is different than when a nugget is directly added to stabilize the computation in the GaSP model. As pointed out in Roustant et al. (2012), when a nugget is added to stabilize the computation, it is also added to the covariance function in prediction, and, hence, the resulting emulator is still an interpolator, meaning that the prediction will be exact at the design points. However, when a noise term is added, it does not go into the covariance function and the prediction at a design point will not be exact (because of the effect of the noise).

Objective Bayesian analysis for the proposed GaSP model with the noise term can be done by defining the prior

$$\tilde{\pi}(\boldsymbol{\theta}, \sigma^2, \boldsymbol{\gamma}, \eta) \propto \frac{\tilde{\pi}(\boldsymbol{\gamma}, \eta)}{\sigma^2}, \tag{12}$$

where $\tilde{\pi}(\boldsymbol{\gamma}, \eta)$ is now the prior for the range and nugget-variance ratio parameters $(\boldsymbol{\gamma}, \eta)$. The reference prior and the jointly robust prior can also be extended to be $\tilde{\pi}^R(\cdot)$ and $\tilde{\pi}^{JR}(\cdot)$ with robust parameterizations listed in Table 2. Based on the computational feasibility of the derivatives and the capacity to identify noisy inputs, the proposed default setting is to use the jointly robust prior with specified prior parameters in Table 2.

As in the previous noise-free GaSP model, one can estimate the range and nugget-variance ratio parameters by their marginal maximum posterior modes

$$(\hat{\gamma}_1, \ldots \hat{\gamma}_p, \hat{\eta}) = \underset{\gamma_1, \ldots, \gamma_p, \eta}{argmax} \; \mathcal{L}(\mathbf{y}^{\mathcal{D}} | \gamma_1, \ldots, \gamma_p, \eta) \tilde{\pi}(\gamma_1, \ldots, \gamma_p, \eta). \tag{13}$$

After obtaining $\hat{\boldsymbol{\gamma}}$ and $\hat{\eta}$, the predictive distribution of the GaSP emulator is almost the same as in Equation (7); simply replace $c(\cdot, \cdot)$ by $\tilde{c}(\cdot, \cdot)$ and $\mathbf{R}$ by $\tilde{\mathbf{R}}$.

Using only the influential inputs of the borehole function, we construct the GaSP emulator with a nugget based on 30 maximin LH samples through the following code:

```
R> m.subset <- rgasp(design = input[ ,c(1,4,6,7,8)], response = output,
+    nugget.est=TRUE)
R> m.subset


Call:
```

$\tilde{\pi}^{R}(\boldsymbol{\gamma},\eta)$	$\lvert\mathbf{I}^{*}(\boldsymbol{\gamma},\eta)\rvert^{1/2}$
$\tilde{\pi}^{R}(\boldsymbol{\xi},\eta)$	$\lvert\mathbf{I}^{*}(\boldsymbol{\xi},\eta)\rvert^{1/2}$ with $\xi_l = \log(1/\gamma_l)$, for $l = 1,...,p$
$\tilde{\pi}^{JR}(\boldsymbol{\beta},\eta)$	$(\sum_{l=1}^{p} C_l\beta_l)^a exp(-b(\sum_{l=1}^{p} C_l\beta_l+\eta))$, with $\beta_l = 1/\gamma_l$, for $l = 1,...,p$

Table 3: Different priors for the parameters in the correlation function implemented in **RobustGaSP**, when a noise term is present. Here $\mathbf{I}^{*}(\cdot)$ is the expected fisher information matrix after integrating out $(\boldsymbol{\theta},\sigma^2)$. The default choices of the prior parameters in $\tilde{\pi}^{JR}(\boldsymbol{\beta},\eta)$ are: $a = 0.2$, $b = n^{-1/p}(a+p)$, and C_l equal to the mean of $|x^{\mathcal{D}}_{il} - x^{\mathcal{D}}_{jl}|$, for $1 \leqslant i,j \leqslant n$, $i \neq j$.

```
rgasp(design = input[, c(1, 4, 6, 7, 8)], response = output,
    nugget.est = TRUE)
Mean parameters:  170.9782
Variance parameter:  229820.7
Range parameters:  0.2489396 1438.028 1185.202 5880.335 44434.42
Noise parameter:  0.2265875
```

To compare the performance of the emulator with and without a noise term, we perform some out-of-sample testing. We build the GaSP emulator by the **RobustGaSP** package and the **DiceKriging** package using the same mean and covariance. In **RobustGaSP**, the parameters in the correlation functions are estimated by marginal posterior modes with the robust parameterization, while in **DiceKriging**, parameters are estimated by MLE with upper and lower bounds. We first construct these four emulators with the following code.

```
R> m.full <- rgasp(design = input, response = output)
R> m.subset <- rgasp(design = input[ ,c(1,4,6,7,8)], response = output,
+    nugget.est=TRUE)
R> dk.full <- km(design = input, response = output)
R> dk.subset <- km(design = input[ ,c(1,4,6,7,8)], response = output,
+    nugget.estim=TRUE)
```

We then compare the performance of the four different emulators at 100 random inputs for the borehole function.

```
R> set.seed(1)
R> dim_inputs <- dim(input)[2]
R> num_testing_input <- 100
R> testing_input <- matrix(runif(num_testing_input*dim_inputs),
+                          num_testing_input,dim_inputs)
R> for(i in 1:8) {
R>  testing_input[,i] <- LB[i] + range[i] * testing_input[,i]
R> }
R> m.full.predict <- predict(m.full, testing_input)
R> m.subset.predict <- predict(m.subset, testing_input[ ,c(1,4,6,7,8)])
R> dk.full.predict <- predict(dk.full, newdata = testing_input,type = 'UK')
R> dk.subset.predict <- predict(dk.subset,
+                     newdata = testing_input[ ,c(1,4,6,7,8)],type = 'UK')
R> testing_output <- matrix(0, num_testing_input, 1)
R> for(i in 1:num_testing_input) {
+   testing_output[i] <- borehole(testing_input[i, ])
+ }
R> m.full.error <- abs(m.full.predict$mean - testing_output)
R> m.subset.error <- abs(m.subset.predict$mean - testing_output)
R> dk.full.error <- abs(dk.full.predict$mean - testing_output)
R> dk.subset.error <- abs(dk.subset.predict$mean - testing_output)
```

Since the **DiceKriging** package seems not to have implemented a method to estimate the noise parameter, we only compare it with the nugget case. The box plot of the absolute errors of these

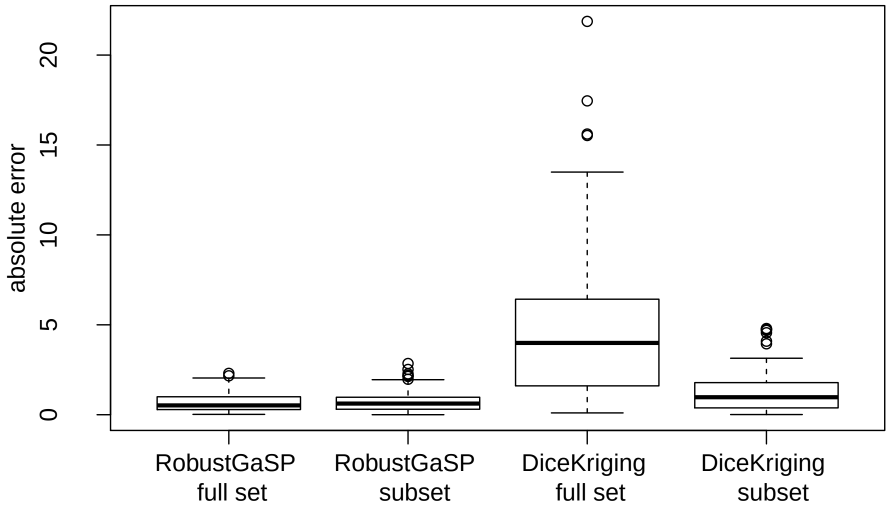


Figure 5: Absolute out-of-sample prediction errors at 100 random samples by different emulators of the borehole function based on $n = 30$ maximin LH samples. The first two boxes are the absolute predictive errors from **RobustGaSP**, with the full set of inputs and with only influential inputs (and a nugget), respectively, whereas the last two boxes are from **DiceKriging** with the full set of inputs and with only influential inputs (and a nugget), respectively.

4 emulators (all with the same correlation and mean function) at 100 held-out points are shown in Figure 5. The performance of the **RobustGaSP** package based on the full set of inputs or only influential inputs with a noise is similar, and they are both better than the predictions from the **DiceKriging** package.

An overview of RobustGaSP

Main functions

The main purpose of the **RobustGaSP** package is to predict a function at unobserved points based on only a limited number of evaluations of the function. The uncertainty associated with the predictions is obtained from the predictive distribution in Equation (7), which is implemented in two steps. The first step is to build a GaSP model through the `rgasp` function. This function allows users to specify the mean function, correlation function, prior distribution for the parameters, and to include a noise term or not. In the default setting, these are all specified. The mean and variance parameters are handled in a fully Bayesian way, and the range parameters in the correlation function are estimated by their marginal posterior modes. Moreover, users can also fix the range parameters, instead of estimating them, change/replace the mean function, add a noise term, etc. The `rgasp` function returns an object of the `"rgasp"` S4 class with all needed estimated parameters, including the mean, variance, noise and range parameters to perform predictions.

The second step is to compute the predictive distribution of the previously created GaSP model through the `predict` function, which produces the predictive means, the 95% predictive credible intervals, and the predictive standard deviations at each test point. As the predictive distribution follows a student t distribution in (7) for any test points, any quantile/percentile of the predictive distribution can be computed analytically. The joint distribution at a set of test points is a multivariate t distribution whose dimension is equal to the number of test points. Users can also sample from the posterior predictive distribution by using the `simulate` function.

The identification of inert inputs can be performed using the `findInertInput` function. As it only depends on the inverse range parameters through Equation (9), there is no extra computational cost in their identification (once the robust GaSP model has been built through the `rgasp` function). We suggest using the jointly robust prior by setting the argument `prior_choice="ref_approx"` in the `rgasp` function before calling the `findInertInput` function, because the penalty given by this prior is close to an L_1 penalty for the logarithm of the marginal likelihood (with the choice of default prior parameters) and, hence, it can shrink the parameters for those inputs with small effect.

Besides, the **RobustGaSP** package also implements the PP GaSP emulator introduced in Gu and Berger (2016) for the computer model with a vector of outputs. In the PP GaSP emulator, the variances and the mean values of the computer model outputs at different grids are allowed to be different, whereas the covariance matrix of physical inputs are assumed to be the same across grids. In estimation, the variance and the mean parameters are first marginalized out with the reference priors. Then the posterior mode is used for estimating the parameters in the kernel. The `ppgasp` function builds a PP GaSP model, which returns an object of the `"ppgasp"` S4 class with all needed estimated parameters. Then the predictive distribution of PP GaSP model is computed through the `predict.ppgasp` function. Similar to the emulator of the output with the scalar output, the `predict.ppgasp` function returns the predictive means, the 95% predictive credible intervals, and the predictive standard deviations at each test point.

The `rgasp` function

The `rgasp` function is one of the most important functions, as it performs the parameter estimation for the GaSP model of the computer model with a scalar output. In this section, we briefly review the implementation of the `rgasp` function and its optimization algorithm.

The $n \times p$ design matrix $\mathbf{x}^{\mathcal{D}}$ and the $n \times 1$ output vector $\mathbf{y}^{\mathcal{D}}$ are the only two required arguments (without default values) in the `rgasp` function. The default setting in the argument `trend` is a constant function, i.e., $\mathbf{h}(\mathbf{x}^{\mathcal{D}}) = \mathbf{1}_n$. One can also set `zero.mean=TRUE` in the `rgasp` function to assume the mean function in GaSP model is zero. By default, the GaSP model is defined to be noise-free, i.e., the noise parameter is 0. However, a noise term can be added with estimated or fixed variance. As the noise is parameterized following the form (10), the variance is marginalized out explicitly and the nugget-variance parameter η is left to be estimated. This can be done by specifying the argument `nugget.est = T` in the `rgasp` function; when the nugget-variance parameter η is known, it can be specified; e.g., $\eta = 0.01$ indicates the nugget-variance ratio is equal to 0.01 in `rgasp` and η will be not be estimated with such a specification.

Two classes of priors of the form (4), with several different robust parameterizations, have been implemented in the **RobustGaSP** package (see Table 3 for details). The prior that will be used is controlled by the argument `prior_choice` in the `rgasp` function. The reference prior $\pi^R(\cdot)$ with $\boldsymbol{\gamma}$ (the conventional parameterization of the range parameters for the correlation functions in Table 1) and $\boldsymbol{\xi} = \log(1/\boldsymbol{\gamma})$ parameterization can be specified through the arguments `prior_choice="ref_gamma"` and `prior_choice="ref_xi"`, respectively. The jointly robust prior $\pi^{JR}(\cdot)$ with the $\boldsymbol{\beta} = 1/\boldsymbol{\gamma}$ parameterization can be specified through the argument `prior_choice="ref_approx"`; this is the default choice used in `rgasp`, for the reasons discussed in Section Statistical framework.

The correlation functions implemented in the **RobustGaSP** package are shown in Table 1, with the default setting being `kernel_type = "matern_5_2"` in the `rgasp` function. The power exponential correlation function requires the specification of a vector of roughness parameters $\boldsymbol{\alpha}$ through the argument `alpha` in the `rgasp` function; the default value is $\alpha_l = 1.9$ for $l = 1, ..., p$, as suggested in Bayarri et al. (2009).

The `ppgasp` function

The `ppgasp` function performs the parameter estimation of the PP GaSP emulator for the computer model with a vector of outputs. In the `ppgasp` function, the output $\mathbf{y}^{\mathcal{D}}$ is a $n \times k$ matrix, where each row is the k-dimensional computer model outputs. The rest of the input quantities of the `ppgasp` function and `rgasp` function are the same.

Thus the `ppgasp` function return the estimated parameters, including k estimated variance parameters, and $q \times k$ mean parameters when the mean basis has q dimensions.

The optimization algorithm

Estimation of the range parameters $\boldsymbol{\gamma}$ is implemented through numerical search for the marginal posterior modes in Equation (6). The low-storage quasi-Newton optimization method (Nocedal (1980); Liu and Nocedal (1989)) has been used in the `lbfgs` function in the **nloptr** package (Ypma (2014)) for optimization. The closed-form marginal likelihood, prior and their derivatives are all coded in C++. The maximum number of iterations and tolerance bounds are allowed to be chosen by users with the default setting as `max_eval=30` and `xtol_rel=1e-5`, respectively.

Although maximum marginal posterior mode estimation with the robust parameterization eliminates the problems of the correlation matrix being estimated to be either $\mathbf{I}_n$ or $\mathbf{1}_n\mathbf{1}_n^\top$, the correlation matrix could still be close to these singularities in some scenarios, particularly when the sample size is very large. In such cases, we also utilize an upper bound for the range parameters $\boldsymbol{\gamma}$ (equivalent to a lower bound for $\boldsymbol{\beta} = 1/\boldsymbol{\gamma}$). The derivation of this bound is discussed in the Appendix. This bound is implemented in the `rgasp` function through the argument `lower_bound=TRUE`, and this is the default setting in **RobustGaSP**. As use of the bound is a somewhat ad hoc fix for numerical problems, we encourage users to also try the analysis without the bound; this can be done by specifying `lower_bound=FALSE`. If the answers are essentially unchanged, one has more confidence that the parameter estimates are satisfactory. Furthermore, if the purpose of the analysis is to detect inert inputs, users are also suggested to use the argument `lower_bound=FALSE` in the `rgasp` function.

Since the marginal posterior distribution could be multi-modal, the package allows for different initial values in the optimization by setting the argument `multiple_starts=TRUE` in the `rgasp` function. The first default initial value for each inverse range parameter is set to be 50 times their default lower bounds, so the starting value will not be too close to the boundary. The second initial value for each of the inverse range parameter is set to be half of the mean of the jointly robust prior. Two initial values of the nugget-variance parameter are set to be $\eta = 0.0001$ and $\eta = 0.0002$ respectively.

Examples

In this section, we present further examples of the performance of the **RobustGaSP** package, and include comparison with the **DiceKriging** package in R. We will use the same data, trend function, and correlation function for the comparisons. The default correlation function in both packages is the Matérn correlation with $\alpha = 5/2$ and the default trend function is a constant function. The only difference is the method of parameter estimation, as discussed in Section Statistical framework, where the **DiceKriging** package implements the MLE (by default) and the penalized MLE (PMLE) methods, Roustant et al. (2018).

The modified sine wave function

It is expected that, for a one-dimensional function, both packages will perform well with an adequate number of design points, so we start with the function called the modified sine wave discussed in Gu (2016). It has the form

$$y = 3\sin(5\pi x) + \cos(7\pi x),$$

where $x = [0, 1]$. We first perform emulation based on 12 equally spaced design points on $[0, 1]$.

```
R> sinewave <- function(x) {
+   3*sin(5*pi*x)*x + cos(7*pi*x)
+ }
R> input <- as.matrix(seq(0, 1, 1/11))
R> output <- sinewave(input)
```

The GaSP model is fitted by both the **RobustGaSP** and **DiceKriging** packages, with the constant mean function.

```
R> m <- rgasp(design=input, response=output)
R> m

Call:
rgasp(design = input, response = output)
Mean parameters:  0.1402334
Variance parameter:  2.603344
Range parameters:  0.04072543
Noise parameter:  0

R> dk <- km(design = input, response = output)
R> dk

Call:
km(design = input, response = output)
```

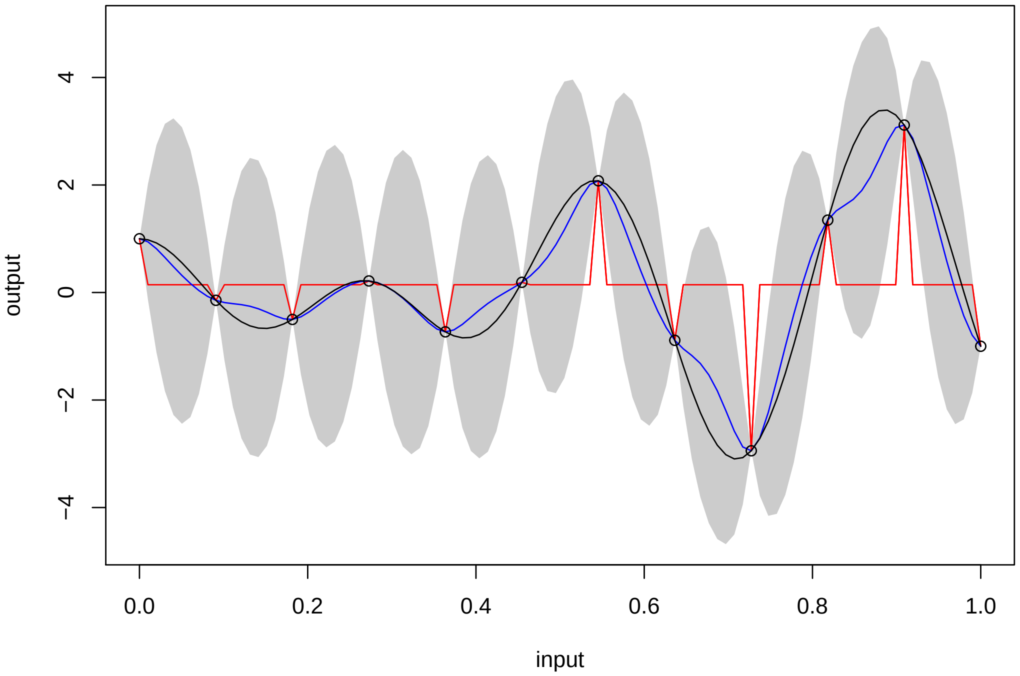


Figure 6: Emulation of the modified sine wave function with 12 design points equally spaced in $[0, 1]$. The black curve is the graph of the function and the outputs at the design points are the black circles. The blue curve is the predictive mean and the grey region is the 95% posterior credible interval obtained by the **RobustGaSP** package. The red curve is the predictive mean produced by the **DiceKriging** package.

```
Trend  coeff.:
               Estimate
 (Intercept)     0.1443

Covar. type  : matern5_2
Covar. coeff.:
                Estimate
theta(design)     0.0000

Variance estimate: 2.327824
```

A big difference between two packages is the estimated range parameter, which is found to be around 0.04 in the **RobustGaSP** package, whereas it is found to be very close to zero in the **DiceKriging** package. To see which estimate is better, we perform prediction on 100 test points, equally spaced in $[0, 1]$.

```
R> testing_input <- as.matrix(seq(0, 1, 1/99))
R> m.predict <- predict(m, testing_input)
R> dk.predict <- predict(dk, testing_input, type='UK')
```

The emulation results are plotted in Figure 6. Note that the red curve from the **DiceKriging** package degenerates to the fitted mean with spikes at the design points. This unsatisfying phenomenon, discussed in Gu et al. (2018), happens when the estimated covariance matrix is close to an identity matrix, i.e., $\hat{\mathbf{R}} \approx \mathbf{I}_n$, or equivalently $\hat{\gamma}$ tends to 0. Repeated runs of the **DiceKriging** package under different initializations yielded essentially the same results.

The predictive mean from the **RobustGaSP** package is plotted as the blue curve in Figure 6 and is quite accurate as an estimate of the true function. Note, however, that the uncertainty in this prediction is quite large, as shown by the wide 95% posterior credible regions.

In this example, adding a nugget is not helpful in **DiceKriging**, as the problem is that $\hat{\mathbf{R}} \approx \mathbf{I}_n$; adding a nugget is only helpful when the correlation estimate is close to a singular matrix (i.e.,

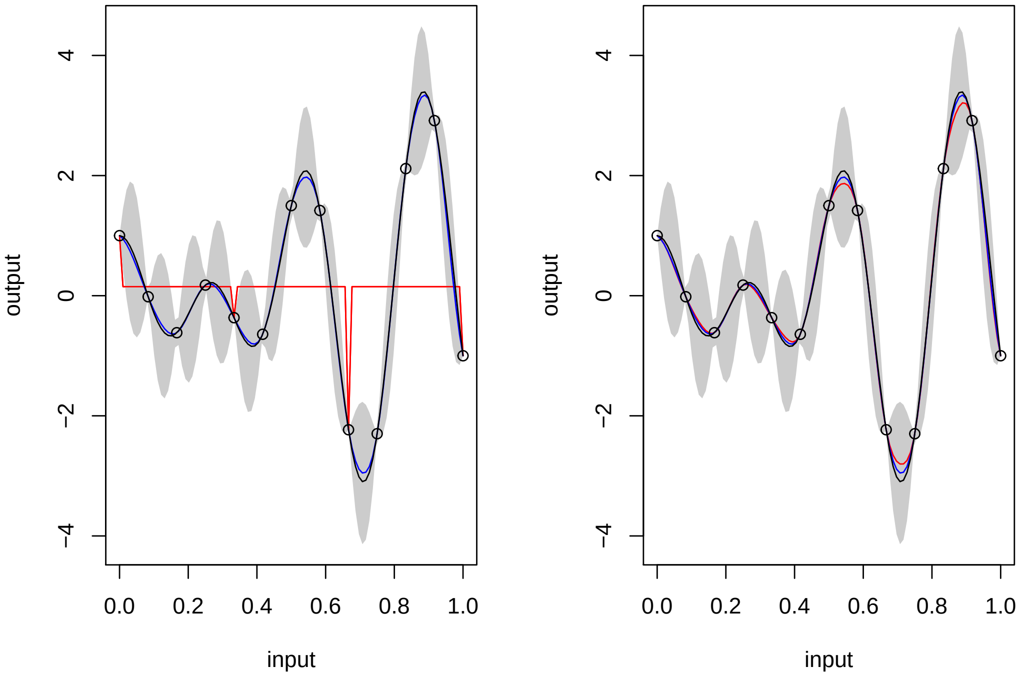


Figure 7: Emulation of the modified sine wave function with 13 design points equally spaced in $[0, 1]$. The black curve is the graph of the function and the outputs at the design points are the black circles. The blue curve is the predictive mean and the grey region is the 95% posterior credible interval found by **RobustGaSP**. The red curve is the predictive mean obtained by **DiceKriging**. The left panel and the right panel are two runs from **DiceKriging**, with different convergences of the optimization algorithm.

$\mathbf{R} \approx \mathbf{1}_n\mathbf{1}_n^\top$). However, increasing the sample size is helpful for the parameter estimation. Indeed, emulation of the modified sine wave function using 13 equally spaced design points in $[0, 1]$ was successful for one run of **DiceKriging**, as shown in the right panel of Figure 7. However, the left panel in Figure 7 gives another run of **DiceKriging** for this data, and this one converged to the problematical $\gamma \approx 0$. The predictive mean from **RobustGaSP** is stable. Interestingly, the uncertainty produced by **RobustGaSP** decreased markedly with the larger number of design points.

It is somewhat of a surprise that even emulation of a smooth one-dimensional function can be problematical. The difficulties with a multi-dimensional input space can be considerably greater, as indicated in the next example.

The Friedman function

The Friedman function was introduced in Friedman (1991) and is given by

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5,$$

where $x_i \in [0, 1]$ for $i = 1, ..., 5$. 40 design points are drawn from maximin LH samples. A GaSP model is fitted using the **RobustGaSP** package and the **DiceKriging** package with the constant mean basis function (i.e., $h(\mathbf{x}) = 1$).

```
R> input <- maximinLHS(n=40, k=5)
R> num_obs <- dim(input)[1]
R> output <- rep(0, num_obs)
R> for(i in 1:num_obs) {
+   output[i] <- friedman.5.data(input[i,])
+ }
R> m <-rgasp(design=input, response=output)
R> dk <- km(design=input, response=output)
```

Prediction on 200 test points, uniformly sampled from $[0,1]^5$, is then performed.

```
R> dim_inputs <- dim(input)[2]
R> num_testing_input <- 200
R> testing_input <- matrix(runif(num_testing_input * dim_inputs),
+                          num_testing_input, dim_inputs)
R> m.predict <- predict(m, testing_input)
R> dk.predict <- predict(dk, testing_input, type='UK')
```

To compare the performance, we calculate the root mean square errors (RMSE) for both methods,

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n^*}(\hat{y}(\mathbf{x}_i^*) - y(\mathbf{x}_i^*))^2}{n^*}},$$

where $y(\mathbf{x}_i^*)$ is the i^{th} held-out output and $\hat{y}(\mathbf{x}_i^*)$ is the prediction for $\mathbf{x}_i^*$ by the emulator, for $i = 1, ..., n^*$.

```
R> testing_output <- matrix(0, num_testing_input, 1)
R> for(i in 1:num_testing_input) {
+   testing_output[i] < -friedman.5.data(testing_input[i,])
+ }
R> m.rmse <- sqrt(mean((m.predict$mean - testing_output)^2))
R> m.rmse

[1] 0.2812935

R> dk.rmse <- sqrt(mean((dk.predict$mean - testing_output)^2))
R> dk.rmse

[1] 0.8901442
```

Thus the RMSE from **RobustGaSP** is 0.28, while the RMSE from **RobustGaSP** is 0.89. The predictions versus the real outputs are plotted in Figure 8. The black circles correspond to the predictive means from the **RobustGaSP** package and are closer to the real output than the red circles produced by the **DiceKriging** package. Since both packages use the same correlation and mean function, the only difference lies in the method of parameter estimation, especially estimation of the range parameters, $\boldsymbol{\gamma}$. The **RobustGaSP** package seems to do better, leading to much smaller RMSE in out-of-sample prediction.

The Friedman function has a linear trend associated with the 4^{th} and the 5^{th} inputs (but not the first three) so we use this example to illustrate specifying a trend in the GaSP model. For realism (one rarely actually knows the trend for a computer model), we specify a linear trend for all variables; thus we use $\mathbf{h}(\mathbf{x}) = (1, \mathbf{x})$, where $\mathbf{x} = (x_1, ..., x_5)$ and investigate whether or not adding this linear trend to all inputs is helpful for the prediction.

```
R> colnames(input) <- c("x1", "x2", "x3", "x4", "x5")
R> trend.rgasp <- cbind(rep(1, num_obs), input)
R> m.trend <- rgasp(design=input, response=output, trend=trend.rgasp)
R> dk.trend <- km(formula ~ x1 + x2 + x3 + x4 + x5, design=input, response=output)
R> colnames(testing_input) <- c("x1", "x2", "x3", "x4", "x5")
R> trend.test.rgasp <- cbind(rep(1, num_testing_input), testing_input)
R> m.trend.predict <- predict(m.trend, testing_input,
+    testing_trend=trend.test.rgasp)
R> dk.trend.predict <- predict(dk.trend, testing_input, type='UK')
R> m.trend.rmse <- sqrt(mean( (m.trend.predict$mean - testing_output)^2))
R> m.trend.rmse

[1] 0.1259403

R> dk.trend.rmse <- sqrt(mean((dk.trend.predict$mean - testing_output)^2))
R> dk.trend.rmse

[1] 0.8468056
```

Adding a linear trend does improve the out-of-sample prediction accuracy of the **RobustGaSP** package; the RMSE decreases to 0.13, which is only about one third of the RMSE of the previous model with the constant mean. However, the RMSE using the **DiceKriging** package with a linear

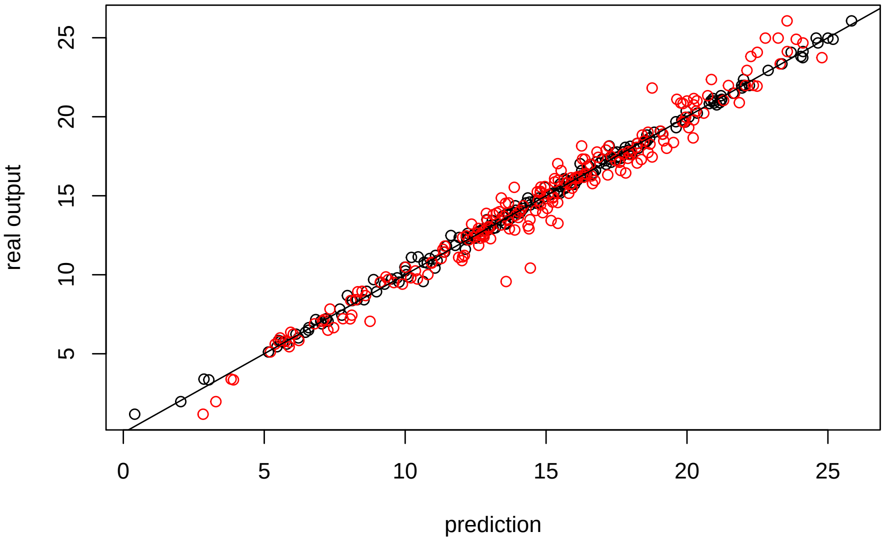


Figure 8: Prediction of 200 held-out test points of the Friedman Function based on 40 maximin LH samples. The y-axis is the real output and the x-axis is the prediction. The black circles are the predictive mean from **RobustGaSP** and the red circles are the predictive mean from **DiceKriging**. A constant mean basis function is used, i.e., $h(\mathbf{x}) = 1$.

mean increases to 0.85, more than 6 times larger than that for the **RobustGaSP**. (That the RMSE actually increased for **DiceKriging** is likely due to the additional difficulty of parameter estimation, since now the additional linear trend parameters needed to be estimated; in contrast, for **RobustGaSP**, the linear trend parameters are effectively eliminated through objective Bayesian integration.) The predictions against the real output are plotted in Figure 9. The black circles correspond to the predictive means from the **RobustGaSP** package, and are an excellent match to the real outputs.

In addition to point prediction, it is of interest to evaluate the uncertainties produced by the emulators, through study of out-of-sample coverage of the resulting credible intervals and their average lengths,

$$P_{CI}(95\%) = \frac{1}{n^*}\sum_{i=1}^{n^*} 1\{y_j(\mathbf{x}_i^*) \in \mathrm{CI}_i(95\%)\},$$

$$L_{CI}(95\%) = \frac{1}{n^*}\sum_{i=1}^{n^*} \mathrm{length}\{\mathrm{CI}_i(95\%)\},$$

where $\mathrm{CI}_i(95\%)$ is the 95% posterior credible interval. An ideal emulator would have $P_{CI}(95\%)$ close to the 95% nominal level and a short average length. We first show $P_{CI}(95\%)$ and $L_{CI}(95\%)$ for the case of a constant mean basis function.

```
R> prop.m <- length(which((m.predict$lower95 <= testing_output)
+            & (m.predict$upper95 >= testing_output))) / num_testing_input
R> length.m <- sum(m.predict$upper95 - m.predict$lower95) / num_testing_input
R> prop.m

[1] 0.97

R> length.m

[1] 1.122993

R> prop.dk <- length(which((dk.predict$lower95 <= testing_output)
+                     & (dk.predict$upper95 >= testing_output))) / num_testing_input
R> length.dk <- sum(dk.predict$upper95 - dk.predict$lower95) / num_testing_input
R> prop.dk
```

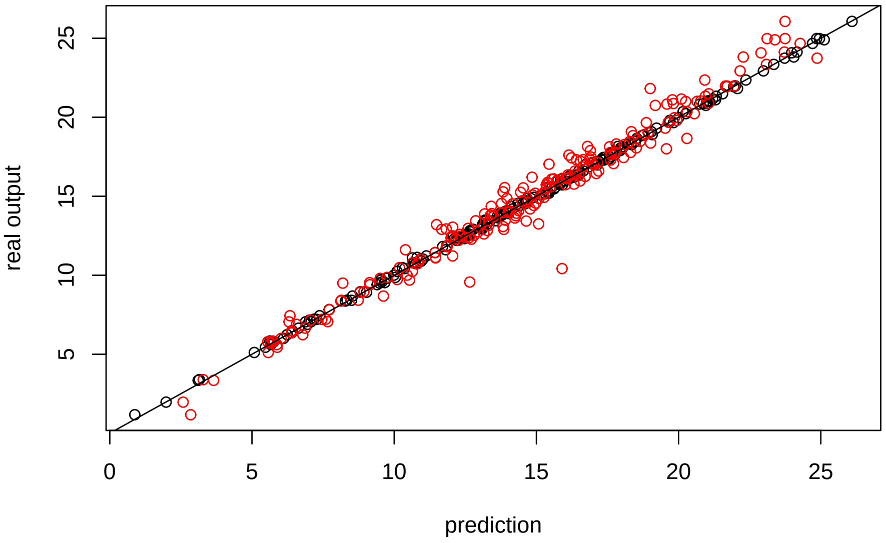


Figure 9: Prediction of 200 held-out test points for the Friedman Function based on 40 maximin LH design points. The y-axis is the real output and the x-axis is the prediction. The black circles are the predictive means obtained from **RobustGaSP**, and the red circles are the predictive means obtained from the **DiceKriging** package. In both cases, linear terms are assumed for the mean basis function, i.e., $\mathbf{h}(\mathbf{x}) = (1, \mathbf{x})$.

```
[1] 0.97

R> length.dk

[1] 3.176021
```

The $P_{CI}(95\%)$ obtained by the **RobustGaSP** is 97%, which is close to the 95% nominal level; and $L_{CI}(95\%)$, the average lengths of the 95% credible intervals, is 1.12. In contrast, the coverage of credible intervals from **DiceKriging** is also 97%, but this is achieved by intervals that are, on average, about three times longer than those produced by **RobustGaSP**.

When linear terms are assumed in the basis function of the GaSP emulator, $\mathbf{h}(\mathbf{x}) = (1, \mathbf{x})$,

```
R> prop.m.trend <- length(which((m.trend.predict$lower95 <= testing_output)
+          &(m.trend.predict$upper95 >= testing_output))) / num_testing_input
R> length.m.trend <- sum(m.trend.predict$upper95 -
+                        m.trend.predict$lower95) / num_testing_input
R> prop.m.trend

[1] 1

R> length.m.trend

[1] 0.8392971

R> prop.dk.trend <- length(which((dk.trend.predict$lower95 <= testing_output)
+         & (dk.trend.predict$upper95 >= testing_output))) / num_testing_input
R> length.dk.trend <- sum(dk.trend.predict$upper95 -
+                           dk.trend.predict$lower95) / num_testing_input
R> prop.dk.trend

[1] 0.985

R> length.dk.trend

[1] 3.39423
```

The $P_{CI}(95\%)$ for **RobustGaSP** is 100% and $L_{CI}(95\%) = 0.839$, a significant improvement over the case of a constant mean. (The coverage of 100% is too high, but at least is conservative and is achieved with quite small intervals.) For **DiceKriging**, the coverage is 98.5% with a linear mean, but the average interval size is now around 4 times as those produced by **RobustGaSP**.

To see whether or not the differences in performance persists when the sample size increases, the same experiment was run on the two emulators with sample size $n = 80$. When the constant mean function is used, the RMSE obtained by the **RobustGaSP** package and the **DiceKriging** package were 0.05 and 0.32, respectively. With $\mathbf{h}(\mathbf{x}) = (1, \mathbf{x})$, the RMSE's were 0.04 and 0.34, respectively. Thus the performance difference remains and is even larger, in a proportional sense, than when the sample size is 40.

DIAMOND computer model

We illustrate the PP GaSP emulator through two computer model data sets. The first testbed is the 'diplomatic and military operations in a non-warfighting domain' (DIAMOND) computer model (Taylor and Lane (2004)). For each given set of input variables, the dataset contains daily casualties from the 2nd and 6th day after the earthquake and volcanic eruption in Giarre and Catania. The input variables are 13-dimensional, including the speed of helicopter cruise and ground engineers, hospital and food supply capacity. The complete list of the input variables and the full data set are given in Overstall and Woods (2016).

The **RobustGaSP** package includes a data set from the DIAMOND simulator, where the training and test output both contain the outputs from 120 runs of the computer model. The following code fit a PP GaSP emulator on the training data using 3 initial starting points to optimize the kernel parameters and an estimated nugget in the PP GaSP model. We then make prediction on the test inputs using the constructed PP GaSP emulator.

```
R> data(humanity_model)
R> m.ppgasp <- ppgasp(design=humanity.X ,response=humanity.Y,
+       nugget.est=TRUE, num_initial_values=3)
R> m_pred <- predict(m.ppgasp, humanity.Xt)
R> sqrt(mean((m_pred$mean - humanity.Yt)^2))

[1] 294.9397

R> sd(humanity.Yt)

[1] 10826.49
```

The predictive RMSE of the PP GaSP emulator is 294.9397, which is much smaller than the standard deviation of the test data. Further exploration shows the output has strong positive correlation with the 11th input (food capacity). We then fit another PP GaSP model where the food capacity is included in the mean function.

```
R> n < -dim(humanity.Y)[1]
R> n_testing=dim(humanity.Yt)[1]
R> H <- cbind(matrix(1, n, 1), humanity.X$foodC)
R> H_testing <- cbind(matrix(1, n_testing, 1), humanity.Xt$foodC)
R> m.ppgasp_trend <- ppgasp(design=humanity.X, response=humanity.Y, trend=H,
+        nugget.est=TRUE, num_initial_values=3)
R> m_pred_trend <- predict(m.ppgasp_trend, humanity.Xt, testing_trend=H_testing)
R> sqrt(mean((m_pred_trend$mean - humanity.Yt)^2))

[1] 279.6022
```

The above result indicates the predictive RMSE of the PP GaSP emulator becomes smaller when the food capacity is included in the mean function. We also fit GaSP emulators by the **DiceKriging** package independently for each daily output. We include the following two criteria.

$$P_{CI}(95\%) = \frac{1}{kn^*} \sum_{i=1}^{k} \sum_{j=1}^{n^*} 1\{y_i^*(\mathbf{x}_j^*) \in CI_{ij}(95\%)\}\,,$$

$$L_{CI}(95\%) = \frac{1}{kn^*} \sum_{i=1}^{k} \sum_{j=1}^{n^*} \text{length}\{CI_{ij}(95\%)\}\,,$$

	RMSE	$P_{CI}(95\%)$	$L_{CI}(95\%)$
Independent GaSP emulator constant mean	720.16	0.99000	3678.5
Independent GaSP emulator selected trend	471.10	0.96667	2189.8
PP GaSP constant mean	294.94	0.95167	1138.3
PP GaSP selected trend	279.60	0.95333	1120.6

Table 4: Predictive performance between the independent GaSP emulator by the **DiceKriging** package (first two rows) and PP GaSP emulator by the **RobustGaSP** package (last two rows). The selected trend means the food capacity input is included in the mean function of the emulator, whereas the constant mean denotes the constant mean function. An estimated nugget is included in all methods. The baseline RMSE is 10817.47 using the mean of the output to predict.

where for $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant n^*$, $y_i^*(\mathbf{x}_j^*)$ is the held-out test output of the i^{th} run at the j^{th} day; $\hat{y}_i^*(\mathbf{x}_j^*)$ is the corresponding predicted value; $CI_{ij}(95\%)$ is the 95% predictive credible interval; and length$\{CI_{ij}(95\%)\}$ is the length of the 95% predictive credible interval. An accurate emulator should have the $P_{CI}(95\%)$ close to the nominal 0.95 and have small $L_{CI}(95\%)$ (the average length of the predictive credible interval).

The predictive accuracy by the independent GaSP emulator by the **DiceKriging** and the PP GaSP emulator for the DIAMOND computer model is recorded in Table 4. First, we noticed the predictive accuracy of both emulators seems to improve with the food capacity included in the mean function. Second, the PP GaSP seems to have much lower RMSE than the Independent GaSP emulator by the **DiceKriging** in this example, even though the kernel used in both packages are the same. One possible reason is that estimated kernel parameters by the marginal posterior mode from the **RobustGaSP** are better. Nonetheless, the PP GaSP is a restricted model, as the covariance matrix is assumed to be the same across each output variable (i.e. casualties at each day in this example). This assumption may be unsatisfying for some applications, but the improved speed in computation can be helpful. We illustrate this point by the following example for the TITAN2D computer model.

TITAN2D computer model

In this section, we discuss an application of emulating the massive number of outputs on the spatio-temporal grids from the TITAN2D computer model (Patra et al. (2005); Bayarri et al. (2009)). The TITAN2D simulates the volcanic eruption from Soufrière Hill Volcano on Montserrat island for a given set of input, selected to be the flow volume, initial flow direction, basal friction angle, and interval friction angle. The output concerned here are the maximum pyroclastic flow heights over time at each spatial grid. Since each run of the TITAN2D takes between 1 to 2 hours, the PP GaSP emulator was developed in Gu and Berger (2016) to emulate the outputs from the TITAN2D. The data from the TITAN2D computer model can be found in https://github.com/MengyangGu/TITAN2D.

The following code will load the TITAN2D data in R:

```
R> library(repmis)
R> source_data("https://github.com/MengyangGu/TITAN2D/blob/master/TITAN2D.rda?raw=True")

[1] "input_variables"          "pyroclastic_flow_heights"
[3] "loc_index"

> rownames(loc_index)

[1] "crater"          "small_flow_area" "Belham_Valley"
```

The data contain three data frames. The input variables are a 683 × 4 matrix, where each row is a set of input variables for each simulated run. The output pyroclastic flow heights is a 683 × 23040 output matrix, where each row is the simulated maximum flow heights on 144 × 160 grids. The index of the location has three rows, which records the index set for the crater area, small flow area and Belham Valley.

We implement the PP GaSP emulator in the **RobustGaSP** package and test on the TITAN2D data herein. We use the first 50 runs to construct the emulator and test it on the latter 633 runs. As argued in Gu and Berger (2016), almost no one is interested in the hazard assessment in the crater area. Thus we test our emulator for two regions with habitat before. The first one is the Belham Valley (a northwest region to the crater of the Soufrière Hill Volcano. The second region is the "non-crater" area, where we consider all the area after deleting the crater area. We also delete all locations where all the outputs are zero (meaning no flow hits the locations in the training data). For those locations, one may predict the flow height to be zero.

The following code will fit the PP GaSP emulator and make predictions on the Balham Valley area for each set of held out output.

```
R> input <- input_variables[1:50, ]
R> testing_input <- input_variables[51:683, ]
R> output <- pyroclastic_flow_heights[1:50, which(loc_index[3,]==1)]
R> testing_output <- pyroclastic_flow_heights[51:683, which(loc_index[3,]==1)]
R> n=dim(output)[1]
R> n_testing <- dim(testing_output)[1]
##delete those location where all output are zero
R> index_all_zero <- NULL
R> for(i_loc in 1: dim(output)[2]) {
+   if(sum(output[ ,i_loc]==0)==50) {
+     index_all_zero <- c(index_all_zero, i_loc)
+   }
+ }
##transforming the output
R> output_log_1 <- log(output+1)
R> m.ppgasp <- ppgasp(design=input[,1:3], response=as.matrix(output_log_1[ ,-index_all_zero]),
+   trend=cbind(rep(1, n),input[,1]), nugget.est=TRUE,max_eval=100, num_initial_values=3)
R> pred_ppgasp=predict.ppgasp(m.ppgasp, testing_input[ ,1:3],
+   testing_trend=cbind(rep(1, n_testing), testing_input[,1]))
R> m_pred_ppgasp_mean <- exp(pred_ppgasp$mean)-1
R> m_pred_ppgasp_LB <- exp(pred_ppgasp$lower95)-1
R> m_pred_ppgasp_UB <- exp(pred_ppgasp$upper95)-1
R> sqrt(mean(((m_pred_ppgasp_mean - testing_output_nonallzero)^2)))
```

```
[1] 0.2999377
```

In the above code, we fit the model using the transformed output and the first three inputs, as the fourth input (internal friction input) has almost no effect on the output. We also transform it back for prediction. As the fourth input is not used for emulation, we add a nugget to the model. The flow volume is included to be in the mean function, as the flow volume is positively correlated with the flow heights in all locations. These settings were used in Gu and Berger (2016) for fitting the PP GaSP emulator to emulate the TITAN2D computer model. The only function we have not implemented in the current version of the **RobustGaSP** package is the "periodic folding" technique for the initial flow angle, which is a periodic input. This method will appear in a future version of the package.

We compare the PP GaSP emulator with the independent GaSP emulator by the **DiceKriging** package with the same choice of the kernel function, mean function and transformation in the output. The PP GaSP emulator performs slightly better in terms of the predictive RMSE and the data covered in the 95% predictive credible interval by the PP GaSP is also slightly closer to the nominal 95% level.

The biggest difference is the computational time for these examples. The computational complexity by the independent GaSP emulator by the **DiceKriging** package is $O(kn^3)$, as it fits k emulators independently for the outputs at k spatial grid. In comparison, the computational complexity by the PP GaSP is the maximum of $O(n^3)$ and $O(kn^2)$. When $k \gg n$, the computational time of the PP GaSP is dominated by $O(kn^2)$, so the computational improvement in this example is thus obvious. Note that n is only 50 here. The ratio of the computational time between the independent GaSP and PP GaSP gets even larger when n increases.

We have to acknowledge that, however, the PP GaSP emulator assumes the same covariance

Belham Valley	RMSE	$P_{CI}(95\%)$	$L_{CI}(95\%)$	Time (s)
Independent GaSP emulator	0.30166	0.91100	0.52957	294.43
PP GaSP	0.29994	0.93754	0.59474	4.4160
Non-crater area	RMSE	$P_{CI}(95\%)$	$L_{CI}(95\%)$	Time (s)
Independent GaSP emulator	0.33374	0.91407	0.53454	1402.04
PP GaSP	0.32516	0.94855	0.60432	20.281

Table 5: Predictive performance between the independent GaSP emulator by the **DiceKriging** package and PP GaSP emulator by the **RobustGaSP** package for the outputs of the TITAN2D computer model in the Belham Valley and non-crater area. 50 runs were used to fit the emulators and the 633 runs were used as the held-out test outputs. The $RMSE$, $P_{CI}(95\%)$, $L_{CI}(95\%)$ and the computational time in seconds are shown in the second column to the fifth column for each method, respectively.

matrix across all output vector and estimate the kernel parameters using all output data. This assumption may not be satisfied in some applications. We do not believe that the PP GaSP emulator performs uniformly better than the independent GaSP emulator. Given the computational complexity and predictive accuracy shown in the two real examples discussed in this paper, the PP GaSP emulator can be used as a fast surrogate of a computer model with massive output.

Concluding remarks

Computer models are widely used in many applications in science and engineering. The Gaussian stochastic process emulator provides a fast surrogate for computationally intensive computer models. The difficulty of parameter estimation in the GaSP model is well-known, as there is no closed-form, well-behaved, estimator for the correlation parameters; and poor estimation of the correlation parameters can lead to seriously inferior predictions. The **RobustGaSP** package implements marginal posterior mode estimation of these parameters for parameterizations that satisfy the “robustness" criteria from Gu et al. (2018). Part of the advantage of this method of estimation is that the posterior has zero density for the problematic cases in which the correlation matrix is an identity matrix or the matrix or all ones. Some frequently used estimators, such as the MLE, do not have this property. Several examples have been provided to illustrate the use of the **RobustGaSP** package. Results of out-of-sample prediction suggest that the estimators in **RobustGaSP**, with small to moderately large sample sizes, perform considerably better than the MLE.

Although the main purpose of the **RobustGaSP** package is to emulate computationally intensive computer models, several functions could be useful for other purposes. For example, the `findInertInputs` function utilizes the posterior modes to find inert inputs at no extra computational cost than fitting the GaSP model. A noise term can be added to the GaSP model, with fixed or estimated variance, allowing **RobustGaSP** to analyze noisy data from either computer models or, say, spatial experiments.

While posterior modes are used for estimating the correlation parameters in the current software, it might be worthwhile to implement posterior sampling for this Bayesian model. In GaSP models, the usual computational bottleneck for such sampling is the evaluation of the likelihood, as each evaluation requires inverting the covariance matrix, which is a computation of order of $O(n^3)$, with n being the number of observations. As discussed in Gu and Xu (2017), however, exact evaluation of the likelihood for the Matérn covariance is only $O(n)$ for the case of a one-dimensional input, using the stochastic differential equation representation of the GaSP model. If this could be generalized to multi-dimensional inputs, posterior sampling would become practically relevant.

Acknowledgements

This research was supported by NSF grants DMS-1007773, DMS-1228317, EAR-1331353, and DMS-1407775. The research of Mengyang Gu was part of his PhD thesis at Duke University. The authors thank the editor and the referee for their comments that substantially improved the article.

Bibliography

J. An and A. Owen. Quasi-regression. *Journal of complexity*, 17(4):588–607, 2001. [p118]

I. Andrianakis and P. G. Challenor. The effect of the nugget on gaussian process emulators of computer models. *Computational Statistics & Data Analysis*, 56(12):4215–4228, 2012. [p112]

M. J. Bayarri, J. O. Berger, R. Paulo, J. Sacks, J. A. Cafeo, J. Cavendish, C.-H. Lin, and J. Tu. A framework for validation of computer models. *Technometrics*, 49(2):138–154, 2007. [p112]

M. J. Bayarri, J. O. Berger, E. S. Calder, K. Dalbey, S. Lunagomez, A. K. Patra, E. B. Pitman, E. T. Spiller, and R. L. Wolpert. Using statistical and computer models to quantify volcanic hazards. *Technometrics*, 51:402–413, 2009. [p113, 114, 123, 131]

R. Carnell. *Lhs: Latin Hypercube Samples*, 2016. URL `https://CRAN.R-project.org/package=lhs`. R package version 0.13. [p116]

H. Chen, J. Loeppky, J. Sacks, and W. Welch. Analysis methods for computer experiments: How to assess and what counts? *Statistical science*, 31(1):40–60, 2016. [p114]

G. M. Dancik. *Mlegp: Maximum Likelihood Estimates of Gaussian Processes*, 2013. URL `https://CRAN.R-project.org/package=mlegp`. R package version 3.1.4. [p112]

J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991. [p126]

M. Gu. *Robust Uncertainty Quantification and Scalable Computation for Computer Models with Massive Output.* PhD thesis, Duke University, 2016. [p112, 124]

M. Gu. Jointly robust prior for Gaussian stochastic process in emulation, calibration and variable selection. *Bayesian Analysis, In Press. arXiv preprint arXiv:1804.09329*, 2018. [p113, 115, 117]

M. Gu and J. O. Berger. Parallel partial Gaussian process emulation for computer models with massive output. *The Annals of Applied Statistics*, 10(3):1317–1347, 2016. [p112, 113, 120, 123, 131, 132]

M. Gu and Y. Xu. Nonseparable Gaussian stochastic process: A unified view and computational strategy. *arXiv preprint arXiv:1711.11501*, 2017. [p133]

M. Gu, J. Palomo, and J. O. Berger. *RobustGaSP: Robust Gaussian Stochastic Process Emulation*, 2016. URL `https://CRAN.R-project.org/package=RobustGaSP`. R package version 0.5.7. [p112]

M. Gu, X. Wang, and J. O. Berger. Robust Gaussian stochastic process emulation. *The Annals of Statistics*, 46(6A):3038–3066, 2018. [p112, 113, 114, 125, 133]

D. Higdon and others. Space and space-time modeling using process convolutions. *Quantitative methods for current environmental issues*, 37–56, 2002. [p116]

M. C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society B*, 63(3):425–464, 2001. [p112]

Y. Li, C. Campbell, and M. Tipping. Bayesian automatic relevance determination algorithms for classifying gene expression data. *Bioinformatics*, 18(10):1332–1339, 2002. [p120]

C. Linkletter, D. Bingham, N. Hengartner, D. Higdon, and Q. Y. Kenny. Variable selection for gaussian process models in computer experiments. *Technometrics*, 48(4):478–490, 2006. [p117]

D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. [p123]

B. MacDonald, P. Ranjan, and H. Chipman. Gpfit: An r package for fitting a gaussian process model to deterministic simulator outputs. *Journal of Statistical Software*, 64(i12), 2015. [p112]

D. J. C. MacKay. Bayesian methods for backpropagation networks. In E. Domany, J. L. van Hemmen, and K. Schulten, editors, *Models of Neural Networks III*, chapter 6, pages 211–254. Springer-Verlag, 1996. [p120]

M. D. Morris, T. J. Mitchell, and D. Ylvisaker. Bayesian design and analysis of computer experiments: Use of derivatives in surface prediction. *Technometrics*, 35(3):243–255, 1993. [p118]

R. M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer-Verlag, 1996. [p120]

J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35 (151):773–782, 1980. [p113, 123]

D. Nychka, R. Furrer, and S. Sain. **fields***: Tools for Spatial Data. R Package Version 8.4-1*, 2016. URL https://CRAN.R-project.org/package=fields. [p112]

A. M. Overstall and D. C. Woods. Multivariate emulation of computer simulators: Model selection and diagnostics with application to a humanitarian relief model. *Journal of the Royal Statistical Society C*, 65(4):483–505, 2016. [p130]

J. Palomo, R. Paulo, G. García-Donato, and others. **SAVE**: An R package for the statistical analysis of computer models. *Journal of Statistical Software*, 64(13):1–23, 2015. [p112]

A. K. Patra, A. Bauer, C. Nichita, E. B. Pitman, M. Sheridan, M. Bursik, B. Rupp, A. Webber, A. Stinton, L. Namikawa, and others. Parallel adaptive numerical simulation of dry avalanches over natural terrain. *Journal of Volcanology and Geothermal Research*, 139(1):1–21, 2005. [p113, 131]

R. Paulo, G. García-Donato, and J. Palomo. Calibration of computer models with multivariate output. *Computational Statistics & Data Analysis*, 56(12):3959–3974, 2012. [p112]

G. Pujol, B. Iooss, A. J. with contributions from Khalid Boumhaout, S. D. Veiga, J. Fruth, L. Gilquin, J. Guillaume, L. Le Gratiet, P. Lemaitre, B. Ramos, T. Touati, and F. Weber. *Sensitivity: Global Sensitivity Analysis of Model Outputs*, 2016. URL https://CRAN.R-project.org/package=sensitivity. R package version 1.12.2. [p117]

C. Ren, D. Sun, and C. He. Objective bayesian analysis for a spatial model with nugget effects. *Journal of Statistical Planning and Inference*, 142(7):1933–1946, 2012. [p120]

O. Roustant, D. Ginsbourger, and Y. Deville. Dicekriging, diceoptim: Two r packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1):1–55, 2012. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v051.i01. [p112, 114, 120]

O. Roustant, D. Ginsbourger, and Y. Deville. *DiceKriging: Kriging Methods for Computer Experiments*, 2018. URL https://CRAN.R-project.org/package=DiceKriging. R package version 1.5.6. [p124]

J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical science*, 4(4):409–423, 1989. [p112]

T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer-Verlag, 2003. [p114]

I. M. Sobol'. On sensitivity estimation for nonlinear mathematical models. *Matematicheskoe Modelirovanie*, 2(1):112–118, 1990. [p117]

I. M. Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and computers in simulation*, 55(1):271–280, 2001. [p117]

M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer-Verlag, 2012. [p114]

B. Taylor and A. Lane. Development of a novel family of military campaign simulation models. *Journal of the Operational Research Society*, 55(4):333–339, 2004. [p130]

M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1(Jun):211–244, 2001. [p120]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, 2002. [p112]

B. Worley. Deterministic uncertainty analysis, ornl-0628. *Available from National Technical Information Service*, 5285, 1987. [p118]

J. Ypma. *Nloptr: R Interface to NLopt*, 2014. URL https://CRAN.R-project.org/package=nloptr. R package version 1.0.4. [p123]

Mengyang Gu
Department of Statistics and Applied Probability
University of California, Santa Barbara
Santa Barbara, California, USA
michaelguzju@gmail.com

Jesús Palomo
Department of Business Administration
Rey Juan Carlos University
Madrid, Spain
jesus.palomo@urjc.es

James O. Berger
Department of Statistical Science
Duke University
Durham, North Carolina, USA
berger@stat.duke.edu

atable: Create Tables for Clinical Trial Reports

by Armin Ströbel

Abstract Examining distributions of variables is the first step in the analysis of a clinical trial before more specific modelling can begin. Reporting these results to stakeholders of the trial is an essential part of a statistician's work. The **atable** package facilitates these steps by offering easy-to-use but still flexible functions.

Introduction

Reporting the results of clinical trials is such a frequent task that guidelines have been established that recommend certain properties of clinical trial reports; see Moher et al. (2010). In particular, Item 17a of CONSORT states that "Trial results are often more clearly displayed in a table rather than in the text". Item 15 of CONSORT suggests "a table showing baseline demographic and clinical characteristics for each group".

The **atable** package facilitates this recurring task of data analysis by providing a short approach from data to publishable tables. The **atable** package satisfies the requirements of CONSORT statements Item 15 and 17a by calculating and displaying the statistics proposed therein, i.e. mean, standard deviation, frequencies, p-values from hypothesis tests, test statistics, effect sizes and confidence intervals thereof. Only minimal post-processing of the table is needed, which supports reproducibility. The **atable** package is intended to be modifiable: it can apply arbitrary descriptive statistics and hypothesis tests to the data. For this purpose, **atable** builds on R's S3-object system.

R already has many functions that perform single steps of the analysis process (and they perform these steps well). Some of these functions are wrapped by **atable** in a single function to narrow the possibilities for end users who are not highly skilled in statistics and programming. Additionally, users who are skilled in programming will appreciate **atable** because they can delegate this repetitive task to a single function and then concentrate their efforts on more specific analyses of the data at hand.

Context

The **atable** package supports the analysis and reporting of randomised parallel group clinical trials. Data from clinical trials can be stored in data frames with rows representing 'patients' and columns representing 'measurements' for these patients or characteristics of the trial design, such as location or time point of measurement. These data frames will generally have hundreds of rows and dozens of columns. The columns have different purposes:

- Group columns contain the treatment that the patient received, e.g. new treatment, control group, or placebo.
- Split columns contain strata of the patient, e.g. demographic data such as age, sex or time point of measurement.
- Target columns are the actual measurements of interest, directly related to the objective of the trial. In the context of ICH E9 ICH E9 (1999), these columns are called 'endpoints'.

The task is to compare the target columns between the groups, separately for every split column. This is often the first step of a clinical trial analysis to obtain an impression of the distribution of data. The **atable** package completes this task by applying descriptive statistics and hypothesis tests and arranges the results in a table that is ready for printing.

Additionally **atable** can produce tables of blank data.frames with arbitrary fill-ins (e.g. X.xx) as placeholders for proposals or report templates.

Usage

To exemplify the usage of `atable`, we use the dataset `arthritis` of **multgee** Touloumis (2015). This dataset contains observations of the self-assessment score of arthritis, an ordered variable with five categories, collected at baseline and three follow-up times during a randomised comparative study of alternative treatments of 302 patients with rheumatoid arthritis.

```
library(atable)
library(multgee)
data(arthritis)
# All columns of arthritis are numeric. Set more appropriate classes:
arthritis = within(arthritis, {
  score = ordered(y)
  baselinescore = ordered(baseline)
  time = paste0("Month ", time)
  sex = factor(sex, levels = c(1,2), labels = c("female", "male"))
  trt = factor(trt, levels = c(1,2), labels = c("placebo", "drug"))})
```

First, create a table that contains demographic and clinical characteristics for each group. The target variables are `sex`, `age` and `baselinescore`; the variable `trt` acts as the grouping variable:

```
the_table <- atable::atable(subset(arthritis, time == "Month 1"),
                            target_cols = c("age", "sex", "baselinescore"),
                            group_col = "trt")
```

Now print the table. Several functions that create a LaTeX-representation Mittelbach et al. (2004) of the table exist: `latex` of **Hmisc** Harrell Jr et al. (2018), `kable` of **knitr** Xie (2018) or `xtable` of **xtable** Dahl et al. (2018). `latex` is used for this document.

Table 1 reports the number of observations per group. The distribution of numeric variable `age` is described by its mean and standard deviation, and the distributions of categorical variable `sex` and ordered variable `baselinescore` are presented as percentages and counts. Additionally, missing values are counted per variable. Descriptive statistics, hypothesis tests and effect sizes are automatically chosen according to the class of the target column; see table 3 for details. Because the data is from a randomised study, hypothesis tests comparing baseline variables between the treatment groups are omitted.

Now, present the trial results with `atable`. The target variable is `score`, variable `trt` acts as the grouping variable, and variable `time` splits the dataset before analysis:

```
the_table <- atable(score ~ trt | time, arthritis)
```

Table 2 reports the number of observations per group and time point. The distribution of ordered variables `score` is presented as counts and percentages. Missing values are also counted per variable and group. The p-value and test statistic of the comparison of the two treatment groups are shown. The statistical tests are designed for two or more independent samples, which arise in parallel group trials. The statistical tests are all non-parametric. Parametric alternatives exist that have greater statistical power if their requirements are met by the data, but non-parametric tests are chosen for their broader range of application. The effect sizes with a 95% confidence interval are calculated; see table 3 for details.

LaTeX is not the only supported output format. All possible formats are:

- LaTeX(as shown in this document), further processed with e.g. `latex` of **Hmisc**, `kable` of **knitr** or `xtable` of **xtable**.
- HTML, further processed with e.g. `knitr::kable` of **knitr**.
- Word, can be further processed with e.g. `flextable` of **flextable** Gohel (2018).
- R's console. Human readable format meant for explorative interactive analysis.

The output format is declared by the argument `format_to` of `atable`, or globally via `atable_options`. The **settings** package van der Loo (2015) allows global declaration of various options of `atable`.

Modifying atable

The current implementation of tests and statistics (see table 3) is not suitable for all possible datasets. For example, the parametric t-test or the robust estimator median may be more adequate for some datasets. Additionally, dates and times are currently not handled by **atable**.

It is intended that some parts of **atable** can be altered by the user. Such modifications are accomplished by replacing the underlying methods or adding new ones while preserving the structures of arguments and results of the old functions. The workflow of **atable** (and the corresponding function in parentheses) is as follows:

1. calculate statistics (`statistics`)
2. apply hypothesis tests (`two_sample_htest` or `multi_sample_htest`)

Table 1: Demographics of dataset arthritis.

Group	placebo	drug
Observations		
	149	153
age		
Mean (SD)	51 (11)	50 (11)
valid (missing)	149 (0)	153 (0)
sex		
female	29% (43)	26% (40)
male	71% (106)	74% (113)
missing	0% (0)	0% (0)
baselinescore		
1	7.4% (11)	7.8% (12)
2	23% (35)	25% (38)
3	47% (70)	45% (69)
4	19% (28)	18% (28)
5	3.4% (5)	3.9% (6)
missing	0% (0)	0% (0)

Table 2: Hypothesis tests of dataset arthritis.

Group	placebo	drug	p	stat	Effect Size (CI)
Month 1					
Observations					
	149	153			
score					
1	6% (9)	1.3% (2)	0.08	9.9e+03	-0.12 (-0.24; 0.0017)
2	23% (35)	10% (16)			
3	34% (50)	50% (77)			
4	30% (45)	33% (51)			
5	6% (9)	3.3% (5)			
missing	0.67% (1)	1.3% (2)			
Month 3					
Observations					
	149	153			
score					
1	6% (9)	2% (3)	0.0065	9e+03	-0.2 (-0.32; -0.08)
2	21% (32)	18% (27)			
3	42% (63)	34% (52)			
4	24% (36)	33% (50)			
5	5.4% (8)	10% (16)			
missing	0.67% (1)	3.3% (5)			
Month 5					

Table 3: R classes, scale of measurement and atable. The table lists the descriptive statistics and hypothesis tests applied by atable to the three R classes factor, ordered and numeric. The table also reports the corresponding scale of measurement. atable treats the classes character and logical as the class factor.

R class	factor	ordered	numeric
scale of measurement	nominal	ordinal	interval
statistic	counts occurrences of every level	as factor	Mean and standard deviation
two-sample test	χ^2 test	Wilcoxon rank sum test	Kolmogorov-Smirnov test
effect size	two levels: odds ratio, else Cramér's ϕ	Cliff's Δ	Cohen's d
multi-sample test	χ^2 test	Kruskal-Wallis test	Kruskal-Wallis test

3. format statistics results (`format_statistics`)
4. format hypothesis test results (`format_tests`).

These five functions may be altered by the user by replacing existing or adding new methods to already existing S3-generics. Two examples are as follows:

Replace existing methods

The **atable** package offers three possibilities to replace existing methods:

- pass a function to `atable_options`. This affects all following calls of `atable`.
- pass a function to `atable`. This affects only a single call of `atable` and takes precedence over `atable_options`.
- replace a function in **atable**'s namespace. This is the most general possibility, as it is applicable to all R packages, but it also needs more code than the other two and is not as easily reverted.

We now define three new functions to exemplify these three possibilities.

First, define a modification of `two_sample_htest.numeric`, which applies `t.test` and `ks.test` simultaneously. See the documentation of `two_sample_htest`: the function has two arguments called `value` and `group` and returns a named list.

```
new_two_sample_htest_numeric <- function(value, group, ...){

  d <- data.frame(value = value, group = group)

  group_levels <- levels(group)
  x <- subset(d, group %in% group_levels[1], select = "value", drop = TRUE)
  y <- subset(d, group %in% group_levels[2], select = "value", drop = TRUE)

  ks_test_out <- stats::ks.test(x, y)
  t_test_out <- stats::t.test(x, y)
  out <- list(p_ks = ks_test_out$p.value,
              p_t = t_test_out$p.value )

  return(out)
}
```

Secondly define a modification of `statistics.numeric`, that calculates the median, MAD, mean and SD. See the documentation of `statistics`: the function has one argument called `x` and the ellipsis `...`. The function must return a named list.

```
new_statistics_numeric <- function(x, ...){

  statistics_out <- list(Median = median(x, na.rm = TRUE),
                         MAD = mad(x, na.rm = TRUE),
                         Mean = mean(x, na.rm = TRUE),
                         SD = sd(x, na.rm = TRUE))

  class(statistics_out) <- c("statistics_numeric", class(statistics_out))
  # We will need this new class later to specify the format
```

Third, define a modification of `format_statistics`: the median and MAD should be next to each other, separated by a semicolon; the mean and SD should go below them. See the documentation of `format_statistics`: the function has one argument called `x` and the ellipsis `...`. The function must return a data.frame with names `tag` and `value` with class factor and character, respectively. Setting a new format is optional because there exists a default method for `format_statistics` that performs the rounding and arranges the statistics below each other.

```
new_format_statistics_numeric <- function(x, ...){

  Median_MAD <- paste(round(c(x$Median, x$MAD), digits = 1), collapse = "; ")
  Mean_SD <- paste(round(c(x$Mean, x$SD), digits = 1), collapse = "; ")

  out <- data.frame(
    tag = factor(c("Median; MAD", "Mean; SD"), levels = c("Median; MAD", "Mean; SD")),
    # the factor needs levels for the non-alphabetical order
    value = c(Median_MAD, Mean_SD),
    stringsAsFactors = FALSE)
  return(out)
}
```

Now apply the three kinds of modification to `atable`: We start with **atable**'s namespace:

```
utils::assignInNamespace(x = "two_sample_htest.numeric",
                         value = new_two_sample_htest_numeric,
                         ns = "atable")
```

Here is why altering `two_sample_htest.numeric` in **atable**'s namespace works: R's lexical scoping rules state that when `atable` is called, R first searches in the enclosing environment of `atable` to find `two_sample_htest.numeric`. The enclosing environment of `atable` is the environment where it was defined, namely, **atable**'s namespace. For more details about scoping rules and environments, see e.g. Wickham (2014), section 'Environments'.

Then modify via `atable_options`:

```
atable_options('statistics.numeric' = new_statistics_numeric)
```

Then modify via passing `new_format_statistics_numeric` as an argument to `atable`, together with actual analysis. See table 4 for the results.

```
the_table <- atable(age ~ trt, arthritis,
                    format_statistics.statistics_numeric = new_format_statistics_numeric)
```

The modifications in `atable_options` are reverted by calling `atable_options_reset()`, changes in the namespace are reverted by calling `utils::assignInNamespace` with suitable arguments.

Replacing methods allows us to create arbitrary tables, even tables independent of the supplied data. We will create a table of a blank data.frame with arbitrary fill-ins (here X.xx) as placeholders. This is usefull for proposals or report templates:

Table 4: Modified atable now calculates the median, MAD, t-test and KS-test for numeric variables. The median is greater than the mean in both the drug and placebo group, indicating a skewed distribution of age. Additionally the KS-test is significant at the 5% level, while the t-test is not.

Group	placebo	drug	p_ks	p_t
Observations				
	447	459		
age				
Median; MAD	55; 10.4	53; 10.4	0.043	0.38
Mean; SD	50.7; 11.2	50.1; 11		

```
# create empty data.frame with non-empty column names
E <- atable::test_data[FALSE, ]

stats_placeholder <- function(x, ...){

  return(list(Mean = "X.xx",
              SD = "X.xx"))
}

the_table <- atable::atable(E, target_cols = c("Numeric", "Factor"),
                            statistics.numeric = stats_placeholder)
```

See table 5 for the results. This table also shows that atable accepts empty data frames without errors.

Add new methods

In the current implementation of **atable**, the generics have no method for class `Surv` of **survival** Therneau (2015). We define two new methods: the distribution of survival times is described by its mean survival time and corresponding standard error; the Mantel-Haenszel test compares two survival curves.

```
statistics.Surv <- function(x, ...){

  survfit_object <- survival::survfit(x ~ 1)

  # copy from survival:::print.survfit:
  out <- survival:::survmean(survfit_object, rmean = "common")

  return(list(mean_survival_time = out$matrix["*rmean"],
              SE = out$matrix["*se(rmean)"]))
}

two_sample_htest.Surv <- function(value, group, ...){

  survdiff_result <- survival::survdiff(value~group, rho=0)
```

Table 5: atable applied to an empty data frame with placeholder statistics for numeric variables. The placeholder-function is applied to the numeric variable, printing X.xx in the table. The empty factor variable is summarized in the same way as non-empty factors: by returning percentages and counts; in this case yielding 0/0 = NaN percent and counts of 0 in every category, as expected. Note, that the empty data frame still needs non-empty column names.

Group	value
Observations	
	0
Numeric	
Mean	X.xx
SD	X.xx
Factor	
G3	NaN% (0)
G2	NaN% (0)
G1	NaN% (0)
G0	NaN% (0)
missing	NaN% (0)

```
  # copy from survival:::print.survdiff:
  etmp <- survdiff_result$exp
  df <- (sum(1 * (etmp > 0))) - 1
  p <- 1 - stats::pchisq(survdiff_result$chisq, df)

  return(list(p = p,
              stat = survdiff_result$chisq))
}
```

These two functions are defined in the user's workspace, the global environment. It is sufficient to define them there, as R's scoping rules will eventually find them after going through the search path, see Wickham (2014).

Now, we need data with class `Surv` to apply the methods. The dataset `ovarian` of **survival** contains the survival times of a randomised trial comparing two treatments for ovarian cancer. Variable `futime` is the survival time, `fustat` is the censoring status, and variable `rx` is the treatment group.

```
library(survival)
# set classes
ovarian <- within(survival::ovarian, {time_to_event = survival::Surv(futime, fustat)})
```

Then, call `atable` to apply the statistics and hypothesis tests. See tables 6 for the results.

```
atable(ovarian, target_cols = c("time_to_event"), group_col = "rx")
```

Table 6: Hypothesis tests of the dataset ovarian.

Group	1	2	p	stat
Observations				
	13	13		
time_to_event				
mean_survival_time	650	889	0.3	1.1
SE	120	115		

Discussion

A single function call does the job, and in conjunction with report-generating packages such as **knitr**, accelerates the analysis and reporting of clinical trials.

Other R packages exist to accomplish this task:

- **furniture** Barrett et al. (2018)
- **tableone** Yoshida and Bohn. (2018)
- **stargazer** Hlavac (2018): focus is more on reporting regression models; no grouping variables, so no two-sample hypothesis tests included; and descriptive statistics are comparable to **atable**
- **DescTools** Signorell (2018): comparable functions are `Desc` (only describes data.frames, no hypothesis tests) and `PercTable` (contingency tables only).

furniture and **tableone** have high overlap with `atable`, and thus we compare their advantages relative to `atable` in greater detail:

Advantages of `furniture::table1` are:

- interacts well with **margrittr**'s pipe `%>%` Bache and Wickham (2014), as mentioned in the examples of `?table1`. This facilitates reading the code.
- handles objects defined by **dplyr**'s `group_by` to define grouping variables Wickham et al. (2019). `atable` has no methods defined for these objects.
- uses non-standard evaluation, which allows the user to create and modify variables from within the function itself, e.g.:

  ```
  table1(df, x2 = ifelse(x > 0, 1, 0)).
  ```

  This is not possible with `atable`.

Advantages of `tableone::CreateTableOne` are:

- allows arbitrary column names and prints these names in the resulting table unaltered. This is useful for generating human-readable reports. Blanks and parentheses are allowed for reporting e.g. 'Sex (Male) x%'. Also, non-ASCII characters are allowed. This facilitates reporting in languages that have little or no overlap with ASCII. `atable` demands syntactically valid names defined by `make.names`.
- counting missing values is easily switched on and off by an argument of `tableone::CreateTableOne`. In `atable` a redefinition of a function is needed.
- allows pairwise comparisons tests when data is grouped into more than two classes. `atable` allows only multivariate tests.

Advantages of `atable` are:

- options may be changed locally via arguments of `atable` and globally via `atable_options`,
- easy expansion via S3 methods,
- formula syntax,
- distinction between `split_cols` and `group_col`,
- accepts empty data.frames. This is useful when looping over a list of possibly empty data frames in subgroup analysis, see table 5,
- allows to create tables with a blank data.frame with arbitrary fill-ins (e.g. X.xx) as placeholders for proposals or report templates, also see table 5.

Changing options is exemplified in section 16.4: passing options to `atable` allows the user to modify a single `atable`-call; changing `atable_options` will affect all subsequent calls and thus spares the user passing these options to every single call.

Descriptive statistics, hypothesis tests and effect sizes are automatically chosen according to the class of the target column. R's S3-object system allows a straightforward implementation and extension of this feature, see section 16.4.

`atable` supports the following concise and self-explanatory formula syntax:

```
atable(target_cols ~ group_col | split_cols, ...)
```

R users are used to working with formulas, such as via the `lm` function for linear models. When fitting a linear model to randomised clinical trial data, one can use

```
lm(target_cols ~ group_col, ...)
```

to estimate the influence of the interventions `group_col` on the endpoint `target_cols`. `atable` mimics this syntax:

```
atable(target_cols ~ group_col, ...)
```

performs a hypothesis test, whether there is an influence of the interventions `group_col` on the endpoint `target_cols`.
Also, statisticians know the notion of conditional probability:

```
P(target_cols | split_cols).
```

This denotes the distribution of `target_cols` given `split_cols`. `atable` borrows the pipe `|` from conditional probability:

```
atable(target_cols ~ group_col | split_cols)
```

shows the distribution of the endpoint `target_cols` within the interventions `group_col` given the strata defined by `split_cols`.

`atable` distinguishes between `split_cols` and `group_col`: `group_col` denotes the randomised intervention of the trial. We want to test whether it has an influence on the `target_cols`; `split_cols` are variables that may have an influence on `target_cols`, but we are not interested in that influence in the first place. Such variables, for example, sex, age group, and time point of measurement, arise often in clinical trials. See table 2: the variable `time` is such a supplementary stratification variable: it has an effect on the arthritis score, but that is not the effect of interest; we are interested in the effect of the intervention on the arthritis score.

The package can be used in other research contexts as a preliminary unspecific analysis. Displaying the distributions of variables is a task that arises in every research discipline that collects quantitative data.

I thank the anonymous reviewer for his/her helpful and constructive comments.

Bibliography

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. [p146]

T. S. Barrett, E. Brignone, and D. J. Laxman. *Furniture: Tables for Quantitative Scientists*, 2018. URL https://CRAN.R-project.org/package=furniture. R package version 1.7.9. [p145]

D. B. Dahl, D. Scott, C. Roosen, A. Magnusson, and J. Swinton. *Xtable: Export Tables to LaTeX or HTML*, 2018. URL https://CRAN.R-project.org/package=xtable. R package version 1.8-3. [p138]

D. Gohel. *Flextable: Functions for Tabular Reporting*, 2018. URL https://CRAN.R-project.org/package=flextable. R package version 0.4.4. [p138]

F. E. Harrell Jr, with contributions from Charles Dupont, and many others. *Hmisc: Harrell Miscellaneous*, 2018. URL https://CRAN.R-project.org/package=Hmisc. R package version 4.1-1. [p138]

M. Hlavac. *Stargazer: Well-Formatted Regression and Summary Statistics Tables*. Central European Labour Studies Institute (CELSI), Bratislava, Slovakia, 2018. URL https://CRAN.R-project.org/package=stargazer. R package version 5.2.2. [p145]

ICH E9. ICH Harmonised Tripartite Guideline. Statistical Principles for Clinical Trials. International Conference on Harmonisation E9 Expert Working Group. *Statistics in medicine*, 18:1905–1942, 1999. ISSN 0277-6715. [p137]

F. Mittelbach, M. Goossens, J. Braams, D. Carlisle, and C. Rowley. *The LaTeX Companion (Tools and Techniques for Computer Typesetting)*. Addison-Wesley Professional, 2004. ISBN 0-201-36299-6. URL https://www.amazon.com/LaTeX-Companion-Techniques-Computer-Typesetting/dp/0201362996?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0201362996. [p138]

D. Moher, S. Hopewell, K. F. Schulz, V. Montori, P. C. Gøtzsche, P. J. Devereaux, D. Elbourne, M. Egger, and D. G. Altman. Consort 2010 explanation and elaboration: Updated guidelines for reporting parallel group randomised trials. *BMJ*, 340, 2010. ISSN 0959-8138. URL https://doi.org/10.1136/bmj.c869. [p137]

A. Signorell. *DescTools: Tools for Descriptive Statistics*, 2018. URL https://cran.r-project.org/package=DescTools. R package version 0.99.24. [p145]

T. M. Therneau. *A Package for Survival Analysis in S*, 2015. URL https://CRAN.R-project.org/package=survival. version 2.38. [p143]

A. Touloumis. R package multgee: A generalized estimating equations solver for multinomial responses. *Journal of Statistical Software*, 64(8):1–14, 2015. URL http://www.jstatsoft.org/v64/i08/. [p137]

M. van der Loo. *Settings: Software Option Settings Manager for R*, 2015. URL https://CRAN.R-project.org/package=settings. R package version 0.2.4. [p138]

H. Wickham. *Advanced R*. Taylor & Francis Inc, 2014. ISBN 1466586966. URL https://dx.doi.org/10.1201/b17487. [p142, 145]

H. Wickham, R. François, L. Henry, and K. Müller. *Dplyr: A Grammar of Data Manipulation*, 2019. URL https://CRAN.R-project.org/package=dplyr. R package version 0.8.0.1. [p146]

Y. Xie. *Knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2018. URL https://yihui.name/knitr/. R package version 1.20. [p138]

K. Yoshida and J. Bohn. *Tableone: Create 'Table 1' to Describe Baseline Characteristics*, 2018. URL https://CRAN.R-project.org/package=tableone. R package version 0.9.3. [p145]

Armin Ströbel
German Center for Neurodegenerative Diseases (DZNE)
Witten
Armin-Michael.Stroebel@dzne.de

Identifying and Testing Recursive vs. Interdependent Links in Simultaneous Equation Models via the SIRE Package

by Gianmarco Vacca and Maria Grazia Zoia

Abstract Simultaneous equation models (SEMs) are composed of relations which either represent unidirectional links, which entail a causal interpretation, or bidirectional links, due to feedback loops, which lead to the notion of interdependence. The issue is of prominent interest in several respects. Investigating the causal structure of a SEM, on the one hand, brings to light the theoretical assumptions behind the model and, on the other hand, pilots the choice of the befitting estimation method and of which policy to implement. This paper provides an operational method to distinguish causal relations from interdependent ones in SEMs, such as macro-econometric models, models in ecology, biology, demography, and so forth. It is shown that the causal structure of a system crucially rests on the feedback loops, which possibly affect the equations. These loops are associated to the non-null entries of the Hadamard product of matrices encoding the direct and indirect links among the SEM dependent variables. The effectiveness of feedbacks is verified with a Wald test based on the significance of the aforementioned non-null entries. An R package, **SIRE** (System of Interdependent/Recursive Equations), provides the operational completion of the methodological and analytic results of the paper. **SIRE** is applied to a macroeconomic model to illustrate how this type of analysis proves useful in clarifying the nature of the complex relations in SEMs.

Introduction

As is well known, each equation in a simultaneous equation model (SEM) represents a specific link between a dependent (endogenous) variable and a set of other variables which play an explicative role for the former. These links can reflect either one-way relations between the dependent and their explicative variables or two-ways relations, ascribable to the presence of feedback loops operating either at a systematic or a stochastic level. SEMs are of recursive type as long as the equations represent unidirectional links. Otherwise, if the equations are bidirectional, the SEM (or part of it) is interdependent. Interdependence is, both structurally connected to the presence of current endogenous variables playing an explicative role, and can result as a by-product of error-term dependencies.

Investigating the nature, causal rather than interdependent, of a SEM is important in several respects. First the analysis, unfolding the dynamics among variables, sheds more light on the rationale behind the theoretical assumptions of the model. For instance, in an economic framework, the distinction between interdependent and causal SEMs leads to models which can be traced back to two main streams of economic theory: Neoclassical and Keynesian (Bellino et al., 2018). Furthermore, the implication of interdependence vs. causality is crucial for undertaking parameter estimation, given that a set of causal equations can be estimated equation by equation by ordinary least squares (OLS), while simultaneous estimation methods, like three stage least squares (3SLS) are required when interdependence occurs. Given that large SEMs have become increasingly popular, the need for an analytical set-up, able to effectively detect and test causality versus interdependence, has of course become more urgent.

Starting from this premise and following Strotz and Wold, 1960; Wold, 1964; and more recently Faliva, 1992; Faliva and Zoia, 1994); in this paper we have devised an operational method to distinguish the causal from the interdependent equations of a SEM. Other approaches for detecting feedback-loops arising in deterministic (error free) models are based on either graph or system theory (see e.g., Gilli 1992). Our methodological proposal goes beyond the aforementioned methods, as besides covering both the cases of deterministic and error-driven feedback effects, it provides a way for testing the feedback effectiveness. In addition, it differs in principle from other approaches, as the one proposed by Granger (see Granger, 1980) and the Covariance Structural Analysis (CSA; Jöreskog). The former essentially rests on a predictability criterion for defining causality regardless of the theory behind the model. The latter, which is meant to find the best parametric approximation of the sample covariance matrix in terms of a given theoretical SEM structure; as such, it does not lead to a causal/interdependent interpretation of the model links as the one developed in our paper.

The feedbacks identified by the method proposed here demand statistical confirmation on certain empirical evidence arguments. Lack of significance of (one or more of) the estimated feedbacks can overturn the nature of the connections among model variables. To this end, a Wald type test is devised to check whether a given equation is significantly affected by feedback or not. The statistic of this test hinges on the parameter matrices of the model: the matrix associated to the endogenous variables playing an explicative role and the dispersion matrix of the error terms. If an equation is affected by feedback loops, the testing procedure allows to diagnose which endogenous variables are significantly connected in the loop of interest. Indeed, testing the significance of feedbacks means also checking if the links among variables, suggested by the theory at the basis of the model, are confirmed according to an empirical evidence argument.

The methodological approach put forth in this paper is implemented in R with the **SIRE** package. Besides integrating functions usually employed for the estimation of SEM's, the package provides new functions meant to duly split a system of equations into its unidirectional and bidirectional links, and test their significance. To our knowledge, extant alternative approaches to causality do not offer a similar test.

The paper is structured as follows. The first section provides the methodological set-up devised to single out causal and interdependent relations in a SEM. In the second section, a Wald-type test is worked out to check whether a given equation is affected by feedbacks or not. The third section shows how the method and the R code work for detecting and testing feedback-loops in a macroeconomic model. An Appendix, with proofs of the main theoretical results, completes the paper.

Detecting Loops in an Equation System

An equation system is a set of structural equations representing economic theory-driven relations linking the variables relevant to the study at hand.

It is customary to specify an equation system as follows

$$\mathbf{y}_t = \mathbf{\Gamma}\mathbf{y}_t + \mathbf{A}\mathbf{z}_t + \boldsymbol{\epsilon}_t \qquad\qquad t = 1, \ldots, T \tag{1}$$

where $\mathbf{y}_t$ is a $L \times 1$ vector of current dependent or endogenous variables, $\mathbf{z}_t$ is a $J \times 1$ vector of explicative variables and $\boldsymbol{\epsilon}_t$ is a $L \times 1$ vector of error terms. T is the sample period. $\mathbf{\Gamma}$ and $\mathbf{A}$ are, respectively, $L \times L$ and $L \times J$ sparse parameter matrices. In particular $\mathbf{\Gamma}$, expressing the relations among current endogenous variables, is a hollow matrix to prevent any endogenous variable from explaining itself. Furthermore, it is assumed that $(\mathbf{I} - \mathbf{\Gamma})$ is of full rank, meaning that the equations are linearly independent.

Error terms are assumed to be non-systematic, stationary in a wide sense, and uncorrelated over time, that is

$$\begin{aligned} E(\boldsymbol{\epsilon}_t) &= \mathbf{0}_L \\ E(\boldsymbol{\epsilon}_t\boldsymbol{\epsilon}'_\tau) &= \begin{cases} \mathbf{\Sigma}_{(L\times L)} & \text{if } t = \tau \\ \mathbf{0}_{(L\times L)} & \text{if } t \neq \tau \end{cases} \end{aligned} \tag{2}$$

Actually, the pattern of relations recognizable in an econometric model can be interpreted either in terms of causal or interdependent schemes. A causal relation among variables is an asymmetric, theoretically-grounded and predictive relations which can be ideally meant as a stimulus-response mechanism (see Wold, 1964 and Strotz and Wold 1960). The equations of a model form a causal chain when, once they are properly ordered, each current endogenous variable turns out to be, on the one hand, resultant of the joint effect of the endogenous which precede it in the chain and, on the other hand, cause of the current endogenous which follow the same endogenous in the chain. A model with equations that form a causal chain is defined recursive. The following simple equation system provides an example of a recursive model (see Figure 1, left panel)

$$\begin{aligned} y_{1,t} =& \boldsymbol{a}'_1\boldsymbol{z}_t + \epsilon_{1,t} \\ y_{2,t} =& \gamma_{2,1}y_{1,t} + \boldsymbol{a}'_2\boldsymbol{z}_t + \epsilon_{2,t} \\ y_{3,t} =& \gamma_{3,2}y_{2,t} + \gamma_{3,1}y_{1,t} + \boldsymbol{a}'_3\boldsymbol{z}_t + \epsilon_{3,t} \\ y_{4,t} =& \gamma_{4,3}y_{3,t} + \gamma_{4,1}y_{1,t} + \boldsymbol{a}'_4\boldsymbol{z}_t + \epsilon_{4,t} \end{aligned} \tag{3}$$

Recursive systems can be easily estimated, equation by equation, using OLS, starting from the top of the chain.

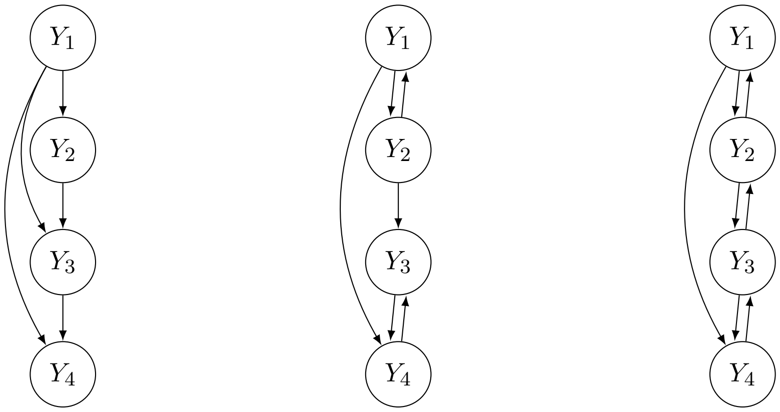
When a causal chain exists among blocks of current endogenous variables, a causal order can be established among those blocks of equations. In this case, the current endogenous variables of a block are effects of the variables belonging to the blocks which come before them in the chain, as well as the causes of the variables belonging to blocks which follow the block at stake in the chain. In this case, the model is of block-recursive type. The following simple equation system provides an example of a recursive model (see Figure 1, middle panel)

$$\begin{aligned} y_{1,t} =& \gamma_{1,2} y_{2,t} + \boldsymbol{a}_1' \boldsymbol{z}_t + \epsilon_{1,t} \\ y_{2,t} =& \gamma_{2,1} y_{1,t} + \boldsymbol{a}_2' \boldsymbol{z}_t + \epsilon_{2,t} \\ y_{3,t} =& \gamma_{3,2} y_{2,t} + \gamma_{3,4} y_{4,t} + \boldsymbol{a}_3' \boldsymbol{z}_t + \epsilon_{3,t} \\ y_{4,t} =& \gamma_{4,3} y_{3,t} + \gamma_{4,1} y_{1,t} + \boldsymbol{a}_4' \boldsymbol{z}_t + \epsilon_{4,t} \end{aligned} \tag{4}$$

Here, the chain is formed by two blocks of variables (y_1, y_2) and (y_3 and y_4) with the variables of the first block explaining those of the second.

Sometimes the composite nature of the connections among variables leads to a closed sequence of dependencies among variables to be ascribed to feedback loops. This type of interaction among endogenous variables is usually called interdependence. Interdependence is structurally connected to the presence of both current endogenous variables on the right-hand side of the model and the correlation between contemporaneous error terms.See the system below as an example in this regard (see Figure 1, right panel)

$$\begin{aligned} y_{1,t} =& \gamma_{1,2} y_{2,t} + \boldsymbol{a}_1' \boldsymbol{z}_t + \epsilon_{1,t} \\ y_{2,t} =& \gamma_{2,1} y_{1,t} + \gamma_{2,3} y_{3,t} + \boldsymbol{a}_2' \boldsymbol{z}_t + \epsilon_{2,t} \\ y_{3,t} =& \gamma_{3,2} y_{2,t} + \gamma_{3,4} y_{4,t} + \boldsymbol{a}_3' \boldsymbol{z}_t + \epsilon_{3,t} \\ y_{4,t} =& \gamma_{4,3} y_{3,t} + \gamma_{4,1} y_{1,t} + \boldsymbol{a}_4' \boldsymbol{z}_t + \epsilon_{4,t} \end{aligned} \tag{5}$$



(a) Recursive model (3). **(b)** block-recursive model (4). **(c)** interdependent model (5).

Figure 1: The three patterns of relations in a simultaneous equation model.

Based on this premise, it is clear that the causal or interdependent features of a model's equations depend on the pair of matrices $\boldsymbol{\Gamma}$ and $\boldsymbol{\Sigma}$. The former matrix highlights the possible (circular) dependencies or feedbacks among endogenous variables, while the latter features those induced by the stochastic components. In fact, the correlation of error terms associated to an equation-pair may transform the link between the endogenous, explained by these equations, into a relation with feedback.

Moreover, the essential information concerning the causal structure of a model can be obtained from the topological properties[1] of the pair of the mentioned matrices and, at the very end, from the topological properties of the associated binary matrices $\boldsymbol{\Gamma}^b$and $\boldsymbol{\Sigma}^b$. [2]

Following Faliva (Faliva, 1992) matrix $\boldsymbol{\Gamma}$ can be split as follows

$$\boldsymbol{\Gamma} = \tilde{\mathbf{C}} + \boldsymbol{\Psi}_0 \tag{6}$$

[1] The term topological properties refers to those properties of a matrix which depend exclusively on the number and the relative position of its null and non-null elements (Marimont, 1969).

[2] A binary matrix associated to a matrix $\mathbf{G}$ is a matrix whose entries are equal to 1 if the corresponding entries of $\mathbf{G}$ are non-null, or 0 otherwise. Binary matrices preserve the topological properties of the parent matrices.

where $\tilde{\mathbf{C}}$ includes the coefficients associated to current endogenous variables involved in feedback loops, and $\mathbf{\Psi}_0$ those associated to endogenous variables involved in causal relations.

Matrix $\tilde{\mathbf{C}}$ is specified as follows

$$\tilde{\mathbf{C}} = \mathbf{C} + \mathbf{\Psi}_1 \tag{7}$$

where $\mathbf{C}$ includes the feedbacks arising in the systematic part of the model and matrix $\mathbf{\Psi}_1$ those induced by the correlation of the error terms. Matrices $\mathbf{C}$ and $\mathbf{\Psi}_1$ are defined as follows

$$\mathbf{C} = \mathbf{\Gamma} * \mathbf{R} \qquad \mathbf{R} = \left\{ \left[\sum_{r=1}^{L-1} \left(\mathbf{\Gamma}^b \right)^r \right]^b \right\}' \tag{8}$$

$$\mathbf{\Psi}_1 = (\mathbf{\Gamma} - \mathbf{C}) * \left[\mathbf{\Sigma}^b \left(\mathbf{I} + \mathbf{R} \right) \right]^b, \tag{9}$$

where the symbol "$*$" denotes the Hadamard product.[3][4] The rationale of (8) hinges on the fact that a direct feedback between variables y_i and y_j corresponds to the simultaneous non-nullity of $\gamma_{i,j}$ and $\gamma_{j,i}$ of coefficient matrix $\mathbf{\Gamma}$. This entails that a direct feedback between these two variables exists if the (i,j)-th element of the matrix [5]

$$\mathbf{\Gamma} * (\mathbf{\Gamma}^b)' \tag{10}$$

is non null. An indirect feedback between the same variables is instead associated to a bidirectional connection between y_i and y_j established through other variables and equations. In algebraic terms this corresponds to the simultaneous non-nullity of the (i,j)-th element of $\mathbf{\Gamma}$ and of the (i,j)-th element of a positive power of $\mathbf{\Gamma}'$ (Fiedler, 2013). This entails that an indirect feedback exists between the mentioned variables if the (i,j)-th element of the following matrix

$$\mathbf{\Gamma}' * \left\{ \left[\sum_{r=2}^{L-1} \left(\mathbf{\Gamma}^b \right)^r \right]^b \right\}' \tag{11}$$

is non-null.

Accordingly, matrix

$$\mathbf{\Psi} = \mathbf{\Gamma} - \mathbf{C} \tag{12}$$

includes the coefficients associated to endogenous variables which, as far as the systematic aspects of the model are concerned, have a causal role.[6]

In order to show how feedbacks operating in the systematic part of a model can be detected, let

[3] The Hadamard product of two matrices, $\mathbf{A}$ and $\mathbf{B}$ of the same order, is defined as the matrix of the term-to-term products of the elements of these matrices, that is $(\mathbf{A} * \mathbf{B})_{(i,j)} = a_{(i,j)} b_{(i,j)}$.

[4] An alternative approach for determining the feedbacks operating at a systematic level in a model is based on graph theory (see Jöreskog and Wold, 1982 and Ponstein, 1966).

[5] The element $\gamma_{j,i}$ of $\mathbf{\Gamma}$ corresponds to the element $\gamma_{i,j}$ of $\mathbf{\Gamma}'$

[6] It is worth mentioning that $\mathbf{\Psi}$ is Hadamard-orthogonal to $\mathbf{C}$ (two matrices $\mathbf{A}$ and $\mathbf{B}$ are said to be Hadamard-orthogonal if $\mathbf{A} * \mathbf{B} = \mathbf{0}$). Furthermore, while matrix $\mathbf{C}$ is co-spectral to $\mathbf{\Gamma}$ (i.e., they have the same eigenvalues), matrix $\mathbf{\Psi}$ is a hollow-nilpotent matrix, like $\mathbf{\Gamma}$ (a square matrix $\boldsymbol{N}$ is nilpotent if $\boldsymbol{N}^k = \mathbf{0}$ for some $k < M$, where M is the matrix dimension). A hollow, nilpotent matrix can always be expressed in triangular form.

us consider as an example the following deterministic model

$$
\begin{aligned}
y_{1,t} &= \gamma_{1,5} y_{5,t} + \gamma_{1,7} y_{7,t} + \boldsymbol{a}_1' \boldsymbol{z}_t \\
y_{2,t} &= \boldsymbol{a}_2' \boldsymbol{z}_{2,t} \\
y_{3,t} &= \gamma_{3,11} y_{11,t} + \boldsymbol{a}_3' \boldsymbol{z}_t \\
y_{4,t} &= \gamma_{4,3} y_{3,t} + \boldsymbol{a}_4' \boldsymbol{z}_t \\
y_{5,t} &= \gamma_{5,10} y_{10,t} + \boldsymbol{a}_5' \boldsymbol{z}_t \\
y_{6,t} &= \gamma_{6,5} y_{5,t} + \gamma_{6,9} y_{9,t} + a_6' \boldsymbol{z}_t \\
y_{7,t} &= \gamma_{7,6} y_{6,t} + \boldsymbol{a}_7' \boldsymbol{z}_t \\
y_{8,t} &= \gamma_{8,12} y_{12,t} + \boldsymbol{a}_8' \boldsymbol{z}_t \\
y_{9,t} &= \gamma_{9,7} y_{7,t} + \boldsymbol{a}_9' \boldsymbol{z}_t \\
y_{10,t} &= \gamma_{10,5} y_{5,t} + \boldsymbol{a}_{10}' \boldsymbol{z}_{2,t} \\
y_{11,t} &= \gamma_{11,12} y_{12,t} + \boldsymbol{a}_{11}' \boldsymbol{z}_t \\
y_{12,t} &= \gamma_{12,4} y_{4,t} + \gamma_{12,11} y_{11,t} + \boldsymbol{a}_{12}' \boldsymbol{z}_t \\
y_{13,t} &= \gamma_{13,2} y_{2,t} + \gamma_{13,6} y_{6,t} + \boldsymbol{a}_{13}' \boldsymbol{z}_t
\end{aligned}
\tag{13}
$$

Matrix $\boldsymbol{\Gamma}^b$ is given by

$$
\boldsymbol{\Gamma}^b = \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix}
\tag{14}
$$

Using (8) and (12), $\boldsymbol{\Gamma}^b$ is split in the following two submatrices

$$
\mathbf{C}^b = \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix}, \quad
\boldsymbol{\Psi}^b = \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix}
\tag{15}
$$

Looking at matrix $\mathbf{C}^b$, we see that the simultaneous non-nullity of the $c_{5,10}$, $c_{10,5}$, $c_{11,12}$, and $c_{12,11}$ elements imply the existence of two direct feedbacks: one between the variable-pair y_5 and y_{10}, and the other between y_{11} and y_{12}. The non-nullity of the $c_{3,11}$, $c_{4,3}$, and $c_{12,4}$ elements denotes the existence of indirect feedbacks between the four variables y_3, y_4, y_{11}, and y_{12}. Similarly, variables y_6, y_7, and y_9 are connected by an (indirect) feeback as a consequence of the non-nullity of the $c_{6,9}$, $c_{7,6}$, and $c_{9,7}$ elements. Looking at matrix $\boldsymbol{\Psi}$ we conclude that variables y_5 and y_7 have a causal role in the first equation. Variables y_5 and y_{12} have the same role in the equations six and eight, while variables y_2 and y_6 play a causal role in the last equation. The results ensuing from the decomposition of $\boldsymbol{\Gamma}^b$ are depicted in Figure 2.

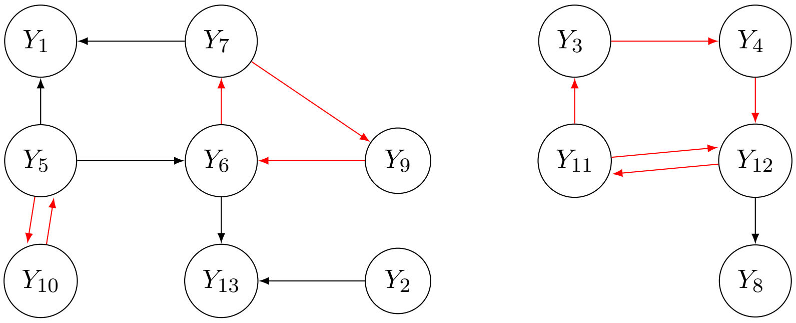


Figure 2: Interdependent links (in red) and causal links (in black) operating in the model (13).

If the error terms are correlated, the causal structure of a model could no longer match that of its systematic counterpart since part of the relations that are recursive at systematic level, namely $\boldsymbol{\Psi}\mathbf{y}_t$, may become interdependent as a consequence of the feedback mechanisms induced by the stochastic terms in $\boldsymbol{\Sigma}$. In this case, matrix $\boldsymbol{\Psi}$ turns out to be the sum of two Hadamard-orthogonal matrices, $\boldsymbol{\Psi}_0$ and $\boldsymbol{\Psi}_1$, that is

$$\boldsymbol{\Psi} = \boldsymbol{\Psi}_0 + \boldsymbol{\Psi}_1 \qquad \boldsymbol{\Psi}_0 * \boldsymbol{\Psi}_1 = \mathbf{0}_{(L\times L)} \tag{16}$$

where

$$\boldsymbol{\Psi}_1^b = \boldsymbol{\Psi} * \mathbf{F} \qquad \mathbf{F} = \left[\boldsymbol{\Sigma}^b\left(\mathbf{I} + \mathbf{R}\right)\right]^b \tag{17}$$

Here, matrix $\boldsymbol{\Psi}_1$ includes the coefficients associated to the endogenous variables involved in loops induced by disturbances. In fact, it can be proved (see 1. in Appendix) that the matrix $\left[\boldsymbol{\Sigma}^b\left(\mathbf{I} + \mathbf{R}\right)\right]^b$ is the binary counterpart of the covariance matrix between the error terms and the endogenous variables given by

$$\mathbb{E}(\boldsymbol{\epsilon}_t'\mathbf{y}_t) = \left[\boldsymbol{\Sigma}\left(\mathbf{I} - \boldsymbol{\Gamma}\right)^{-1}\right] \tag{18}$$

The non-null elements of the above matrix express the effect of the model's left-hand side (LHS) endogenous variables on the right-hand side (RHS) ones, which are induced by the error term correlation.

Equations (16) and (17) rest on the logical relations between the concepts of causality and predictability, where the notion of optimal predictor (in mean-square sense) tallies with that of conditional expectation. In fact, given that causal relations are also predictive, but not vice-versa, we can define as causal those relations that are both causal in the deterministic model and predictive in a stochastic context. This means that if the conditional expectations of the relations, which are causal in the deterministic model, namely $\boldsymbol{\Psi}\mathbf{y}_t$, are not affected by the error terms, then $\boldsymbol{\Psi}\mathbf{y}_t$ turns out to also have a causal role in a stochastic context. Accordingly, we can say that the stochastic specification is neutral with respect to the underlying systematic causal structure if the following holds (Faliva, 1992)

$$\mathbb{E}(\boldsymbol{\Psi}\mathbf{y}_t + \boldsymbol{\epsilon}_t|\boldsymbol{\Psi}\mathbf{y}_t) = \boldsymbol{\Psi}\mathbf{y}_t + \mathbb{E}(\boldsymbol{\epsilon}_t|\boldsymbol{\Psi}\mathbf{y}_t) = \boldsymbol{\Psi}\mathbf{y}_t \tag{19}$$

meaning that

$$\mathbb{E}(\boldsymbol{\epsilon}_t|\boldsymbol{\Psi}\mathbf{y}_t) = \mathbf{0} \tag{20}$$

Otherwise, the correlation between the error terms and the endogenous variables may affect the conditional expectation of the error term as follows (see Faliva, 1992)

$$\mathbb{E}(\boldsymbol{\epsilon}_t|\boldsymbol{\Psi}\mathbf{y}_t) = -\boldsymbol{\Psi}_1\mathbf{y}_t \tag{21}$$

which, in turn, implies that

$$\mathbb{E}(\boldsymbol{\Psi}\mathbf{y}_t + \boldsymbol{\epsilon}_t|\boldsymbol{\Psi}\mathbf{y}_t) = \boldsymbol{\Psi}\mathbf{y}_t - \boldsymbol{\Psi}_1\mathbf{y}_t = \boldsymbol{\Psi}_0\mathbf{y}_t \tag{22}$$

In this case, only the subset $\boldsymbol{\Psi}_0\mathbf{y}_t$ of the original set of causal relations, playing a predictive role, is causal. This, in turn, implies that the overall feedback operating in the system is included in matrix $\mathbb{C} = \mathbf{C} + \boldsymbol{\Psi}_1$.

To highlight the role played by the stochastic specification on the model causal structure, let us consider as an example the following specification for matrix $\boldsymbol{\Sigma}^b$

$$\boldsymbol{\Sigma}^b = \begin{bmatrix}
1 &&&&&&&&&&&& \\
\cdot & 1 &&&&&&&&&&& \\
\cdot & \cdot & 1 &&&&&&&&&& \\
1 & 1 & \cdot & 1 &&&&&&&&& \\
1 & \cdot & \cdot & 1 & 1 &&&&&&&& \\
\cdot & 1 & 1 & 1 & \cdot & 1 &&&&&&& \\
\cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 &&&&&& \\
1 & 1 & \cdot & 1 & 1 & 1 & \cdot & 1 &&&&& \\
\cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & 1 & 1 &&&& \\
1 & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & 1 &&& \\
\cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & 1 && \\
1 & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \\
\cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & 1 & \cdot & 1
\end{bmatrix} \tag{23}$$

Then, matrices $\widetilde{\mathbf{C}^b}$ and $\boldsymbol{\Psi}_0^b$ are

$$\widetilde{\mathbf{C}^b} = \begin{bmatrix}
\cdot&\cdot&\cdot&\cdot&1&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot \\
\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot \\
\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot
\end{bmatrix} = \mathbf{C}^b + \boldsymbol{\Psi}_1^b = \begin{bmatrix}
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot \\
\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot \\
\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot
\end{bmatrix} + \begin{bmatrix}
\cdot&\cdot&\cdot&\cdot&1&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\
\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot
\end{bmatrix} \tag{24}$$

$$\mathbf{\Psi}_0^b = \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix} \tag{25}$$

The non-null correlation between the pairs of error terms $\{\epsilon_5, \epsilon_1\}$, $\{\epsilon_6, \epsilon_{13}\}$, $\{\epsilon_7, \epsilon_1\}$ and $\{\epsilon_{12}, \epsilon_8\}$ (see Equation (23)) has transformed the relations among the pairs of variables $\{y_5, y_1\}$, $\{y_6, y_{13}\}$, $\{y_7, y_1\}$, and $\{y_{12}, y_8\}$, which were causal in the deterministic model (13), into interdependent links. Figure 3 shows the effect of the stochastic specification (23) on the feedbacks originally detected in the deterministic model (13).

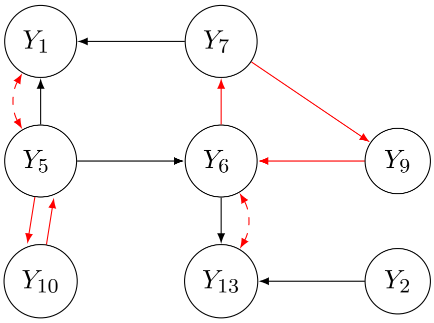


Figure 3: Interdependent (in red) and causal (in black) links operating in the model (13) when the stochastic specification is as in (23). Dashed red lines with double-headed arrows denote interdependent links induced by the correlation of the error terms.

The flow-chart in Figure 4 shows the different cases, according to the structure of matrices $\mathbf{\Gamma}$ and $\mathbf{\Sigma}$.

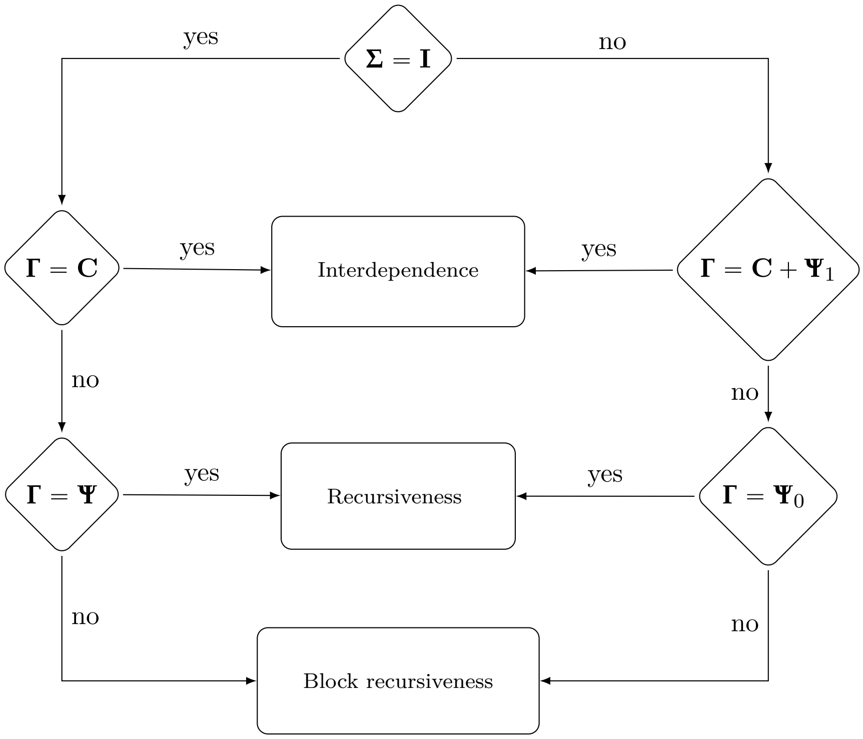

yes
$\mathbf{\Sigma} = \mathbf{I}$
no
$\mathbf{\Gamma} = \mathbf{C}$
yes
Interdependence
yes
$\mathbf{\Gamma} = \mathbf{C} + \mathbf{\Psi}_1$
no
no
$\mathbf{\Gamma} = \mathbf{\Psi}$
yes
Recursiveness
yes
$\mathbf{\Gamma} = \mathbf{\Psi}_0$
no
no
Block recursiveness

Figure 4: Flow-chart showing the possible outcome of the system decomposition in terms of $\mathbf{\Gamma}$ and $\mathbf{\Sigma}$.

Testing the Significance of Feedback Loops

In the previous section an analytic framework was set up to describe the potential feedbacks operating in a model. In fact, the analysis developed, relying on binary matrices, was meant to be qualitative since it only highlights the feedback set that potentially operates in a model, given the characteristics of its relations and its stochastic specification. Only once the model has been duly estimated, can the coefficients of matrix $\widetilde{\mathbf{C}}$ be properly evaluated. At this point, it proves useful to devise a procedure for testing the significance of the estimated loops (see Faliva and Zoia, 1994). To this end, let us observe that, once the matrix including all the feedbacks operating in the model

$$\mathbf{C} + \mathbf{\Psi}_1 = \mathbf{C} + \mathbf{\Psi}_1 = \mathbf{C} + (\mathbf{\Psi} * \mathbf{F}) \qquad\qquad \mathbf{F} = \mathbf{\Sigma}(\mathbf{I} - \mathbf{\Gamma})^{-1} \tag{26}$$

have been properly estimated, a test for the effective functioning of feedback loops can be established, based on the significance of its non-null entries. Any given equation, say the j-th one, turns out to be involved in feedback loops with other equations of the model whenever the $j-$th row of the above matrix is not a null vector. Should the (j,i)-th entry of this matrix be non-null, then a feedback between the j-th and the i-th equation would be expected to exist (see A.7 in the Appendix). Actually, it can be proved (see 2. in Appendix) that, in light of the identity

$$\mathbf{C} + (\mathbf{\Psi} * \mathbf{F}) = (\mathbf{C} + \mathbf{\Psi}) * \mathbf{F} = \mathbf{\Gamma} * \mathbf{F} \tag{27}$$

a test for the significance of the loops can be based on the exam of the statistical non-nullity of the elements of matrix $\mathbf{\Gamma} * \mathbf{F}$ which, unlike $\mathbb{C}$, does not require the preliminary split of $\mathbf{\Gamma}$ into its components, given the feedback loops $\mathbf{C} + \mathbf{\Psi}_1$ and causal links $\mathbf{\Psi}_0$.

In this context (following Faliva and Zoia, 1994), it can be proved that the j-th row of matrix $\mathbf{\Gamma} * \mathbf{F}$ measures both the direct effect of the RHS endogenous variables on the j-th one and the feedback effect of the latter on the former variables. In fact, the direct effects of the RHS endogenous variables, collected in vector $\mathbf{y}_o$, on variable y_j are included in the j-th row of matrix $\mathbf{\Gamma}$ (excluding its j-th element), that is

$$\frac{\partial E(y_j|\mathbf{y}_o)}{\partial \mathbf{y}_o} = \mathbf{e}_j'\mathbf{\Gamma}\mathbf{M}_j \tag{28}$$

Here, $\mathbf{e}_j$ is the L-dimensional j-th elementary vector and $\mathbf{M}_j$ is the $(L \times (L-1))$ selection matrix obtained from the identity matrix by deleting its j-th column, that is

$$\mathbf{M}_j = \left[\underset{(L,1)}{\mathbf{e}_1}, \ldots \underset{(L,1)}{\mathbf{e}_{j-1}}, \underset{(L,1)}{\mathbf{e}_{j+1}}, \ldots \underset{(L,1)}{\mathbf{e}_{L-1}} \right] \tag{29}$$

The feedback effects of the y_j variable on its explicative endogenous variables, $\mathbf{y}_o$, are included in the j-th row of matrix $\mathbf{F}$ (excluding its j-th element), that is

$$\frac{\partial E(\mathbf{y}_o'|y_j)}{\partial y_j} = (\mathbf{M}_j'\mathbf{F}'\mathbf{e}_j)' \tag{30}$$

To prove (30), let us focus on the j-th equation and consider this equation as the first of the system, with the others in sequence, that is

$$\underset{(1,1)}{y_j} = \underset{(1,L-1)}{\gamma_j'} \underset{(L-1,1)}{\mathbf{y}_o} + \underset{(1,J)}{\mathbf{a}_j'} \underset{(J,1)}{\mathbf{z}} + \underset{(1,1)}{\epsilon_j} \tag{31}$$

$$\underset{(L-1,1)}{\mathbf{y}_o} = \underset{(L-1,1)}{\boldsymbol{\eta}} \underset{(1,1)}{y_j} + \underset{(L-1,L-1)}{\mathbf{\Gamma}_o} \underset{(L-1,1)}{\mathbf{y}_o} + \underset{(L-1,J)}{\mathbf{A}_o} \underset{(J-1,1)}{\mathbf{z}} + \underset{(L-1,1)}{\boldsymbol{\epsilon}_o} \tag{32}$$

$$\begin{pmatrix} \epsilon_j \\ \boldsymbol{\epsilon}_o \end{pmatrix} \sim \mathcal{N}_L(\mathbf{0}, \mathbf{\Sigma}) \qquad \text{where} \qquad \mathbf{\Sigma} = \begin{pmatrix} \sigma_{jj} & \sigma_{jo} \\ \sigma_{oj} & \mathbf{\Sigma}_o \end{pmatrix} \tag{33}$$

Looking at the j-th equation, it is clear that vector $\gamma_j' = \mathbf{e}_j'\mathbf{\Gamma}\mathbf{M}_j$ measures the direct effect of the (RHS) endogenous variables on y_j. In order to determine the feedback effect of y_j on $\mathbf{y}_o$, let us rewrite (32) as follows

$$\mathbf{y}_o = \boldsymbol{\eta}(\gamma_j'\mathbf{y}_o + \mathbf{a}_j'\mathbf{z} + \epsilon_j) + \mathbf{\Gamma}_o\mathbf{y}_o + \mathbf{A}_o\mathbf{z} + \boldsymbol{\epsilon}_o \tag{34}$$

Next, given that, under normality, the following holds

$$\boldsymbol{\epsilon}_o = \frac{\boldsymbol{\sigma}_{oj}}{\sigma_{jj}}\epsilon_j + \boldsymbol{\zeta}_o; \qquad \boldsymbol{\zeta}_o \perp \epsilon_j \tag{35}$$

the set of equations (34) can be conveniently rewritten in the form

$$(\mathbf{I}-\mathbf{G})\mathbf{y}_o = \mathbf{D}\mathbf{z} + \mathbf{d}\epsilon_j + \boldsymbol{\zeta}_o \tag{36}$$

where

$$\mathbf{G} = \boldsymbol{\eta}\boldsymbol{\gamma}_j' + \boldsymbol{\Gamma}_o; \qquad \mathbf{D} = \boldsymbol{\eta}\mathbf{a}_j' + \mathbf{A}_o; \qquad \mathbf{d} = \boldsymbol{\eta} + \frac{\boldsymbol{\sigma}_{oj}}{\sigma_{jj}} \tag{37}$$

This, in turn (see 3. in Appendix) entails

$$\frac{\partial E(\mathbf{y}_o'|\epsilon_j)}{\partial y_j} = \frac{\partial E(\mathbf{y}_o'|\epsilon_j)}{\partial \epsilon_j}\frac{\partial \epsilon_j}{\partial y_j} = [(\mathbf{I}-\mathbf{G})^{-1}\mathbf{d}]' = \boldsymbol{\varphi}_j' = \frac{1}{\sigma_{jj}}\mathbf{e}_j'\mathbf{F}\mathbf{M}_j \tag{38}$$

Thus, we can conclude that the presence of non-zero elements in the vector

$$\boldsymbol{\rho}_j' = \boldsymbol{\gamma}_j' * \boldsymbol{\varphi}_j' = (\mathbf{e}_j'\boldsymbol{\Gamma}\mathbf{M}_j) * (\frac{1}{\sigma_{jj}}\mathbf{e}_j'\mathbf{F}\mathbf{M}_j) = \mathbf{e}_j'\frac{1}{\sigma_{jj}}(\boldsymbol{\Gamma} * \mathbf{F})\mathbf{M}_j \tag{39}$$

reveals the simultaneous action of both the direct effects of $\mathbf{y}_o$ on y_j and the feedback effects of y_j on $\mathbf{y}_o$.

Accordingly, testing the significance of $\boldsymbol{\rho}_j$ means checking whether the j-th endogenous is involved in feedback loops with other endogenous variables.

Actually, the statistic of the test can be derived from (39), by deleting from $\boldsymbol{\gamma}_j'$ the elements that, according to the exclusion constraints postulated by the economic theory, are null. This leads to move from the former $\boldsymbol{\rho}_j$ vector to the following compressed vector

$$\tilde{\boldsymbol{\rho}}_j = \tilde{\boldsymbol{\gamma}}_j' * \tilde{\boldsymbol{\varphi}}_j{}' = (\mathbf{S}_j\boldsymbol{\gamma}_j)' * (\mathbf{S}_j\boldsymbol{\varphi}_j)' \tag{40}$$

which has no zero entries. Here $\mathbf{S}_j$ is a selection matrix selecting from $\boldsymbol{\gamma}_j$ and $\boldsymbol{\varphi}_j$ the non-null entries. Accordingly, the reference model (31)-(33) can be restated as

$$y_j = \tilde{\boldsymbol{\gamma}}_j'\mathbf{y}_r + \tilde{\mathbf{a}}_j'\mathbf{z}_r + \epsilon_j \tag{41}$$

$$\mathbf{y}_r = \mathbf{K}\mathbf{z} + \tilde{\boldsymbol{\varphi}}_j\epsilon_j + \boldsymbol{\epsilon}_r \tag{42}$$

$$f_{(\epsilon_j,\boldsymbol{\epsilon}_r)} \sim \mathcal{N}_L\left(\mathbf{0}, \begin{pmatrix} \sigma_{jj} & \mathbf{0}' \\ \mathbf{0} & \boldsymbol{\Omega} \end{pmatrix}\right) \tag{43}$$

where

$$\mathbf{y}_r = \mathbf{S}_j\mathbf{y}_o, \quad \tilde{\mathbf{a}}_j = \mathbf{S}_r\mathbf{a}_j, \quad \mathbf{z}_r = \mathbf{S}_r\mathbf{z} \tag{44}$$

and $\mathbf{S}_r$ is the matrix selecting the non-null entries from $\mathbf{a}_j'$ and the sub-set of predetermined variables playing an explicative role in the j-th equation. Furthermore,

$$\mathbf{K} = \mathbf{S}_j(\mathbf{I}-\mathbf{G})^{-1}\mathbf{D}, \qquad \boldsymbol{\epsilon}_r = \mathbf{S}_j(\mathbf{I}-\mathbf{G})^{-1}\boldsymbol{\zeta}_o, \qquad \boldsymbol{\Omega} = E(\boldsymbol{\epsilon}_r\boldsymbol{\epsilon}_r') \tag{45}$$

Hence, the issue of the nature, unidirectional rather than bidirectional, of the equation at stake can be unfolded by testing a hypothesis in the given form

$$\begin{cases} H_0 : \tilde{\boldsymbol{\rho}}_j = \mathbf{0} \\ H_1 : \tilde{\boldsymbol{\rho}}_j \neq \mathbf{0} \end{cases} \tag{46}$$

The Wald test takes the form

$$W = \hat{\tilde{\boldsymbol{\rho}}}_j{}'(\hat{\mathbf{J}}\hat{\boldsymbol{\Psi}}^{-1}\hat{\mathbf{J}}')^{-1}\hat{\tilde{\boldsymbol{\rho}}}_j \tag{47}$$

where $\hat{\tilde{\boldsymbol{\rho}}}_j$ is the maximum likelihood estimate of $\tilde{\boldsymbol{\rho}}_j$ (see 4. in Appendix), and $\hat{\mathbf{J}}$, $\hat{\boldsymbol{\Psi}}$ are, respectively, the Jacobian matrix

$$\hat{\mathbf{J}} = \left.\frac{\partial\tilde{\boldsymbol{\rho}}_j(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \tag{48}$$

and the information matrix

$$\hat{\mathbf{\Psi}} = \left.\frac{\partial^2 l(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}'}\right|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \tag{49}$$

evaluated in correspondence of the maximum likelihood estimate of the parameter vector

$$\boldsymbol{\theta}' = [\tilde{\boldsymbol{\gamma}}_j', \tilde{\mathbf{a}}_j', \tilde{\boldsymbol{\varphi}}_j', \text{vec}(\mathbf{K}), \sigma_{jj}, \text{vech}(\mathbf{\Omega})]' \tag{50}$$

Under the null hypothesis

$$W \underset{as.}{\approx} \chi^2_k \tag{51}$$

where k is the dimension of $\tilde{\boldsymbol{\rho}}_j$.

If the Wald test provides evidence that the j-th equation is involved in a statistically significant feedback loop with other equations of the model, it is worth singling out the variables that are primarily responsible for the feedback at hand. They can be identified by checking the significance of each non-null element of $\hat{\tilde{\boldsymbol{\rho}}}_j$. Under the null that the i-th element of $\hat{\tilde{\boldsymbol{\rho}}}_j$ is non-zero, the Wald statistic, for testing the significance of the loop bridging the i-th and j-th endogenous, turns out to be

$$W = (\mathbf{e}_i'\hat{\tilde{\boldsymbol{\rho}}}_j)'[\mathbf{e}_i(\hat{\mathbf{J}}\hat{\mathbf{\Psi}}^{-1}\hat{\mathbf{J}}')^{-1}\mathbf{e}_i'](\hat{\tilde{\boldsymbol{\rho}}}_j\mathbf{e}_i) \underset{as.}{\approx} \chi^2_1 \tag{52}$$

Detecting and testing causal and interdependent links in a model with SIRE

Investigating potential feedbacks with SIRE

The analysis developed in the previous sections allows the identification of the potential feedbacks operating in a model. By assuming the stochastic specification of the model as known, the investigation can be carried out by using binary matrices $\mathbf{\Gamma}^b$ and $\mathbf{\Sigma}^b$ without a preliminary estimation of the model. The causal structure, which emerges from this analysis, is implied by the theory underlying the model and mirrored by the topological properties of matrices $\mathbf{\Gamma}$ and $\mathbf{\Sigma}$. It is also important to point out that the feedback loops thus detected are only potential, because their effectiveness must find confirmation in empirical evidence. We start by loading the **SIRE** package.

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

The function `causal_decompose()` is devised for decomposing the matrix $\mathbf{\Gamma}$. If the structure of $\mathbf{\Sigma}$ is assumed as known by the user, the function takes the following arguments:

- `data`: *not appropriate to simulated context, set to* `NULL`.
- `eq.system`: the system of equations.
- `resid.est`: *not appropriate to simulated context, set to* `NULL`.
- `instruments`: *not appropriate to simulated context, set to* `NULL`.
- `sigma.in`: the binary matrix $\mathbf{\Sigma}^b$.

and provides the following output:

- `eq.system`: the system of equations given as input.
- `gamma`: the binary matrix $\mathbf{\Gamma}^b$.
- `sigma`: the binary matrix $\mathbf{\Sigma}^b$ given as input.
- `C`: the binary matrix of the coefficients associated to the endogenous variables involved in interdependent mechanisms operating at a systematic level.
- `Psi1`: the binary matrix of the coefficients associated to the endogenous variables involved in interdependent mechanisms induced by error correlation (if `Sigma` is not diagonal).
- `Psi0`: the binary matrix of the coefficients associated to the endogenous variables having a causal role.
- `all.graph`: the DAG object for the undecomposed path diagram (via the R package **igraph**; Amestoy, 2017).
- `dec.graph`: the DAG object for the decomposed path diagram.

Furthermore, if the error terms are assumed to be spherical, then the **SIRE** package simply splits $\mathbf{\Gamma}$ in two sub-matrices $\mathbf{C}^b$ and $\mathbf{\Psi}^b$, reflecting the interdependent and causal relations operating in the system at a deterministic level.

With regard to the system (13), the corresponding code is

```
> eq.system <- list(
+              eq1 = y1 ~ y5 + y7, eq2 = y2 ~ z,
+              eq3 = y3 ~ y11, eq4 = y4 ~ y3,
+              eq5 = y5 ~ y10, eq6 = y6 ~ y5 + y9,
+              eq7 = y7 ~ y6, eq8 = y8 ~ y12,
+              eq9 = y9 ~ y7, eq10 = y10 ~ y5,
+              eq11 = y11 ~ y12, eq12 = y12 ~ y4 + y11,
+              eq13 = y13 ~ y2 + y6)
> #fictitious Sigma matrix
> Sigma <- diag(length(eq.system))
> #function call
> decompose.A <- causal_decompose(eq.system , sigma.in = Sigma)
```

The output is comprised of matrices $\mathbf{C}^b$ and $\mathbf{\Psi}^b$ given in (15). The graphical representation of the system, given in Figure 2, is obtained with the `tkplot()` function of the R package **igraph**

```
> tkplot(decompose.A$dec.graph)
```

The following example refers to a matrix $\mathbf{\Sigma}^b$ specified as in (23)

```
> # indexes of non-null elements of Sigma
> sigma.idx <- cbind(
+  rbind(rep(1,5),c(4,5,8,10,12)),   #y1
+  rbind(rep(2,4),c(4,6,8,9)),       #y2
+  rbind(rep(3,4),c(6,7,11,13)),     #y3
+  rbind(rep(4,6),c(5,6,8,9,10,12)), #y4
+  rbind(rep(5,3),c(8,10,12)),       #y5
+  rbind(rep(6,5),c(7,8,9,11,13)),   #y6
+  rbind(rep(7,2),c(11,13)),         #y7
+  rbind(rep(8,3),c(9,10,12)),       #y8
+  rbind(rep(10,1),c(12)),           #y10
+  rbind(rep(11,1),c(13)))           #y11
> # fictitious Sigma matrix
> low.tri <- as.matrix(Matrix::sparseMatrix(i = sigma.idx[2,] , j = sigma.idx[1,], x = 1,
+                                           dims = rep(length(eq.system),2)))
> Sigma <- low.tri + t(low.tri) + diag(length(eq.system))
> # function call
> decompose.B <- causal_decompose(eq.system = eq.system,
+                              sigma.in = Sigma)
```

In this case, the package provides as output matrix $\mathbf{C}^b$ and splits matrix $\mathbf{\Psi}^b$ into sub-matrices $\mathbf{\Psi}_1^b$ and $\mathbf{\Psi}_0^b$, as in (24) and (25). The `tkplot()` function can still be used to obtain the pictures of the relations among the variables given in Figure 3.

The next section will show how to perform the decomposition with `causal_decompose()` if the structure of $\mathbf{\Sigma}$ is not known and the goal is to carry out estimation and feedback testing from observed data.

Finding significant feedbacks with SIRE: an application to Italian macroeconomic data

As pointed out in the previous section, empirical evidence aside, the results of a decomposition based on binary matrices $\mathbf{\Gamma}^b$ and $\mathbf{\Sigma}^b$ must be considered as preliminary since they show only the potential links acting in the system. The effectiveness of these links demands a confirmation based on a sound empirical-evidence argument. In fact, the lack of significance of one or more of the feedbacks thus detected can alter the nature of the connections among the endogenous variables found by the preliminary decomposition, which is based only on the topological properties of matrices $\mathbf{\Gamma}$ and $\mathbf{\Sigma}$. In order to show how effective feedbacks operating in a model can be detected and tested, we have applied the functionalities of **SIRE** to the Klein model (see Klein, 1950, and Greene, 2003). This model, originally conceived for the US economy, has been recast for the Italian economy. The Italian macroeconomic variables, mirroring the US counterparts, are available at http://dati.istat.it/. The given model is composed of $n = 60$ observations on a quarterly basis and six equations explaining the following endogenous variables: consumption expenses for Italian families [C], added value [CP], private wages from dependent employment [WP], gross investment [I], gross capital stock [K], gross domestic product [GDP]. The model is specified as follows

$$\begin{bmatrix} \mathrm{C}_t \\ \mathrm{I}_t \\ \mathrm{WP}_t \\ \mathrm{GDP}_t \\ \mathrm{CP}_t \\ \mathrm{K}_t \end{bmatrix} = \mathbf{a}_0 + \begin{bmatrix} 0 & \gamma_{12} & 0 & 0 & \gamma_{15} & 0 \\ 0 & 0 & 0 & 0 & 0 & \gamma_{26} \\ 0 & \gamma_{32} & 0 & \gamma_{34} & 0 & 0 \\ \gamma_{41} & \gamma_{42} & 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma_{53} & 0 & 0 & 0 \\ 0 & \gamma_{62} & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{C}_t \\ \mathrm{I}_t \\ \mathrm{WP}_t \\ \mathrm{GDP}_t \\ \mathrm{CP}_t \\ \mathrm{K}_t \end{bmatrix} + \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & 0 & 0 & 0 \\ 0 & 0 & a_{34} & 0 \\ 0 & 0 & a_{44} & 0 \\ 0 & 0 & 0 & a_{55} \\ 0 & a_{62} & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{CP}_{t-1} \\ \mathrm{K}_{t-1} \\ \mathrm{GDP}_{t-1} \\ \mathrm{T}_t \end{bmatrix} + \begin{bmatrix} e_{\mathrm{C}} \\ e_{\mathrm{I}} \\ e_{\mathrm{WP}} \\ e_{\mathrm{GDP}} \\ e_{\mathrm{CP}} \\ e_{\mathrm{K}} \end{bmatrix} \tag{53}$$

where $\mathbf{a}_0$ is the intercept vector. As equation (53) shows, the set of predetermined variables includes one exogenous variable, taxes [T_t], and three lagged endogenous variables, that is: the one-lagged added value [CP_{t-1}], the one-lagged gross capital stock [K_{t-1}], and the one-lagged gross domestic product [GDP_{t-1}]. We first load the data into the R workspace.

```
> data(macroIT)
```

Following Greene, the model equations have been estimated with 3SLS by using the R package **systemfit** (Henningsen and Hamann, 2017). The one-lagged capital stock [K_{t-1}], [T_t], [CP_{t-1}], and [GDP_{t-1}] have been employed as instrumental variables. Matrix $\mathbf{\Sigma}$, if the user does not specify its structure, is estimated by using the covariance matrix of the structural residuals. The function `causal_decompose()` can be also employed to estimate both the model via 3SLS and the $\mathbf{\Sigma}$ matrix, and yields three matrices: $\mathbf{C}$, $\mathbf{\Psi}_1$, and $\mathbf{\Psi}_0$. The first two include the coefficients associated to variables affected by feedback loops, operating either at a deterministic level or induced by error terms, the third contains the coefficients associated to variables playing a causal role in the system. This version of `causal_decompose()` takes the following arguments:

- `data`: data frame containing all the variables in the equations.
- `eq.system`: list containing all the equations, as in **systemfit**.
- `resid.est`: denotes the method used to estimate $\mathbf{\Sigma}$, on the basis of 3SLS residuals; this method is specified in **systemfit**.
- `instruments`: set of instruments used to estimate the model, introduced either as a list or as a character vector, as in **systemfit**.
- `sigma.in`: *not appropriate to empirical context, set to* `NULL`.

The output of this function is a list containing the following objects:

- `eq.system`: the same list of equations provided as input.
- `gamma`, `C`, `Psi0`, `Psi1`, `A`, and `Sigma`: respectively matrices $\mathbf{C}$, $\mathbf{\Psi}_0$, $\mathbf{\Psi}_1$, $\mathbf{A}$, and $\mathbf{\Sigma}$.
- `systemfit`: the output of the `systemfit()` function used to estimate the model.
- `all.graph`: the DAG object for the undecomposed path diagram.
- `dec.graph`: the DAG object for the decomposed path diagram.
- `path`: the data-set containing all the paths/relations among the endogenous variables, along with their classification (i.e., causal, interdependent). The graph highlights which interdependent relations work at a systematic level and which are induced by the effect of correlations among residuals).

The code below performs the decomposition using the `macroIT` data

```
> #system of equations
> eq.system <- list(eq1 <- C ~  CP  + I +  CP_1  ,
+                   eq2 <- I ~ K + CP_1,
+                   eq3 <- WP ~  I + GDP +  GDP_1,
+                   eq4 <- GDP ~ C + I + GDP_1,
+                   eq5 <- CP ~   WP  + T,
+                   eq6 <- K ~ I + K_1)
> #instruments
> instruments <- ~ T +  CP_1 + GDP_1 + K_1
> #decomposition
> dec.macroIT <- causal_decompose(data = macroIT,
+                                 eq.system = eq.system,
+                                 resid.est = "noDfCor",
+                                 instruments = instruments)
```

Table 1 shows the results of the model estimation. Since some coefficients are not statistically significant (such as the coefficient associated to [I] in the equation explaining [C] and the coefficient associated to [GDP] in the equation explaining [WP]), the model has been re-estimated and the coefficient matrix associated to the explicative endogenous variables decomposed again.

```
> #system of equations
> eq.system <- list(eq1 <- C ~ CP + CP_1  ,
+                   eq2 <- I ~ K,
+                   eq3 <- WP ~  I +  GDP_1,
+                   eq4 <- GDP ~ C + I + GDP_1,
+                   eq5 <- CP ~   WP  + T,
+                   eq6 <- K  ~ I + K_1)
> #instruments
> instruments <- ~ T +  CP_1 + GDP_1 + K_1
> #decomposition
> dec.macroIT.new <- causal_decompose(data = macroIT,
+                                     eq.system = eq.system,
+                                     resid.est = "noDfCor",
+                                     instruments = instruments)
```

The results of the last estimation process are shown in Table 2. Looking at the Theil inequality indexes (Theil, 1961) reported in the last column of the table, we can see that the estimated equations fit the data very well. In fact, all Theil indexes are close to zero. The estimated covariance matrix of the structural error terms is given by

$$\hat{\mathbf{\Sigma}} = \begin{bmatrix} 10.93 & & & & & \\ -2.51 & 2.61 & & & & \\ 10.75 & -5.04 & 52.31 & & & \\ -7.55 & 1.55 & 3.66 & 7.15 & & \\ -9.6 & 4.27 & -19.73 & 6.07 & 15.08 & \\ 0.43 & -0.68 & 0.53 & -0.09 & -0.68 & 0.81 \end{bmatrix} \tag{54}$$

while matrices $\mathbf{C} + \mathbf{\Psi}_1$ and $\mathbf{\Psi}_0$ turn out to be

$$\begin{aligned} \mathbf{C} + \mathbf{\Psi}_1 &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.73 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.67 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 1.02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.65 & 0 & 0 & 0 & 0 \\ 1.09 & 0.39 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.48 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 & 0 & 1.02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.73 \\ 0 & -1.65 & 0 & 0 & 0 & 0 \\ 1.09 & 0.39 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.48 & 0 & 0 & 0 \\ 0 & 0.67 & 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{55}$$

$$\mathbf{\Psi}_0 = \mathbf{0} \tag{56}$$

The matrix in Equation (55) embodies all the coefficients associated to variables involved in feedback loops, while matrix (56) includes those associated to variables playing a causal role. Looking at (55) we find a direct feedback between variables [I] and [K], while the variables of the pairs [I, WP], [I, GDP], [C, GDP], [CP, C], and [CP, WP] are directly linked (a black arrow connects the variables of each pair) as well as explained by equations with correlated errors. Accordingly, the variables of each pair may be internally connected by feedback loops. The goal of our testing procedure will be to bring out which of these feedbacks, being significant, are truly effective. Figure 5 depicts the links operating in this model, using the function `tkplot()` of the **igraph** package. In this figure, a unidirectional arrow denotes that a variable is explicative for another. If two variables are explicative one for the other, a direct feedback loop exists, depicted as two red arrows going in opposite directions. Instead, a red, dashed, curved, two-headed arrow between two variables indicates the existence of a feedback induced by error correlation.

```
> tkplot(dec.macroIT.new$dec.graph)
```

Testing for feedback effects

The significance of these loops has been investigated by using the function `feedback_ml()` which performs the Wald test given in (52). The 3SLS parameter estimates have been used as preliminary estimates to obtain the maximum likelihood (ML) estimates of the parameters needed to build the test statistic. In particular, in order to reach the global maximum of the log-likelihood, the initial 3SLS parameter estimates have been randomly perturbed a certain number of times. The optimizer chosen for the scope is included in the **Rsolnp** package where the function `gosolnp` is specially designed for the randomization of starting values. The function `feedback_ml()` takes the following arguments:

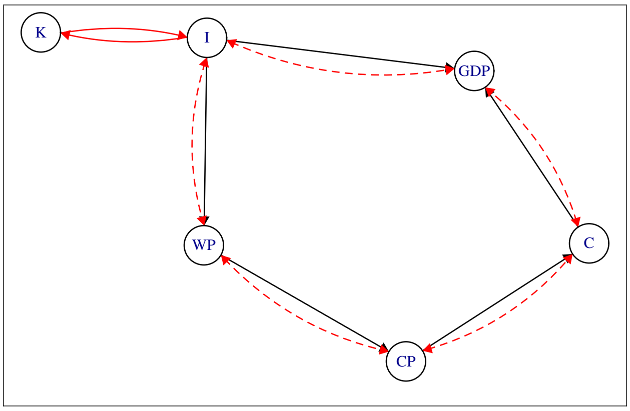


Figure 5: Path diagram of the macroeconomic model. Unidirectional arrows denote that one variable is explicative for another. The two red unidirectional arrows denote the presence of a direct feedback. The red, dashed, curved, double-headed arrows between pairs of variables denote feedback loops induced by error correlation.

	$\mathbf{a}_0$	C_t	I_t	WP_t	GDP_t	CP_t	K_t	T_t	GDP_{t-1}	CP_{t-1}	K_{t-1}	Theil
C_t	5.50 (11.97)	-	-0.1 (0.19)	-	-	10.1 *** (0.09)	-	-	-	-0.35*** (0.099)	-	0.0073
I_t	10.28 (7.627)	-	-	-	-	-	0.72 *** (0.089)	-	-	0.005 (0.04)	-	0.0115
WP_t	-320.17*** (28.87)	-	-1.93*** (0.453)	-	0.46 (0.277)	-	-	-	1.04 *** (0.251)	-	-	0.0288
GDP_t	4.55 (10.732)	1.02 *** (0.098)	0.37 * (0.181)	-	-	-	-	-	0.31 *** (0.075)	-	-	0.004
CP_t	144.49*** (10.848)	-	-	0.49 *** (0.045)	-	-	-	3.62 *** (0.373)	-	-	-	0.0074
K_t	-5.69*** (1.599)	-	0.68 *** (0.073)	-	-	-	-	-	-	-	0.49 *** (0.053)	0.0063

Table 1: Macroeconomic model: preliminary estimates with 3SLS. The last column shows the Theil index for each model equation.
●: significant at level α=0.1
*: significant at level α=0.05
**: significant at level α=0.01
***: significant at level α=0.001

	$\mathbf{a}_0$	C_t	I_t	WP_t	GDP_t	CP_t	K_t	T_t	GDP_{t-1}	CP_{t-1}	K_{t-1}	Theil
C_t	10.06 (9.17)	-	-	-	-	1.02 *** (0.115)	-	-	-	-0.39*** (0.111)	-	0.0076
I_t	11.22*** (2.202)	-	-	-	-	-	-	-	-	0.73 ** (0.027)	-	0.0114
WP_t	-299.12*** (26.387)	-	-1.65*** (0.444)	-	-	-	-	-	1.39 *** (0.134)	-	-	0.0304
GDP_t	3.04 (8.89)	1.09 *** (0.118)	0.39 ** (0.112)	-	-	-	-	-	0.28 *** (0.076)	-	-	0.0042

- `data`: data frame containing all the variables in the equations.
- `out.decompose`: the output from the previous causal decomposition which is called by using the command `causal_decompose()`.
- `lb` and `ub`: upper and lower bound of the parameter space (as in `gosolnp`).
- `nrestarts`, `nsim` and `seed.in`: parameters tuning the number of random initializations (as in `gosolnp`).

The output of this function is a list containing the following objects:

- `rho.est`: a data frame containing the estimated feedback loops for a given equation. The first column of this data frame, `feedback eqn.`, provides the indexes of the equations involved in the feedback loop with the equation given in input, while the coefficients associated to the explicative endogenous for the equation in question are shown in the column `rho.est`.
- `loglik`: the estimated log-likelihood of the best model.
- `theta.hessian`: the estimated Hessian matrix $\hat{\mathbf{I}}$.
- `rho.jacobian`: the estimated Jacobian matrix $\hat{\mathbf{J}}$.
- `wald`: the value of the Wald test statistic W.

As an example, let us assume that the interest is in testing the significance of the feedbacks affecting the second equation, explaining the endogenous variable [I]. According to the previous analysis, this variable is connected to [K] by a bidirectional link.

The Wald test for the significance of this feedback is performed by using the function `feedback_ml()` specified as follows

```
> test.E2=feedback_ml(data = macroIT,
+                     out.decompose = dec.macroIT.new,
+                     eq.id = 2,
+                     lb = min(dec.macroIT.new$Sigma) - 10,
+                     ub = max(dec.macroIT.new$Sigma) + 10,
+                     nrestarts = 10,
+                     nsim = 20000,
+                     seed.in = 1)
```

By visualizing the estimate of $\boldsymbol{\rho}$ and the Wald statistic

```
> test.E2$rho.tbl
  Feedback eqn.   rho.est
1             6 0.1641469

> test.E2$wald
         [,1]
[1,] 4.115221
```

we can see that the existence of a feedback loop between [I] and [K] is confirmed.

Table 3 shows the results of the test for all the equations of the model. Looking at the p-values we conclude that all feedbacks are significant except the ones involving [CP] and [GDP]. For what concerns [CP], it is explained by [WP] without a feedback effect from the latter to the former. Regarding [GDP], which is affected by feedback effects, a deeper analysis is required in order to understand which of its two explicative variables [C] and [I] (if not both) are responsible for it. To this end, we have applied the Wald statistic given in (52) which leads us to conclude that only [C] is involved in a feedback loop with [GDP]. In the end, the path diagram fully describing the recurrent and interdependent relationships in the model is displayed in Figure 6.

Discussion

The set of functions worked out in the paper allows a system of simultaneous equations to be split into recursive and/or interdependent subsystems. The user can rely on `causal_decompose()` in two ways: to assess the presence of interdependent relations with a known structure of correlation among the error terms, or to estimate the whole model in presence of empirical data.

The significance of the feedback loops operating in the model is tested with a Wald test using the `feedback_ml()` function. The 3SLS parameter estimates are used as preliminary estimates to obtain the maximum likelihood ones, which are needed to build the test.

Equation	Feedback Variable	Joint W	p-value	Singular W	p-value
C	CP	386.6	$< .001$		
I	K	4.115	0.042	-	-
WP	I	25.55	$< .001$	-	-
GDP	C	95.368	< 0.0001	84.315	< 0.0001
	I			0.352	0.553
CP	WP	0.046	0.831	-	-
K	I	19.595	<0.0001	-	-

Table 3: Macroeconomic model: tests for feedback effects for the final model. Joint W denotes the Wald statistic used to test the set of feedback loops affecting a given variable (see (47). Singular W denotes the Wald statistic used to test the feedback effect between two specific variables (see (52)).

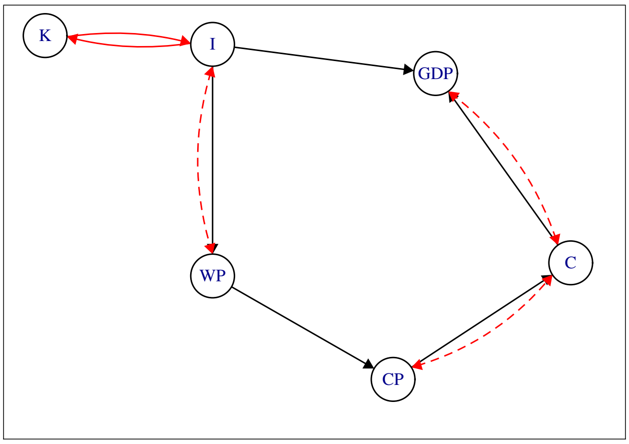


Figure 6: Path diagram of the modified macroeconomic model after testing for feedback effects. Black arrows denote causal link ($\mathbf{\Upsilon}_0$), red arrows denote interdependent links ($\mathbf{C}$), black arrows and red dashed arrows denote interdependent links induced by the correlation of the error terms ($\mathbf{\Upsilon}_1$).

As for the rationale of our procedure, which rests on a properly devised test, it is worth taking into account the considerable concern raised recently in the statistical community about the use of significance testing (see Wasserstein and Lazar, 2016). In this connection, in order to avoid improper use of p-values and significance-related results, it may be worth addressing the issue of detecting feedback mechanisms in a simultaneous equations model with different approaches. Among them, the construction of confidence intervals and the employment of Bayesian methods look particularly promising for future investigation.

Moving now on more technical notes:

- The ML estimation is performed by concentrating the likelihood with respect to the 3SLS estimates of $\mathbf{A}$ in Equation (1), to reduce the computation required to otherwise re-estimate parameters that are unnecessary for the computation of the feedback effect.
- As far as the error covariance matrix $\mathbf{\Sigma}$ is concerned, in the current formulation of the test its estimate $\hat{\mathbf{\Sigma}}$ is not involved by itself in any testing sub-routine (in fact, all of its elements are

retained after the 3SLS step), and computing the related matrix of standard errors is therefore of secondary importance. However, if a matrix normal distribution is hypothesized on $\mathbf{E}$, then the distribution of $\hat{\boldsymbol{\Sigma}}$ turns out to be a L-dimensional Wishart with T degrees of freedom and scale matrix $\boldsymbol{\Sigma}$. Thus, the variance of its elements can be calculated straightforwardly (see Gupta and Nagar, 1999).

Bibliography

P. R. Amestoy. *Igraph: Network Analysis and Visualization*, 2017. URL https://CRAN.R-project.org/package=igraph. R package version 1.1.2. [p158]

E. Bellino, S. Nerozzi, and M. G. Zoia. Introduction to luigi pasinetti's 'causality and interdependence . . . '. *Structural Change and Economic Dynamics*, 2018. ISSN 0954-349X. URL https://doi.org/10.1016/j.strueco.2018.09.007. [p149]

M. Faliva. Recursiveness vs. interdependence in econometric models: A comprehensive analysis for the linear case. *Journal of the Italian Statistical Society*, 1(3):335–357, 1992. [p149, 151, 154]

M. Faliva and M. G. Zoia. Detecting and testing causality in linear econometric models. *Journal of the Italian Statistical Society*, 3(1):61–76, 1994. [p149, 156]

M. Fiedler. *Special Matrices and Their Applications in Numerical Mathematics: Second Edition*. Dover Books on Mathematics. Dover Publications, 2013. [p152]

M. Gilli. Causal ordering and beyond. *International Economic Review*, pages 957–971, 1992. URL https://doi.org/10.2307/2527152. [p149]

C. W. Granger. Testing for causality: a personal viewpoint. *Journal of Economic Dynamics and control*, 2:329–352, 1980. URL https://doi.org/10.1016/0165-1889(80)90069-x. [p149]

W. H. Greene. *Econometric Analysis*. Pearson Education, 2003. [p159, 160]

A. K. Gupta and D. K. Nagar. *Matrix Variate Distributions*. Monographs and Surveys in Pure and Applied Mathematics. Taylor & Francis, 1999. URL https://doi.org/10.1201/9780203749289. [p166]

A. Henningsen and J. D. Hamann. *Systemfit: Estimating Systems of Simultaneous Equations*, 2017. URL https://CRAN.R-project.org/package=systemfit. R package version 1.1-20. [p160]

K. G. Jöreskog. Structural analysis of covariance and correlation matrices. *Psychometrika*, 43(4): 443–477, 1978. [p149]

K. G. Jöreskog and H. O. A. Wold. *Systems Under Indirect Observation: Causality, Structure, Prediction*. Number 139, pt. 2 in Contributions to Economic Analysis. North-Holland Publishing Company, 1982. [p152]

L. R. Klein. *Economic Fluctuations in the United States, 1921-1941*. Monographs of the Cowles Commission for Research in Economics. John Wiley & Sons, 1950. [p159]

R. B. Marimont. System connectivity and matrix properties. *The Bulletin of Mathematical Biophysics*, 31(2):255–274, 1969. [p151]

J. Ponstein. *Matrices in Graph and Network Theory*. Van Gorcum's natuurwetenschappelijke reeks. Van Gorcum & Comp., 1966. [p152]

R. H. Strotz and H. O. A. Wold. Recursive vs. nonrecursive systems: An attempt at synthesis (part I of a triptych on causal chain systems). *Econometrica*, 28(2):417–427, 1960. [p149, 150]

H. Theil. *Economic Forecasts and Policy*. Contributions to economic analysis. North-Holland Publishing Company, 1961. [p161]

R. L. Wasserstein and N. A. Lazar. The ASA's statement on p-values: Context, process, and purpose. *The American Statistician*, 70(2):129–133, 2016. URL https://doi.org/10.1080/00031305.2016.1154108. [p165]

H. O. A. Wold. *Econometric Model Building: Essays on the Causal Chain Approach*. Contributions to economic analysis. North-Holland Publishing Company, 1964. [p149, 150]

Proof of relevant formulas

In this Appendix we provide the proofs of some relevant formulas of the paper.

1. Let $\mathbf{\Sigma}$ and $\mathbf{R}$ be defined as in Section 2. Then, the proof that $\mathbf{\Sigma}^b(\mathbf{I}+\mathbf{R})$ is the binary matrix associated to $\mathbf{\Sigma}\,(\mathbf{I}-\mathbf{\Gamma})^{-1}$ is based on the following two theorems.

   Theorem 1. *If two conformable matrices,* $\mathbf{A}$ *and* $\mathbf{B}$*, are such that*

   $$\mathbf{A} = \mathbf{A} * \mathbf{H} \qquad \mathbf{B} = \mathbf{B} * \mathbf{K} \tag{A.1}$$

   then the binary matrix associated to $\mathbf{AB}$ *is* $(\mathbf{HK})^b$*.* ■

   Theorem 2. *If a non-singular matrix* $\mathbf{A}$ *is such that*

   $$\mathbf{A} = \mathbf{A} * \mathbf{H} \tag{A.2}$$

   where $\mathbf{H}$ *is a given binary matrix, then*

   $$(\mathbf{A}^{-1}) * (\mathbf{I} + \sum_{n=1}^{N-1} \mathbf{H}^n)^b = (\mathbf{A}^{-1}) \tag{A.3}$$

   where N is the matrix dimension. ■

   Now, upon noting that

   $$(\mathbf{I}-\mathbf{\Gamma}) = (\mathbf{I}-\mathbf{\Gamma}) * (\mathbf{I}-\mathbf{\Gamma}^b), \tag{A.4}$$

   reference to Theorem 2 leads to conclude that

   $$(\mathbf{I}-\mathbf{\Gamma})^{\mathbf{-1}} = (\mathbf{I}-\mathbf{\Gamma})^{\mathbf{-1}} * (\mathbf{I}+\mathbf{R}) \tag{A.5}$$

   Next, taking into account that $\mathbf{\Sigma}^b$ and $(\mathbf{I}+\mathbf{R})$ are the binary counterparts of the $\mathbf{\Sigma}$ matrix and $(\mathbf{I}-\mathbf{\Gamma})^{\mathbf{-1}}$ reference to Theorem 1 entails the following

   $$\mathbf{\Sigma}(\mathbf{I}-\mathbf{\Gamma}) = [\mathbf{\Sigma}(\mathbf{I}-\mathbf{\Gamma})] * [\mathbf{\Sigma}^b(\mathbf{I}+\mathbf{R})]. \tag{A.6}$$

2. The proof that $\mathbf{C}$ and $\mathbf{F}$, defined as in Section 3, satisfy the following relationship

   $$\mathbf{C} * \mathbf{F} = \mathbf{C} \tag{A.7}$$

   hinges on a preliminary result given in the following theorem.

   Theorem 3. *The matrices* $\mathbf{C}$ *and* $\mathbf{I}+\mathbf{R}$ *satisfy the following relationship*

   $$\mathbf{C}^b * (\mathbf{I}+\mathbf{R}) = \mathbf{C}^b \tag{A.8}$$

   Proof

   Taking into account that the Hadamard product is both commutative ($\mathbf{A} * \mathbf{B} = \mathbf{B} * \mathbf{A}$) and idempotent for binary matrices ($\mathbf{A}^b * \mathbf{A}^b = \mathbf{A}^b$), and being $\mathbf{\Gamma}$ hollow, the following holds

   $$\mathbf{\Gamma}^b * \mathbf{I} = \mathbf{0}, \tag{A.9}$$

   simple computations yield

   $$\mathbf{C}^b * (\mathbf{I}+\mathbf{R}) = \mathbf{\Gamma}^b * \mathbf{R} * (\mathbf{I}+\mathbf{R}) = \mathbf{\Gamma}^b * \mathbf{R} * \mathbf{I} + \mathbf{\Gamma}^b * \mathbf{R} * \mathbf{R} = \mathbf{C}^b \tag{A.10}$$

   ■

   Now, consider the following theorem (where the symbol $\mathbf{A} \geqslant \mathbf{0}$ denotes that all the elements of matrix $\mathbf{A}$ are non negative numbers):

   Theorem 4. *Let* $\mathbf{B} \geqslant \mathbf{0}$ *and* $\mathbf{A}^b * \mathbf{B}^b = \mathbf{A}^b$*. If* $\mathbf{C} \geqslant \mathbf{0}$*, then*

   $$\mathbf{A}^b * (\mathbf{B}+\mathbf{C})^b = \mathbf{A}^b \tag{A.11}$$

   ■

   Given this premise, we can now prove (A.7). To this end, let us write $\mathbf{\Sigma}^b$ as follows

   $$(\mathbf{I}+\mathbf{\Delta}) = \mathbf{\Sigma}^b \tag{A.12}$$

   where $\mathbf{\Delta} * \mathbf{I} = \mathbf{0}$ is a hollow matrix, and note that, in light of (A.12) and (A.5), the binary matrix associated to $\mathbf{F}$ is, according to Theorem 1, given by

   $$\mathbf{F}^b = [(\mathbf{I}+\mathbf{\Delta})(\mathbf{I}+\mathbf{R})]^b \tag{A.13}$$

Next, use of Theorems 3 and 4, yields the following

$$\mathbf{C}^b * \mathbf{F}^b = \mathbf{C}^b * [(\mathbf{I}+\boldsymbol{\Delta})(\mathbf{I}+\mathbf{R})]^b = \mathbf{C}^b * [(\mathbf{I}+\mathbf{R}) + \boldsymbol{\Delta}(\mathbf{I}+\mathbf{R})]^b = \mathbf{C}^b \tag{A.14}$$

as $(\boldsymbol{\Delta}(\mathbf{I}+\mathbf{R}))^b \geqslant \mathbf{0}$. This, in turn, entails that

$$\mathbf{C}^b + \boldsymbol{\Psi}_1^b * \mathbf{F}^b = (\mathbf{C}^b + \boldsymbol{\Psi}_1^b) * \mathbf{F}^b = \boldsymbol{\Gamma}^b * \mathbf{F}^b \tag{A.15}$$

which means that $\mathbf{C} + \boldsymbol{\Psi}_1 * \mathbf{F}$ and $\boldsymbol{\Gamma} * \mathbf{F}$ have the same topological structure. ■

3. **Proof of** (38) . Formula (38) can be proved as follows. First, note that matrix $\boldsymbol{\Gamma}^*$ weighting the current endogenous explicative variables in the model (31), (32) can be expressed as

$$\boldsymbol{\Gamma}^* = \mathbf{P}_j \boldsymbol{\Gamma} \mathbf{P}_j \tag{A.16}$$

where $\boldsymbol{P}_j$ is a permutation matrix obtained from an identity matrix by interchanging its first row with its j-th row. Then note that

$$\boldsymbol{\Gamma}^* = \begin{bmatrix} 0 & \gamma_j' \\ \\ \boldsymbol{\eta} & (\mathbf{I} - \boldsymbol{\Gamma}_o) \end{bmatrix}$$

and that

$$(\mathbf{I} - \boldsymbol{\Gamma}^*)^{-1} = \begin{bmatrix} 1 + \gamma_j' \mathbf{L}^{-1} \boldsymbol{\eta} & \gamma_j' \mathbf{L}^{-1} \\ \\ \mathbf{L}^{-1} \boldsymbol{\eta} & \mathbf{L}^{-1} \end{bmatrix}$$

, where

$$\mathbf{L} = \mathbf{I} - \boldsymbol{\Gamma}_o - \boldsymbol{\eta}\gamma_j' = (\mathbf{I} - \mathbf{G}) \tag{A.17}$$

Accordingly

$$\frac{1}{\sigma_{jj}} \mathbf{M}_1' (\mathbf{I} - \boldsymbol{\Gamma}^*)^{-1} \boldsymbol{\Sigma} \mathbf{e}_1 = (\mathbf{I} - \mathbf{G})^{-1} \mathbf{d} = \boldsymbol{\varphi}_j, \tag{A.18}$$

where $\mathbf{e}_j$ is the first elementary vector, $\boldsymbol{\Sigma}$, $\mathbf{G}$ and $\mathbf{d}$ are defined as in (33)and (37) respectively, and $\mathbf{M}_1$ is the selection matrix obtained from the identity matrix by deleting its first column. Now, taking into account that the following holds

$$(\mathbf{I} - \boldsymbol{\Gamma}^*) = \mathbf{P}_j (\mathbf{I} - \boldsymbol{\Gamma}) \mathbf{P}_j \tag{A.19}$$

in light of (A.16), and that the following proves true

$$(\mathbf{I} - \boldsymbol{\Gamma}^*)^{-1} = \mathbf{P}_j (\mathbf{I} - \boldsymbol{\Gamma})^{-1} \mathbf{P}_j, \tag{A.20}$$

as $\mathbf{P}_j$ is both symmetric and orthogonal, some computations yield

$$\begin{aligned} \boldsymbol{\varphi}_j &= \frac{1}{\sigma_{jj}} \mathbf{M}_1' (\mathbf{I} - \boldsymbol{\Gamma}^{*'})^{-1} \boldsymbol{\Sigma} \mathbf{e}_1 = \frac{1}{\sigma_{jj}} \mathbf{M}_1' \mathbf{P}_j (\mathbf{I} - \boldsymbol{\Gamma})^{-1} \mathbf{P}_j \boldsymbol{\Sigma} \mathbf{P}_j \mathbf{P}_j \mathbf{e}_1 = \\ &= \frac{1}{\sigma_{jj}} \mathbf{M}_j' (\mathbf{I} - \boldsymbol{\Gamma})^{-1} \boldsymbol{\Sigma} \mathbf{e}_j = \frac{1}{\sigma_{jj}} \mathbf{M}_j' \mathbf{F}' \mathbf{e}_j \end{aligned} \tag{A.21}$$

■

4. **Derivation of the log-likelihood for the model** (41)-(43)
The logarithm of the density in (43) is given by

$$\ln f_{(\epsilon_j, \boldsymbol{\epsilon}_r)} = c - \frac{1}{2} \ln \sigma_{jj} - \frac{1}{2} \ln |\boldsymbol{\Omega}| - \frac{\epsilon_j^2}{2\sigma_{jj}} - \frac{1}{2} \boldsymbol{\epsilon}_r' \boldsymbol{\Omega}^{-1} \boldsymbol{\epsilon}_r \tag{A.22}$$

where c is a constant term. Now, upon noting that

$$|\mathbf{J}| = \left| \frac{\partial(\epsilon_j, \boldsymbol{\epsilon}_r)'}{\partial(y_j, \mathbf{y}_r')} \right| = 1, \tag{A.23}$$

and assuming to operate with N observations on the variables of interest, the log-likelihood function can be written as

$$l = \sum_{t=1}^{N} l_{(y_j,\mathbf{y}_r')} = k - \frac{N}{2}\ln\sigma_{jj} - \frac{N}{2}\ln|\boldsymbol{\Omega}| - \frac{\boldsymbol{\alpha}'\mathbf{H}\boldsymbol{\alpha}}{2\sigma_{jj}} - \frac{1}{2}\mathrm{tr}(\boldsymbol{\Xi}'\boldsymbol{\Omega}^{-1}\boldsymbol{\Xi}\mathbf{H}) \tag{A.24}$$

where

$$\boldsymbol{\alpha}' = \left[1,\ -\tilde{\boldsymbol{\gamma}}_j',\ -\tilde{\mathbf{a}}_j'\mathbf{S}_j\right] \tag{A.25}$$

$$\boldsymbol{\Xi} = \left[-\tilde{\boldsymbol{\varphi}}_j,\ \mathbf{I} + \tilde{\boldsymbol{\varphi}}_j\tilde{\boldsymbol{\gamma}}_j', \tilde{\boldsymbol{\varphi}}_j\tilde{\mathbf{a}}_j'\mathbf{S}_j - \boldsymbol{K}\right] \tag{A.26}$$

$$\boldsymbol{\nu}' = \left[y_j, \mathbf{y}_o, \mathbf{z}\right] \tag{A.27}$$

$$\mathbf{H} = \left(\sum_{t=1}^{N}\boldsymbol{\nu}_t\boldsymbol{\nu}_t'\right), \tag{A.28}$$

and k is a constant term. Formula (A.24) can be obtained by noting that, in light of (41), the following holds

$$\epsilon_j = y_j - \tilde{\boldsymbol{\gamma}}_j'\mathbf{y}_r - \tilde{\mathbf{a}}_j'\mathbf{z}_r = \left[1, -\tilde{\boldsymbol{\gamma}}_j', -\tilde{\mathbf{a}}_j'\mathbf{S}_r\right]\begin{bmatrix} y_j \\ \\ \mathbf{y}_r \\ \\ \mathbf{z}\end{bmatrix} = \boldsymbol{\alpha}'\boldsymbol{\nu} \tag{A.29}$$

and that, according to (42), we have

$$\boldsymbol{\epsilon}_r = \mathbf{y}_r - \mathbf{K}\mathbf{z} - \tilde{\boldsymbol{\varphi}}_j\epsilon_j = \tag{A.30}$$

$$= \mathbf{y}_r - \mathbf{K}\mathbf{z} - \tilde{\boldsymbol{\varphi}}_j(y_j - \tilde{\boldsymbol{\gamma}}_j'\mathbf{y}_r - \tilde{\mathbf{a}}_j'\mathbf{S}_r\mathbf{z}) = \tag{A.31}$$

$$= \left[-\tilde{\boldsymbol{\varphi}}_j, \mathbf{I} + \tilde{\boldsymbol{\varphi}}_j\tilde{\boldsymbol{\gamma}}_j', \tilde{\boldsymbol{\varphi}}_j\tilde{\mathbf{a}}_j'\mathbf{S}_r - \mathbf{K}\right]\begin{bmatrix} y_j \\ \\ \mathbf{y}_r \\ \\ \mathbf{z}\end{bmatrix} = \boldsymbol{\Xi}\boldsymbol{\nu} \tag{A.32}$$

This implies that

$$\boldsymbol{\epsilon}_r'\boldsymbol{\Omega}^{-1}\boldsymbol{\epsilon}_r = \mathrm{tr}(\boldsymbol{\nu}'\boldsymbol{\Xi}'\boldsymbol{\Omega}^{-1}\boldsymbol{\Xi}\boldsymbol{\nu}) = \mathrm{tr}(\boldsymbol{\Xi}'\boldsymbol{\Omega}^{-1}\boldsymbol{\Xi}\boldsymbol{\nu}\boldsymbol{\nu}') \tag{A.33}$$

■

Gianmarco Vacca
Department of Economic Policy
Largo Gemelli 1, 20123 Milan, Italy
Italy
gianmarco.vacca@unicatt.it

Maria Grazia Zoia
Department of Economic Policy
Largo Gemelli 1, 20123 Milan, Italy
Italy
maria.zoia@unicatt.it

BINCOR: An R package for Estimating the Correlation between Two Unevenly Spaced Time Series

by Josue M. Polanco-Martinez, Martin A. Medina-Elizalde, Maria Fernanda Sanchez Goni, Manfred Mudelsee

Abstract This paper presents a computational program named **BINCOR** (BINned CORrelation) for estimating the correlation between two unevenly spaced time series. This program is also applicable to the situation of two evenly spaced time series not on the same time grid. **BINCOR** is based on a novel estimation approach proposed by Mudelsee (2010) for estimating the correlation between two climate time series with different timescales. The idea is that autocorrelation (e.g. an AR1 process) means that memory enables values obtained on different time points to be correlated. Binned correlation is performed by resampling the time series under study into time bins on a regular grid, assigning the mean values of the variable under scrutiny within those bins. We present two examples of our **BINCOR** package with real data: instrumental and paleoclimatic time series. In both applications **BINCOR** works properly in detecting well-established relationships between the climate records compared.

Introduction

There are several approaches for quantifying the potential association between two evenly spaced climate time series, e.g. Pearson's and Spearman's correlation or the cross-correlation function (CCF). However, these methods should not be directly applied when the time series are unevenly spaced ("irregular"), particularly when two time series under analysis are not sampled at identical points in time, as is usually the case in climate research, especially in paleoclimate studies (Emile-Geay, 2016; Mudelsee, 2014; Weedon, 2003). The most common way of tackling this problem is to interpolate the original unevenly spaced climate time series in the time domain so as to obtain equidistance and the same times. The series can then be analysed using existing conventional correlation analysis techniques. However, experience shows that interpolation has its drawbacks: depending on the features of the method applied, the interpolated time series may show deviations in terms of variability or noise properties, and additional serial dependence may be introduced (Horowitz, 1974; Mudelsee, 2014; Olafsdottir and Mudelsee, 2014). Thus, interpolation should be avoided as far as possible.

Fortunately, there are some algorithms and software available to carry out this task, at least for unevenly spaced climate time series sampled at identical points in time (Mudelsee, 2003; Olafsdottir and Mudelsee, 2014). However, there are few statistical techniques for estimating the correlation between two time series not sampled at identical points in time and their corresponding computational implementations. One exception is the Gaussian-Kernel-based cross-correlation (gXCF) method and its associated software named NESTOOLBOX (Rehfeld et al., 2011; Rehfeld and Kurths, 2014; Rehfeld and Bedartha, 2014) and the extended version (Roberts et al., 2017) that includes a confidence interval obtained by a bootstrapping resampling approach; another exception is binned correlation as proposed by Mudelsee (2010, 2014). However, the software for this method is not freely available on the Internet.

Binned correlation is a statistical technique developed to estimate the correlation between two unevenly spaced time series sampled at different points in time. It is also applicable to two evenly spaced time series that are not on the same time grid (Mudelsee, 2014). It is performed by resampling the time series into time bins on a regular grid, and then assigning the mean values of the variable under scrutiny within those bins. Mudelsee (2010) proposes a novel approach adapting the binned correlation technique (used mainly with astronomical data) to analyse climate time series taking into account their memory (or persistence), which is a genuine property of climate time series. Autocorrelation, persistence, memory or serial dependence is characteristic of weather and climate fluctuations, and is recorded in climate time series (Wilks, 2011; Mudelsee, 2002). A simple persistence model used to "represent" climate time series is a first-order autoregressive (AR1) process where a fluctuation depends only on its own immediate past plus a random component (Gilman et al., 1963; Mann and Lees, 1996; Mudelsee, 2002). However, paleoclimate time series are usually unevenly spaced in time, and it is necessary to use an AR1 version for the case of uneven spacing, such as the method proposed by Robinson (1977). The technique of Mudelsee (2010) requires the concept of nonzero persistence times, enabling the mixing information (i.e. covariance) to be recovered, even when the two timescales differ. The **BINCOR** package presented in this paper is based on a method that is not applicable when one or both of the time series under examination have zero persistence. Similarly, this method is not applicable when the time series are sampled with significantly longer spacing than the persistence time, so that the effectively sampled persistence time is zero. A fundamental condition for using this method is that the time spacing should not be much larger than the persistence times. Enough common data points then fall within a time bin, and knowledge can be acquired on the covariance (Mudelsee, 2010, 2014).

In this paper we present a computational package named **BINCOR** (BINned CORrelation), which is based on the approach proposed by Mudelsee (2010, 2014). The **BINCOR** package contains (i) a main function named `bin_cor`, which is used to convert the irregular time series to a binned time series; (ii) two complementary functions (`cor_ts` and `ccf_ts`) for computing the correlation between the two binned climate time series obtained with the `bin_cor` function; and (iii) an additional function (`plot_ts`) for plotting the "primary" vs. the binned time series. This package is programmed in R language and is available at the CRAN repository (https://CRAN.R-project.org/package=BINCOR).

This paper is divided into four sections. The first outlines the method and the computational program. The second presents a Monte Carlo experiment to study the effect of binning size selection. In the Examples section we apply **BINCOR** to a couple of unevenly spaced real-world climate data sets: instrumental and paleoclimate. Finally, the Summary section presents our main conclusions.

The BINCOR package

The method

In this section we outline the main mathematical ideas behind the binned correlation technique for unevenly spaced sampled at different points in time, following the methodology introduced by Mudelsee (2010, 2014). The procedure is described as follows:

1. Input: two unevenly spaced climate time series $\{X(i), T_X\}_{i=1}^{N_X}$ and $\{Y(i), T_Y\}_{i=1}^{N_Y}$, where T_X, T_Y and N_Y, N_Y are the time domains and the sample sizes of each series, respectively.

2. Compute the average spacing between samples
   - $\bar{d}_X = [T_X(N_X) - T_X(1)]/(N_X - 1)$
   - $\bar{d}_Y = [T_Y(N_X) - T_Y(1)]/(N_Y - 1)$
   - $\bar{d}_{XY} = [\bar{T}_{\max} - \bar{T}_{\min}]/(N_X + N_Y - 1)$

   where $\bar{T}_{\max} = \max[T_X(N_X), T_Y(N_Y)]$ and $\bar{T}_{\min} = \min[T_X(1), T_Y(1)]$.

3. Estimate the bin-width ($\bar{\tau}$) taking into account the persistence (memory) estimated for each unevenly spaced climate time series, X and Y denoted as $\hat{\tau}_X$ and $\hat{\tau}_Y$, respectively. To estimate the persistence, an AR1 model (Robinson, 1977) is fitted to each unevenly spaced time series (Mudelsee, 2002). **BINCOR** includes three rules for estimating the bin-width (the options are shown in Table 1), but we prefer to use rule number 3 as the default value (FLAGTAU=3) because in terms of the RMSE (Section Monte Carlo experiments) of this rule Monte Carlo simulations are superior to the other rules for estimating the bin-width (Mudelsee, 2014).
   - Estimate the bias-corrected equivalent autocorrelation coefficients
     $$\hat{\bar{a}}'_X = \exp(-\bar{d}_X/\hat{\tau}'_X),\ \hat{\bar{a}}'_Y = \exp(-\bar{d}_Y/\hat{\tau}'_Y)\ \text{, and}\ \hat{\bar{a}}'_{XY} = \sqrt{\hat{\bar{a}}'_X \cdot \hat{\bar{a}}'_Y}$$
   - Estimate the bin-width as $\bar{\tau} = -\bar{d}_{XY}/\ln(\hat{\bar{a}}'_{XY})$ (Eq. 7.48 in Mudelsee (2002)), the default option (FLAGTAU=3) in the BINCOR package, other options are:

4. Determine the number of bins: $N_b = (\bar{T}_{\max} - \bar{T}_{\min})/\bar{\tau}$

5. Set: $\lim_{\inf}(n = 1) = \bar{T}_{\min}$. Then, for $n = 1, 2, \ldots, N_b$, define (Figure 1):

   (a) $\lim_{\sup}(n) = \bar{T}_{\min} + n \cdot \bar{\tau}$

   (b) $\text{id}T_X = \text{WHICH}\ [T_X \geqslant \lim_{\inf}(n)\ \text{AND}\ T_X \leqslant \lim_{\sup}(n)]$

   (c) $\text{id}T_Y = \text{WHICH}\ [T_Y \geqslant \lim_{\inf}(n)\ \text{AND}\ T_Y \leqslant \lim_{sup}(n)]$

   (d) $\text{L}T_X = \ \text{LENGTH}(\text{id}T_X)$

   (e) $\text{L}T_Y = \ \text{LENGTH}(\text{id}T_Y)$
   if ($\text{L}T_X > 0$ AND $\text{L}T_Y > 0$)

$\bar{\tau}$ rule	FLAGTAU option	Reference
$\tau_x + \tau_y$	1	Eq. 7.44 in Mudelsee (2014)
$\max(\tau_x, \tau_y)$	2	Eq. 7.45 in Mudelsee (2014)
$-\bar{d}_{XY} / \ln(\hat{\bar{a}}'_{XY})$	3	Eq. 7.48 in Mudelsee (2014)

Table 1: The FLAGTAU options and its corresponding methods (rules) to estimate the bin-width.

i. $F(n)$ = mean of $X(\mathrm{id}T_X)$
ii. $G(n)$ = mean of $Y(\mathrm{id}T_Y)$
iii. $T(n) = [\lim_{\inf}(n) + \lim_{\sup}(n)]\ /\ 2$

(f) $\lim_{\inf}(n) = \lim_{\sup}(n)$

6. Output: two binned climate time series $\{T_n,\ F(n)\}_{n=1}^{N_b}$ and $\{T_n, G(n)\}_{n=1}^{N_b}$, where N_b is the number of bins.
7. Estimate the correlation between the two binned time series. This can be done through the native R functions `cor` and `ccf` or by means of the **BINCOR** functions `cor_ts` and `ccf_ts`.

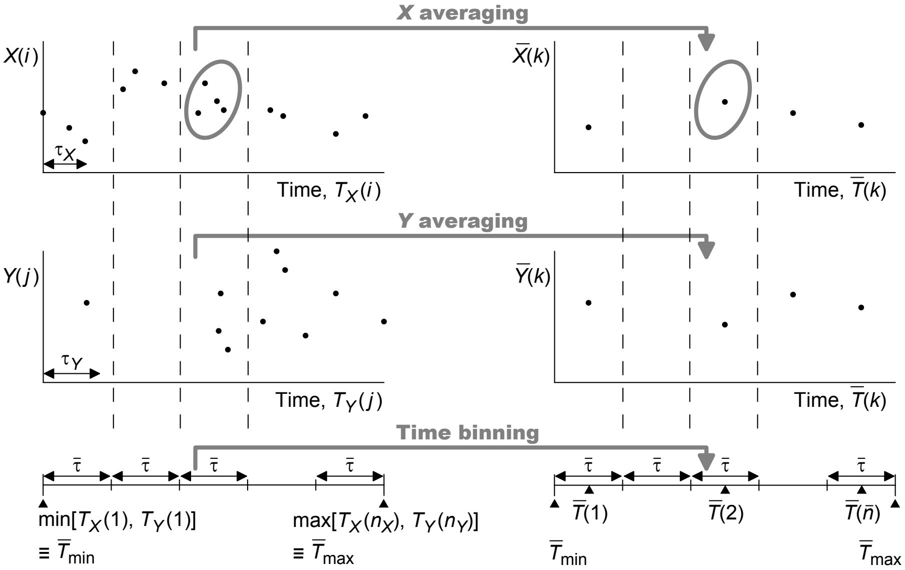


Figure 1: Graphical representation for the binned correlation procedure presented in Step 5. Modified from (Mudelsee, 2010, 2014).

Monte Carlo experiments

We conducted Monte Carlo experiments to study how the specific rules (Table 1) chosen for calculating the bin-width based on persistence reduce the error compared to arbitrarily choosing a bin-width. The parameter configuration for the Monte Carlo experiments is presented in Figure 2. To carry out the Monte Carlo simulations, we used the bivariate Gaussian AR1 process for uneven time spacings (Mudelsee, 2014), which is given by

$$\begin{aligned} X(1) &= \mu^X_{N(0,1)}(1), \\ Y(1) &= \mu^Y_{N(0,1)}(1), \\ X(t) &= a_X X(t-1) + \mu^X_{N(0,1-a_X^2)}(t), \quad t = 2, ..., N, \\ Y(t) &= a_Y Y(t-1) + \mu^Y_{N(0,1-a_Y^2)}(t), \quad t = 2, ..., N, \end{aligned} \tag{B.3.1}$$

where a_X and a_Y, the autoregressive parameters for $X(t)$ and $Y(t)$, are defined as (Mudelsee, 2014): $a_X = exp\{-[T_X(t) - T_X(t-1)]/\tau_X\}$ and $a_Y = exp\{-[T_Y(t) - T_Y(t-1)]/\tau_Y\}$. The correlation (by construction) between $X(t)$ and $Y(t)$ is ρ_{XY} (see Mudelsee, 2014, pp. 307-309 for more details about the statistical properties of the bivariate AR1 process for unevenly spaced time series). To generate the uneven timescales for $X(i)$ and $Y(j)$, we follow the methodology proposed by (see Mudelsee, 2014, pp. 299-304), which consists of producing a number (10 N) of data pairs on an evenly spaced grid of 1.0, discarding 90% of points and retaining 10% of X and Y ($N_x = N_y = N$) points. The time points for $X(i)$ and $Y(j)$ are subject to the following conditions:

1. Control case (equal timescales):
   - Condition 1: $N_X = N_Y$
   - Condition 2: $\{T_X(i)\}_{i=1}^{N_X} = \{T_Y(j)\}_{j=1}^{N_Y}$
2. "Well" mixed unequal timescales:
   - Condition 1: $T_X(i) \neq T_Y(j)$ for all i and j
   - Condition 2: $T_X(1) < T_Y(1) < T_X(2) < T_Y(2) < T_X(3) < ... < T_X(N_X) < T_Y(N_Y)$
3. "Wildly" mixed unequal timescales:
   - There are not conditions for this case.

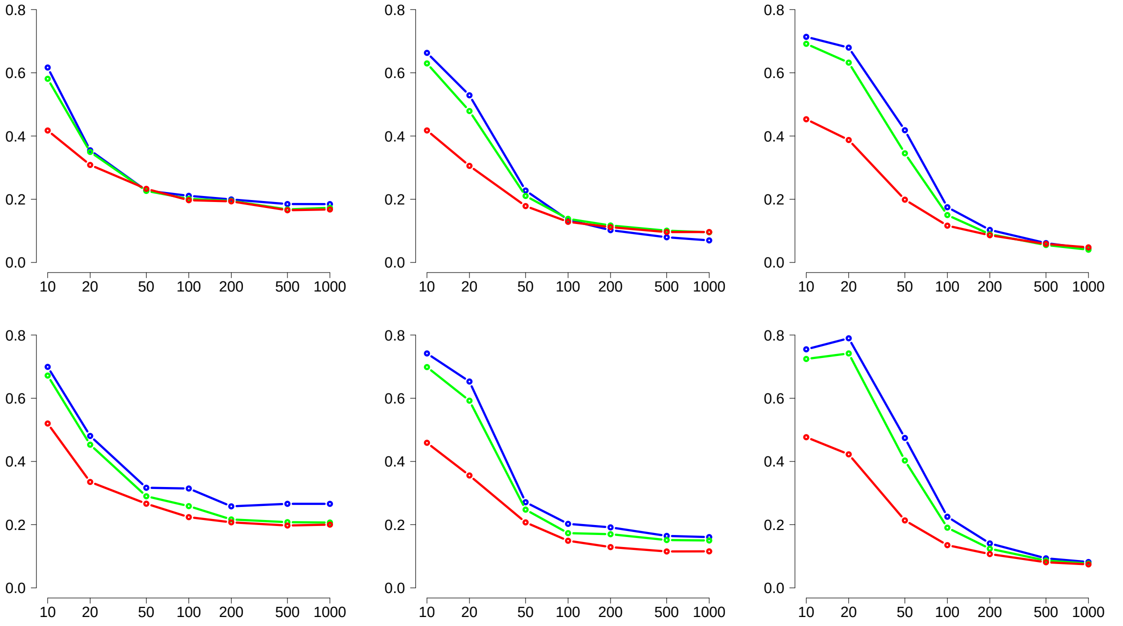


Figure 2: Monte Carlo experiments to test the impact of the rules (Table 1) used to calculate the bin-width and their role in the estimation of the binned correlation. The persistence figures for X and Y are 10 (column 1), 20 (column 2) and 50 (column 3), respectively. The constraints for the resampling timescales are for well mixed (first row) and wildly mixed (second row) cases. The horizontal axis indicates the sample sizes (in log10 scale) and the vertical axis shows the RMSE that is determined via averaging $(\hat{\rho}_{XY} - \rho_{XY})^2$ over 5,000 simulations. The blue, green and red curves indicate rules 1 (sum), 2 (max) and 3 (the default rule option in **BINCOR**).

The outcome of the Monte Carlo experiments is as follows: 1) For equal timescales (figures not shown), all three rules behave similarly (as expected) in terms of RMSE, although the RMSE increases slightly as the persistence increases. 2) The well mixed case shows that for RMSE the rules take two different "patterns" with the first two rules (sum and max) on one hand and the third rule (the default rule option) on the other. This difference is most noticeable in the first values of the samples (from 10 to 100) and is most pronounced with high persistence values (τ_x and τ_y). The rule that shows the smallest RMSE is rule 3 (the default option), though it is important to point out that for $\tau_x = \tau_y = 50$ the RMSE figures are practically indistinguishable for sample sizes from 200 to 1000. 3) Finally, RMSE in the wildly mixed case behaves more or less similarly to the well mixed case, though rule 3 yields the smallest RMSE for all three persistence values. Bearing in mind that the wildly mixed case does not impose conditions on generating timescales, and in practice the unevenly spaced climate time series could contain some degree of randomness in the sampling times, the best rule in terms of RMSE for estimating bin-width ($\bar{\tau}$) and binned correlation can be said to be number 3, i.e. the default rule used in **BINCOR** to estimate the bin-width.

The computer program

The **BINCOR** package developed in R version 3.1.2[1] to be run from the command line runs on all major operating systems and is available from the CRAN repository (http://CRAN.R-project.org/package=BINCOR). The **BINCOR** package contains four functions: 1) `bin_cor` (the main function for building the binned time series); 2) `plot_ts` (for plotting and comparing the "primary" and binned time series); 3) `cor_ts` (for estimating the correlation between the binned time series); and 4) `ccf_ts` (for estimating the cross-correlation between the binned time series). The graphical outputs can be displayed on the screen or saved as PNG, JPG, or PDF graphics files. **BINCOR** depends on the **dplR** (Bunn et al., 2015) and **pracma** (Borchers, 2015) packages. The **dplR** package is used by the function `bin_cor` to calculate the persistence for the climate time series under study, whereas the **pracma** package is used by the functions `cor_ts` and `ccf_ts` to remove the linear trend before estimating the correlation.

The first (and main) function, `bin_cor`, estimates the binned time series taking into account the memory or persistence of the unevenly spaced climate time series to be analysed (Mudelsee, 2002). It has the following syntax:

```
R> bin_cor(ts1, ts2, FLAGTAU=3, ofilename),
```

where

- `ts1` and `ts2` are unevenly spaced time series.
- `FLAGTAU` defines the method used to estimate the bin-width ($\bar{\tau}$). There are three methods included in **BINCOR** for estimating bin-width (Table 1), but we prefer to use (`FLAGTAU = 3`) as the default rule because Monte Carlo simulations perform better in terms of RMSE than the other rules in estimating the bin-width and the binned correlations (Mudelsee, 2014).
- 'ofilename' is the name of the output file (in ASCII format) which contains the binned time series.

`bin_cor` returns a list object containing the following outputs:

```
"Binned_time_series", "Auto._cor._coef._ts1", "Persistence_ts1", "Auto._cor._coef._ts2",
"Persistence_ts2", "bin width", "Number_of_bins",  "Average spacing", "VAR. ts1",
"VAR. bin ts1", "VAR. ts2", "VAR. bin ts2", "VAR. ts1 - VAR bints1",
"VAR. ts2 - VAR bints2", "% of VAR. lost ts1", "% of VAR. lost ts2".
```

The names of the outputs are self-explanatory, but we wish to highlight that `Average spacing` is the mean value of the times for the binned time series; `VAR. ts1`, `VAR. bin ts1`, `VAR. ts2` and `VAR. bin ts2` are the variances for `ts1` and `ts2` for their respective binned time series; the next two outputs are the differences between the variances of `ts1` and `ts2` and their corresponding binned time series; and the last two outputs are the percentages of variance lost for `ts1` and `ts2` as a result of the binned process.

The second function, called `plot_ts`, plots the "primary" (unevenly spaced) time series and the binned time series. The `plot_ts` function contains the following elements:

```
R> plot_ts(ts1, ts2, bints1, bints2, varnamets1="", varnamets2="",
   colts1=1, colts2=1, colbints1=2, colbints2=2, ltyts1=1,
   ltyts2=1, ltybints1=2, ltybints2=2, device="screen", ofilename),
```

[1] It was also tested in R 3.4.1.

where the input arguments ts1 and ts2 are the unevenly spaced time series, bints1 and bints2 are the binned time series, varnamets1 and varnamets2 are the names of the variables under study, colts1, colts2 (by default both curves are in black) and colbints1, colbints2 (by default both curves are in red) are the colours for the "primary" and binned times series; ltyts1, ltyts2, ltybints1 and ltybints2 are the types of line to be plotted for the "primary" and binned times series, respectively (1 = solid, 2 = dashed, 3 = dotted, 4 = dot-dashed, 5 = long-dashed, 6 = double-dashed); device is the type of output device ("screen" by default, the other options being "jpg," "png," and "pdf"); resfig is the image resolution in "ppi" (by default R does not record a resolution in the image file, except for BMP; 150 ppi could be a suitable value); 'ofilename' is the output filename; and finally, Hfig, WFig and Hpdf, Wpdf are the height and width of the output for the JPG/PNG and PDF formats, respectively.

The third function, cor_ts, calculates three types of correlation coefficient: Pearson's correlation, Spearman's and Kendall's rank correlations. These correlation coefficients are estimated through the native R function cor.test from the R package Stats. The cor_ts function has an option to remove the linear trend of the time series under analysis – other pre-processing methods could be used before the cor_ts function is applied. This function has the following syntax:

```
R> cor_ts(bints1, bints2, varnamets1="", varnamets2="",
   KoCM, rmltrd="N", device="screen", Hfig, Wfig, Hpdf, Wpdf,
   resfig, ofilename)
```

where KoCM indicates the correlation estimator: pearson for Pearson (the option by default), spearman for Spearman and kendall for Kendall; rmltrd is the option to remove the linear trend in the time series under study (by default the linear trend is not removed, but the function can be enabled via the option "Y" or "y"). The other parameters are described some lines above. cor_ts has as its output a list object containing the main information for the estimated correlation coefficient (e.g. a 95% confidence interval for Pearson and a p-value for Spearman and Kendall). The cor_ts function also provides a scatterplot for the binned time series, which can be plotted on the screen (by default) or saved in JPG, PNG or PDF formats (the parameter 'ofilename' is available to assign a name to this output).

Finally, the fourth function, ccf_ts, estimates and plots the cross-correlation between two evenly spaced paleoclimate time series. We use the native R function ccf (R Stats package) to estimate the cross-correlation in our ccf_ts function. The ccf_ts function has the following syntax:

```
R> ccf_acf <- ccf_ts(bints1, bints2, lagmax=NULL, ylima=-1, ylimb=1,
          rmltrd="N", RedL=T, device="screen", Hfig, Wfig,
          Hpdf, Wpdf, resfig, ofilename)
```

All these elements are already defined above except the parameters lagmax=NULL, ylima=-1, ylimb=1 and RedL. The first parameter indicates the maximum lag for which the cross-correlation is calculated (its value depends on the length of the data set), the next two parameters indicate the extremes of the range in which the CCF will be plotted and the last parameter (the default option is TRUE) plots a straight red line to highlight the correlation coefficient at lag 0. The ccf_ts function generates as its output the acf (auto-correlation function; ACF) R object, which is a list with the following parameters: lag is a three dimensional array containing the lags at which the ACF is estimated; acf is an array with the same dimensions as lag containing the estimated ACF; type is the type of correlation (correlation (the default), covariance and partial); n.used is the number of observations in the time series; and snames provides the names of the time series (bints1 and bints2).

Examples

Assessing the link between El Niño-Southern Oscillation and Northern Hemisphere sea surface temperature

We first examine two evenly-spaced annually-resolved instrumental climate records that cover the time interval from 1850 to 2006 ($N = 157$ points)[2]. To test our **BINCOR** package we created irregular time series by randomly removing 20% of the data from the evenly spaced time series. We note that the new "sampling" times are not necessarily the same for both irregular series. The new irregular time series ("primary" hereafter) consist of 125 data points and have an average temporal spacing $\bar{d}$ of 1.24 years. Specifically the two time series used were a record of Northern Hemisphere

[2] The data sets can be obtained from the following URL http://www.meteo.psu.edu/holocene/public_html/supplements/MultiproxySpatial09/results/ (NINO3 full and Northern Hemisphere full).

(NH) sea surface temperature (SST) anomalies (HadCRUT3, Brohan et al. (2006)) and a record of equatorial Pacific SST anomalies from the El Niño 3 region (2.5°S to 2.5°N, 92.5 to 147.5°W) (Mann et al., 2009), which is a indicator of El Niño-Southern Oscillation (ENSO). Both time series, especially the NH-SST data, show strong autocorrelation (plots not shown) and long-term trends (inspected by Mann-Kendall test; ENSO, `z`=6.52 and `p-value` < 0.001 and NH-SST, `z` $= 10.214$ and `p-value` < 0.001). To generate the sample data, we fit a linear model to each evenly spaced time series and, after removing the model fitted to the evenly spaced data, we use the residuals (i.e. the difference between the observed data and the model fitted) to build the irregular time series and then create the binned time series.

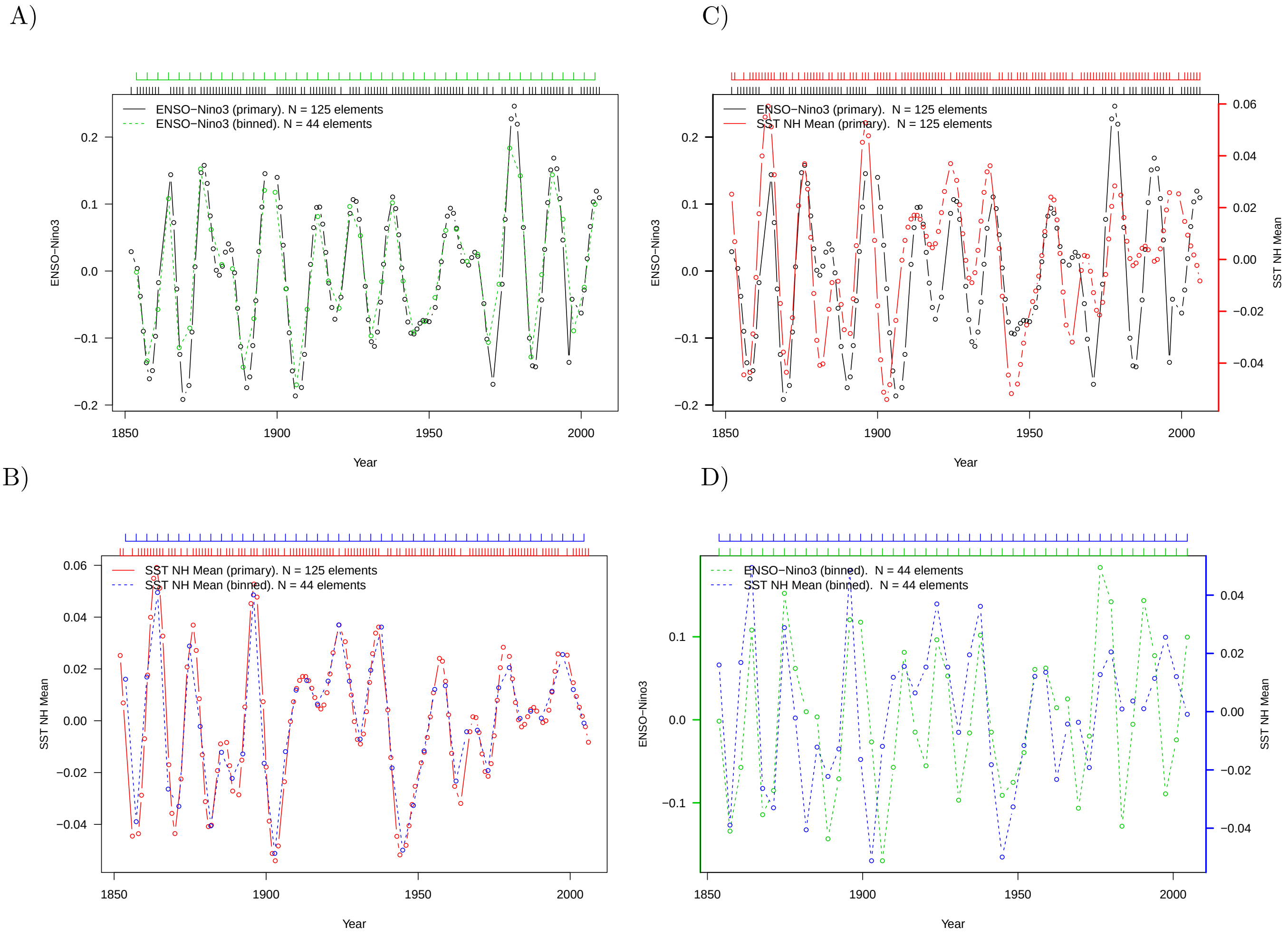


Figure 3: "Primary" (unevenly spaced) and binned ENSO-Niño3 (Mann et al., 2009) and NH-SST (Brohan et al., 2006). The autocorrelation and persistence values for ENSO are $\hat{\bar{a}}' = 0.82$ and $\hat{\tau} = 6.25$ years, and for NH-SST are $\hat{\bar{a}}' = 0.86$ and $\bar{\tau} = 8.05$ years. The horizontal top axes indicate the sampling times for the plotted time series.

The code used to generate Figure 3 is shown below.

```
# Load the package
library(BINCOR)

# Load the time series under analysis: Example 1 and Figure 1 (ENSO vs. NHSST)
data(ENSO)
data(NHSST)

# Compute the binned time series though our bin_cor function
bincor.tmp <- bin_cor(ENSO.dat, NHSST.dat, FLAGTAU=3, "output_ENSO_NHSST.tmp")
binnedts   <- bincor.tmp$Binned_time_series
```

```
# Applying our plot_ts function
# "Screen"
plot_ts(ENSO.dat, NHSST.dat, binnedts[,1:2], binnedts[,c(1,3)], "ENSO-Nino3",
"SST NH Mean", colts1=1, colts2=2, colbints1=3, colbints2=4, device="screen")
```

Figures 3 A and 3 B show the binned time series (ENSO in green and NH-SST in red) obtained with our bin_cor function. Although we use residuals, they show a relative high autocorrelation ($\hat{\bar{a}}'_{\text{ENSO}} = 0.82$ and $\hat{\bar{a}}'_{\text{SST}} = 0.86$) and their corresponding estimated bias-corrected persistence values are $\hat{\bar{\tau}}_{\text{ENSO}} = 6.25$ years and $\bar{\tau}_{\text{SST}} = 8.05$ years. The number of bins and, thus, the number of elements for each binned time series is 44 and the distance between elements is 3.5 years. We also plot the "primary" climate time series (in black) to compare them with the binned series. Visually, the binned time series are roughly similar to the "primary" series. This observation is also supported by the statistical similarity method (Frentzos et al., 2007) as implemented in the R package **TSdist** (Mori et al., 2015, 2016). The dissimilarity metric (DISSIM) has the following interpretation: a value of zero indicates a perfect relationship such that the closer DISSIM is to zero, the more similar are the time series. The DISSIM between the binned and "primary" ENSO time series and the binned and "primary" NH-SST series are 3.70 and 0.84, respectively. This corroborates the similarity between the "primary" and binned time series observed visually. Figure 3 also shows a comparison between the "primary" climate time series (Figure 3 C) and the binned series (Figure 3 D). Note that this plot shows that the number of elements ($N = 125$) is the same for both "primary" series, but this is not strictly necessary: our bin_cor function is able to tackle time series with different numbers of elements.

The second result obtained from our **BINCOR** package, and more specifically from the cor_ts function, is shown in Figure 4, which shows the scatterplot between the ENSO (x-axis) and NH-SST (y-axis) binned time series. This scatterplot shows a moderate increasing trend from left to right, suggesting a potentially positive relationship between the two binned time series. This pattern can be confirmed statistically by means of the cor_ts function output, which also provides the correlation coefficient between two time series under analysis. For this case, the Pearson's correlation (with 95% confidence interval) obtained is $\bar{r}_{XY} = 0.53$ [0.28; 0.71] (other estimators can also be used in cor_ts). This value is close to the Pearson's correlation estimated for the evenly spaced climate time series, which is $\bar{r}_{XY} = 0.58$ [0.46; 0.67]. The relatively high correlation obtained between these two climate records is expected; ENSO-related climate variability is observed in many regions outside the equatorial Pacific, particularly in the tropical North Atlantic (Enfield and Mayer, 1997; Garcia-Serrano et al., 2017).

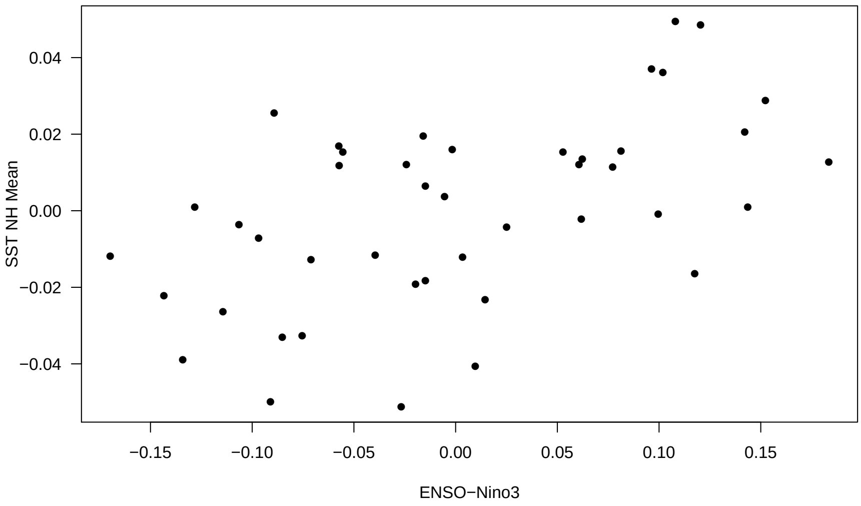


Figure 4: Scatterplot for the ENSO-Niño3 (Mann et al., 2009) and NH-SST (Brohan et al., 2006) binned time series. The Pearson's correlation coefficient (with 95% confidence interval) is $\bar{r}_{XY} = 0.53$ [0.28; 0.71].

The code used to generate Figure 2 is shown below.

```
# Load packages
library(BINCOR)
library(pracma)

# Load the time series under analysis: Example 1 and Figure 2 (ENSO vs. NHSST)
data(ENSO)
data(NHSST)

# Compute the binned time series though our bin_cor function
bincor.tmp <- bin_cor(ENSO.dat, NHSST.dat, FLAGTAU=3, "output_ENSO_NHSST.tmp")
binnedts   <- bincor.tmp$Binned_time_series

# Compute the scatterplot by means of our function cor_ts
# PDF format (scatterplot) and Pearson
cor_ts(binnedts[,1:2], binnedts[,c(1,3)], "ENSO-Nino3", "SST NH Mean",
KoCM="pearson", rmltrd="y", device="pdf", Hpdf=6, Wpdf=9, resfig=300,
ofilename="scatterplot_ENSO_SST")
```

Abrupt climate changes during the last glacial

We report an analysis of two temporally unevenly-spaced pollen records from two marine sediment cores (MD04-2845 and MD95-2039)[3] collected on the south-western European margin (Figure 5). The aim of this case study is to show the use of **BINCOR** to estimate the correlation between two unevenly spaced paleoclimate time series by means of the cross-correlation function. The pollen time series analysed in this example span the interval between 73,000 and 15,000 years before present (BP), thus covering the last glacial period (LGP). The climate during the LGP was characterised by millennial variability with "abrupt" transitions between cold stadials and warm interstadials known as Dansgaard-Oeschger (D-O) cycles (Dansgaard et al., 1993; Wolff et al., 2012). The D-O cycles are characterised by rather fast atmospheric warming events over Greenland of up to 16 °C that occur within a period of approximately 40 years, followed by gradual cooling leading to the cold stadials (**?**Wolff et al., 2012).

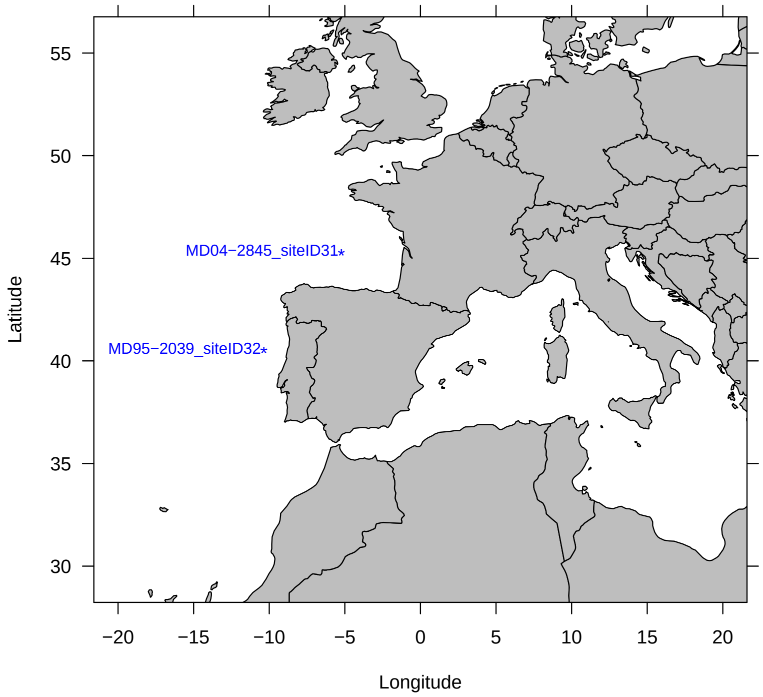


Figure 5: Geographical locations for the pollen time series under analysis (**?**). The labels indicate the names of the sites where the pollen data were obtained.

[3] The data sets can be obtained from `https://doi.pangaea.de/10.1594/PANGAEA.870867`. These time series come from a global pollen and charcoal database (**?**) drawn up under the framework of the INQUA International Focus Group ACER (Abrupt Climate Changes and Environmental Responses).

Figure 6 illustrates the variations in the pollen percentages of the temperate forest, a type of vegetation typical of moderate, warm, wet climates. Figure 6 A shows the primary and binned pollen records from site MD04-2845 (Sanchez Goni et al., 2008; **?**). Figure 6 B shows the primary and binned pollen records from site MD95-2039 (Roucoux et al., 2005; **?**). We use the pollen time series with a harmonised, consistent chronology (**?**) to carry out a fair comparison. We apply our `bin_cor` and `plot_ts` functions and obtain the binned time series, which have 27 elements, and a temporal distance between elements of 1220 years. The binned time series show a relatively high level of autocorrelation, $\hat{\bar{a}}'_{\mathrm{MD04-2845}} = 0.85$ and $\hat{\bar{a}}'_{\mathrm{MD95-2039}} = 0.80$, and an estimated bias-corrected persistence values of $\hat{\bar{\tau}}_{\mathrm{MD04-2845}} = 3400$ years and $\bar{\tau}_{\mathrm{MD95-2039}} = 1300$ years. It can be observed from Figures 6 A and 6 B that the binned time series are roughly similar to the "primary" time series, although binning causes some information loss. This is due to the high degree of irregularity in the sampling of the "primary" time series, which makes it difficult to resample when the binned time series are built. In addition, information is lost because the length of the bin is dependent on the persistence and autocorrelation of the "primary" time series. Finally, Figures 6 C and 6 D show that the two pollen time series, presented as the primary and binned data, may be significantly correlated. This is discussed below.

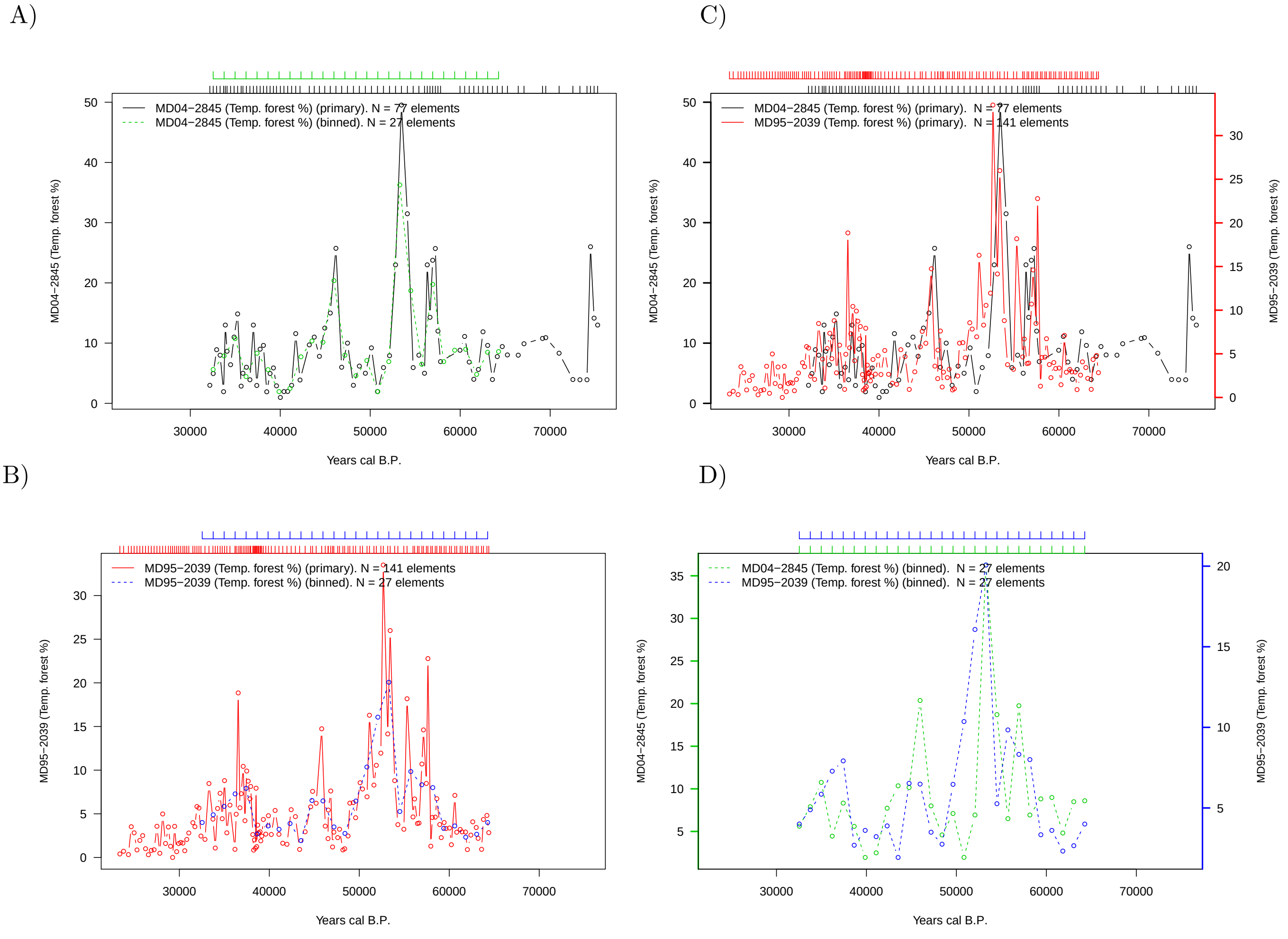


Figure 6: "Primary" (unevenly spaced) and binned pollen time series under analysis (**?**). The numbers of elements for both time series are provided in the legend. The autocorrelation and persistence values for the time series from site MD04-2845 are $\hat{\bar{a}}' = 0.85$ and $\hat{\tau} = 3400$ years, and those from site MD95-2039 are $\hat{\bar{a}}' = 0.80$ and $\tau = 1300$ years. The horizontal top axes indicate the sampling times for the plotted time series.

The code used to generate Figure 6 is as follows.

```
# Load the package
library(BINCOR)
library(pracma)

# Load the time series under analysis: Example 2 and Figure 6
data(MD04_2845_siteID31)
data(MD95_2039_siteID32)

# Compute the binned time series though our bin_cor function
bincor.tmp <- bin_cor(ID31.dat, ID32.dat, FLAGTAU=3, "salida_ACER_ABRUPT.tmp")
binnedts   <- bincor.tmp$Binned_time_series

# To avoid NA values
bin_ts1 <- na.omit(bincor.tmp$Binned_time_series[,1:2])
bin_ts2 <- na.omit(bincor.tmp$Binned_time_series[,c(1,3)])

# Applying our plot_ts function
# PDF format
plot_ts(ID31.dat, ID32.dat, bin_ts1, bin_ts2, "MD04-2845 (Temp. forest)",
"MD95-2039 (Temp. forest )", colts1=1, colts2=2, colbints1=3, colbints2=4,
device="pdf", Hpdf=6, Wpdf=9, resfig=300, ofilename="ts_ACER_ABRUPT")
```

The cross-correlation (CCF) analysis obtained with our `ccf_ts` function is shown in Figure 7. Before applying the `ccf_ts` function, a linear trend was removed from the binned time series by enabling the `rmltrd` option in `ccf_ts`, and then the residuals were used. The CCF reveals a high correlation ($r_{xy} = 0.53$) between the binned time series at lag 0. The high correlation between the pollen records from sites MD04-2845 and MD95-2039 reflects similar responses by vegetation to regional climate variability, particularly to changes in precipitation and temperature. However, the most noticeable result in our CCF analysis is that the maximum correlation ($\mathrm{r}_{xy} = 0.63$) is obtained at lag 1. At face value, this result suggests that pollen variability at site MD04-2845 leads that observed at site MD95-2039 by 1220 years. Nevertheless, these sites are located relatively close to each other and are in the same climate domain today, so it is difficult to envisage such a time difference in the response of vegetation (pollen) to rapid climatic changes in the past. The most plausible explanation for this out-of-phase relationship probably lies in the chronological uncertainties of the age models applied to these records. Despite best-efforts to harmonise the different time series in the ACER database using radiometric dating (?), the lack of ^{14}C dates for site MD95-2039 forced us to build the age model for this site by tuning the planktic foraminifera and GRIP ice core oxygen isotopic records (Roucoux et al., 2005). This tuning could affect the time series from site MD95-2039 and introduce unacknowledged chronological uncertainties (Blaauw, 2012; Hu et al., 2017). To summarise, with the present state of data quality we cannot rule out the idea that timescale uncertainties –rather than climate impact adaptation – caused the lag observed.

The code used to generate Figure 7 is the following.

```
# Load packages
library(BINCOR)
library(pracma)

# Load the time series under analysis: Example 2 and Figure 7 (ID31 vs. ID32)
data(MD04_2845_siteID31)
data(MD95_2039_siteID32)

# Compute the binned time series though our bin_cor function
bincor.tmp <- bin_cor(ID31.dat, ID32.dat, FLAGTAU=3, "salida_ACER_ABRUPT.tmp")
binnedts   <- bincor.tmp$Binned_time_series

# To avoid NA values
bin_ts1 <- na.omit(bincor.tmp$Binned_time_series[,1:2])
bin_ts2 <- na.omit(bincor.tmp$Binned_time_series[,c(1,3)])

# Applying our ccf_ts function
# PDF format
ccf_acf <- ccf_ts(bin_ts1, bin_ts2, RedL=TRUE, rmltrd="y", device="pdf", Hpdf=6,
Wpdf=9, resfig=300, ofilename="ccf_ID31_ID32_res")
```

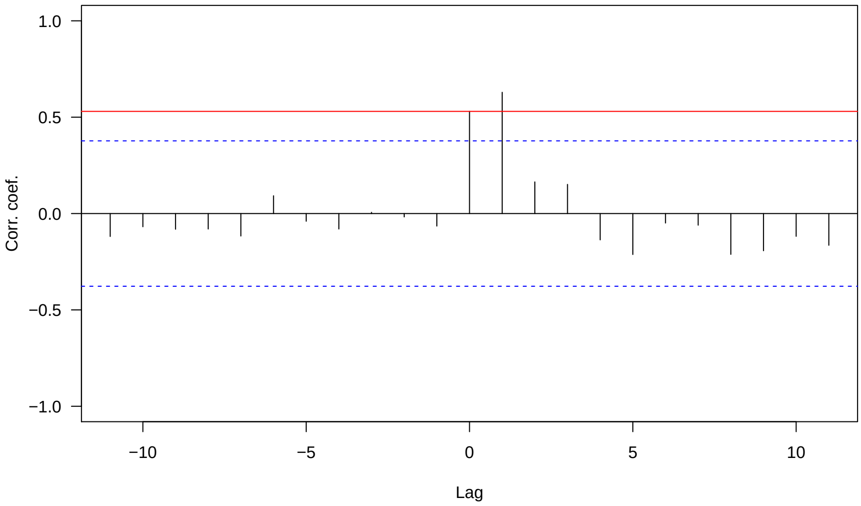


Figure 7: Cross-correlation for the residuals of the binned pollen time series from sites MD04-2845 and MD95-2039 (?). The CCF correlation coefficients at lag 0 and 1 are 0.53 and 0.63, respectively. The red line indicates the correlation coefficient for lag 0. Each lag is equivalent to 1220 years.

Summary

We present a computational package named **BINCOR** (BINned CORrelation) that can be used to estimate the correlation between two unevenly spaced climate time series which are not necessarily sampled at identical points in time, and between two evenly spaced time series which are not on the same time grid. **BINCOR** is based on a novel estimation approach proposed by Mudelsee (2010). This statistical technique requires the concept of nonzero persistence times, thus enabling mixing information to be recovered, even when the two timescales examined differ (Mudelsee, 2014). The package contains four functions (`bin_cor`, `cor_ts`, `ccf_ts` and `plot_ts`) with a number of parameters to obtain a high degree of flexibility in the analysis. **BINCOR** is programmed in `R` language and is available from the CRAN repository. The results when **BINCOR** s applied to real climate data sets suggest that the `R` package **BINCOR** performs and works properly in detecting relationships between instrumental and paleoclimate records.

Acknowledgements

JMPM was funded by a Basque Government post-doctoral fellowship. MM's work was supported by the European Commission via Marie Curie Initial Training Network LINC (project number 289447) under the Seventh Framework Programme. Thanks to Charo Sánchez for help to use the i2BASQUE HPC facilities, to the two anonymous reviewers and Editor (Olivia Lau) for their input and comments that have improved the quality of the manuscript. The authors thank the support of the computing infrastructure of the i2BASQUE (Basque Government) academic network. The persistence time estimation software is freely available via `http://www.climate-risk-analysis.com/software/`.

Bibliography

M. Blaauw. Out of tune: The dangers of aligning proxy archives. *Quaternary Science Reviews*, 36: 38–49, 2012. URL `https://doi.org/10.1016/j.quascirev.2010.11.012`. [p180]

H. W. Borchers. *pracma: Practical Numerical Math Functions*, 2015. URL `http://CRAN.R-project.org/package=pracma`. R package version 1.8.8. [p174]

P. Brohan, J. J. Kennedy, I. Harris, S. F. Tett, and P. D. Jones. Uncertainty estimates in regional and global observed temperature changes: A new data set from 1850. *Journal of Geophysical Research: Atmospheres*, 111(D12), 2006. URL `http://dx.doi.org/10.1029/2005JD006548`. [p176, 177]

A. Bunn, M. Korpela, F. Biondi, F. Campelo, P. Merian, F. Qeadan, C. Zang, A. Buras, J. Cecile, M. Mudelsee, and M. Schulz. *Dendrochronology Program Library in R*, 2015. URL http://CRAN.R-project.org/package=dplR. R package version 1.6.3. [p174]

W. Dansgaard, S. Johnsen, H. Clausen, D. Dahl-Jensen, N. Gundestrup, C. Hammer, C. Hvidberg, J. Steffensen, A. Sveinbjornsdottir, J. Jouzel, and G. Bond. Evidence for general instability of past climate from a 250-kyr ice-core record. *Nature*, 364(6434):218–220, 1993. URL http://dx.doi.org/10.1038/364218a0. [p178]

J. Emile-Geay. *Data Analysis in the Earth & Environmental Sciences*. Ed. Figshare, 2016. [p170]

D. B. Enfield and D. A. Mayer. Tropical Atlantic sea surface temperature variability and its relation to El Nino-Southern Oscillation. *Journal of Geophysical Research: Oceans*, 102(C1):929–945, 1997. URL http://dx.doi.org/10.1029/96JC03296. [p177]

E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based most similar trajectory search. In *2007 IEEE 23rd International Conference Data Engineering*, pages 816–825, 2007. URL http://dx.doi.org/10.1109/ICDE.2007.367927. [p177]

J. Garcia-Serrano, C. Cassou, H. Douville, A. Giannini, and F. J. Doblas-Reyes. Revisiting the ENSO teleconnection to the Tropical North Atlantic. *Journal of Climate*, 30(17):6945–6957, 2017. URL https://doi.org/10.1175/JCLI-D-16-0641.1. [p177]

D. L. Gilman, F. J. Fuglister, and J. M. Mitchell Jr. On the power spectrum of “red noise”. *Journal of the Atmospheric Sciences*, 20(2):182–184, 1963. URL https://doi.org/10.1175/1520-0469(1963)020<0182:OTPSON>2.0.CO;2. [p170]

L. Horowitz. The effects of spline interpolation on power spectral density. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 22(1):22–27, 1974. URL http://dx.doi.org/10.1109/TASSP.1974.1162536. [p170]

J. Hu, J. Emile-Geay, and J. Partin. Correlation-based interpretations of paleoclimate data–where statistics meet past climates. *Earth and Planetary Science Letters*, 459:362–371, 2017. URL https://doi.org/10.1016/j.epsl.2016.11.048. [p180]

M. E. Mann and J. M. Lees. Robust estimation of background noise and signal detection in climatic time series. *Climatic Change*, 33(3):409–445, 1996. URL https://doi.org/10.1007/BF00142586. [p170]

M. E. Mann, Z. Zhang, S. Rutherford, R. S. Bradley, M. K. Hughes, D. Shindell, C. Ammann, G. Faluvegi, and F. Ni. Global signatures and dynamical origins of the Little Ice Age and Medieval Climate Anomaly. *Science*, 326(5957):1256–1260, 2009. URL https://doi.org/10.1126/science.1177303. [p176, 177]

U. Mori, A. Mendiburu, and J. Lozano. TSdist: Distance measures for time series data. *R package version 3.4*, 2, 2015. URL http://CRAN.R-project.org/package=TSdist. [p177]

U. Mori, A. Mendiburu, and J. A. Lozano. Distance measures for time series in R: The TSdist package. *R Journal*, 8(2):451–459, 2016. [p177]

M. Mudelsee. TAUEST: A Computer Program for Estimating Persistence in Unevenly Spaced Weather/Climate Time Series. *Computers & Geosciences*, 28(1):69–72, 2002. URL https://doi.org/10.1016/S0098-3004(01)00041-3. [p170, 171, 174]

M. Mudelsee. Estimating Pearson’s correlation coefficient with bootstrap confidence interval from serially dependent time series. *Mathematical Geology*, 35(6):651–665, 2003. URL https://doi.org/10.1023/B:MATG.0000002982.52104.02. [p170]

M. Mudelsee. *Climate Time Series Analysis: Classical Statistical and Bootstrap Methods*. Springer-Verlag, 2010. ISBN 9048194814. [p170, 171, 172, 181]

M. Mudelsee. *Climate Time Series Analysis: Classical Statistical and Bootstrap Methods*. Springer-Verlag, Second edition, 2014. ISBN 9048194814. [p170, 171, 172, 173, 174, 181]

K. Olafsdottir and M. Mudelsee. More accurate, calibrated bootstrap confidence intervals for estimating the correlation between two time series. *Mathematical Geosciences*, 46(4):411–427, 2014. URL https://doi.org/10.1007/s11004-014-9523-4. [p170]

K. Rehfeld and G. Bedartha. *NESTOOLBOX – Toolbox for the Analysis of Non-Equidistantly Sampled Time Series*, 2014. URL http://tocsy.pik-potsdam.de/nest.php. Matlab/Octave, version 1.01. [p170]

K. Rehfeld and J. Kurths. Similarity estimators for irregular and age-uncertain time series. *Climate of the Past*, 10(1):107–122, 2014. URL https://doi.org/10.5194/cp-10-107-2014. [p170]

K. Rehfeld, N. Marwan, J. Heitzig, and J. Kurths. Comparison of correlation analysis techniques for irregularly sampled time series. *Nonlinear Processes in Geophysics*, 18(3):389–404, 2011. URL https://doi.org/10.5194/npg-18-389-2011. [p170]

J. Roberts, M. Curran, S. Poynter, A. Moy, T. van Ommen, T. Vance, C. Tozer, F. S. Graham, D. A. Young, C. Plummer, J. Pedro, D. Blankenship, and M. Siegert. Correlation confidence limits for unevenly sampled data. *Computers & Geosciences*, 104:120–124, 2017. URL https://doi.org/10.1016/j.cageo.2016.09.011. [p170]

P. Robinson. Estimation of a time series model from unequally spaced data. *Stochastic Processes and their Applications*, 6(1):9–24, 1977. URL https://doi.org/10.1016/0304-4149(77)90013-8. [p170, 171]

K. Roucoux, L. De Abreu, N. Shackleton, and P. Tzedakis. The response of NW Iberian vegetation to North Atlantic climate oscillations during the last 65 kyr. *Quaternary Science Reviews*, 24(14): 1637–1653, 2005. URL https://doi.org/10.1016/j.quascirev.2004.08.022. [p179, 180]

M. F. Sanchez Goni, A. Landais, W. J. Fletcher, F. Naughton, S. Desprat, and J. Duprat. Contrasting impacts of Dansgaard–Oeschger events over a western European latitudinal transect modulated by orbital parameters. *Quaternary Science Reviews*, 27(11):1136–1151, 2008. URL https://doi.org/10.1016/j.quascirev.2008.03.003. [p179]

G. P. Weedon. *Time-Series Analysis and Cyclostratigraphy: Examining Stratigraphic Records of Environmental Cycles*. Cambridge Univ Press, Cambridge, 2003. [p170]

D. S. Wilks. *Statistical Methods in the Atmospheric Sciences*, volume 100. Academic press, 2011. [p170]

E. W. Wolff, S. P. Harrison, R. Knutti, M. F. Sanchez Goni, O. Wild, A.-L. Daniau, V. Masson-Delmotte, I. C. Prentice, and R. Spahni. How has climate responded to natural perturbations? In S. E. Cornell, I. C. Prentice, J. I. House, and C. J. Downy, editors, *Understanding the Earth System : Global Change Science for Application*, pages 72–101. Cambridge University Press, 2012. [p178]

Josue M. Polanco-Martinez
Basque Centre for Climate Change - BC3
Sede Building 1, 1st floor - Scientific Campus of the UPV/EHU
48940 Leioa
&
Econometrics Research Group - Institute of Public Economics
University of the Basque Country
48015 Bilbao
SPAIN
josue.m.polanco@gmail.com, josue.polanco@bc3research.org

Martin A. Medina-Elizalde
Dept. of Geosciences, Auburn University
2050 Beard Eaves Coliseum, 36849 Auburn, AL
USA
mam0199@auburn.edu

Maria F. Sanchez Goni
Ecole Pratique des Hautes Etudes (EPHE), PSL University & UMR EPOC CNRS 5805, University of Bordeaux
Allee Geoffroy St Hilair, 33615 Pessac
FRANCE
maria.sanchez-goni@u-bordeaux.fr

Manfred Mudelsee
Climate Risk Analysis, 37581 Bad Gandersheim
&
Alfred Wegener Institute (AWI) - Helmholtz Centre for Polar and Marine Research
27570 Bremerhaven
GERMANY
mudelsee@climate-risk-analysis.com

Optimization Routines for Enforcing One-to-One Matches in Record Linkage Problems

by Diego Moretti, Luca Valentino and Tiziana Tuoto

Abstract Record linkage aims at quickly and accurately identifying if two records represent the same real world entity. In many applications, we are interested in restricting the linkage results to "1 to 1" links, that is a single record does not appear more than once in the output. This can be dealt with the transport algorithm. The optimization problem, however, grows quadratically in the size of the input, quickly becoming untreatable for cases with a few thousand records. This paper compares different solutions, provided by some R packages for linear programming solvers. The comparison is done in terms of memory usage and execution time. The aim is to overcome the current implementation in the toolkit RELAIS, specifically developed for record linkage problems. The results highlight improvements beyond expectations. In fact the tested solutions allow successfully executing the "1 to 1" reduction for large size datasets up to the largest sample surveys at National Statistical Institutes.

Introduction

Record linkage is a process that aims at quickly and accurately identifying if two (or more) records represent the same real world entity. A record linkage project can be performed for different purposes and the variety of the uses makes it a powerful instrument to support decisions in large commercial organizations and government institutions. In official statistics, the field in which this work is developed, the combined use of statistical survey, administrative data and other new data sources (the so-called Big Data) is largely widespread and strongly stimulates the investigation of new methodologies and instruments to deal with record linkage projects.

This work is developed in the field of official statistics: in this area, the combined use of statistical surveys, administrative data and other new data sources (the so-called Big Data) is largely widespread and strongly stimulates the investigation of new methodologies and instruments to deal with record linkage projects.

Since the earliest contributions to modern record linkage (Newcombe et al., 1959; Fellegi and Sunter, 1969) there has been a proliferation of different approaches, that make use also of techniques based on data mining, machine learning, soft computing and others. Record linkage can be seen as a complex process consisting of several distinct phases involving different knowledge areas.

A record linkage process becomes trivial if the input files share a common error-free unit identifier, but it can be quite complex when common identifiers are error prone or no common identifier exists at all and one has to rely on shared covariates, as is actually the case with real data (Hernandez and Stolfo, 1998).

To effectively face the record linkage problem, the Italian National Statistical Institute (Istat) designed and developed a toolkit, RELAIS, that is the result of many experiences gained performing several integration processes in different contexts (Cibella et al., 2012). This software is configured as an open source project with the aim of facing the record linkage complexity by decomposing the whole problem in its constituting phases and dynamically adopting the most appropriate technique for each step. It is therefore possible to define the most suitable strategy for each linkage problem depending on the specific data requirements (Cibella et al., 2007). Software and related documentation can be downloaded from the Istat website (ISTAT, 2015) and the European website for sharing and reusing solutions for public administrations (JOINUP, 2015)

The "Selection of unique links" is a step of the record linkage that has not been thoroughly investigated by the statistical and IT communities. This work focuses on it. Traditionally, Jaro (1989) suggested to solve it with an optimization procedure, the simplex algorithm. The solution is constrained by the size of the data processed: the optimization algorithm must solve an input matrix that grows quadratically in the input files size. In this work, we analyse the solutions already proposed and implemented and compare them with the alternatives available in the R CRAN. It is worth noting that our intention is not to provide a comparison of algorithms or optimization solutions in a general context, our goal is focused on the specific solution of the record linkage phase.

The work aims at presenting alternative algorithms and their implementations for selecting unique links in record linkage problems as they are faced in official statistics. To ensure accessibility and reuse, we consider R as the environment for running the alternative algorithms, although, due to the algorithm complexity, R often only provides a wrapper for other more efficient programming languages. The comparison is presented throughout real life examples, derived from long experiences in data linkage at NSIs. The provided experiments are designed to account for the several features and characteristics of real world data, in terms of size, accuracy of input data and expected output. The comparison is motivated by the need for improving the current optimizer so as to process larger datasets in a short time, without losing the linkage efficacy measured in terms of precision and recall.

The work is organised as follows: in next section we shortly introduce the most common formalization for record linkage problems (Fellegi and Sunter, 1969), and specifically the optimization problem we need to solve. We describe the current module for the selection of unique links in RELAIS. Then we propose a short section about alternative algorithms. We compare the alternatives throughout several use cases, based on applications of record linkage in NSIs. Once the best implementation has been identified, further enhancements are highlighted. Finally, in the last section, we resume results and concluding remarks.

The record linkage and the optimization phase

Formalization of the probabilistic record linkage decision model

Fellegi and Sunter (1969) firstly defined record linkage as a decision problem: given two lists, say A and B, of size n_A and n_B, the linkage process can be viewed as a classification problem where the pairs in the Cartesian product $\Omega = ((a,b), a \in A \quad \text{and } b \in B)$ have to be assigned into two subsets M and U, independent and mutually exclusive, such that M is the set of Matches ($a = b$) while U is the set of Non-Matches ($a \neq b$).

In order to assign the pairs (a,b) either to the set M or U, k common attributes (the matching variables) are compared. The statistical model for record linkage is built upon the so called comparison vectors γ

$$\gamma_{(ab)} = \left(\gamma_{(ab)1}, \ldots, \gamma_{(ab)k}\right)$$

where, in the simplest setting,

$$\gamma_{(ab)j} = \begin{cases} 1 & \gamma_{(ab)j} = \gamma_{(ab)j} \\ \\ 0 & \gamma_{(ab)j} \neq \gamma_{(ab)j} \end{cases}, \qquad j = 1, \ldots, k.$$

The comparison vectors $\gamma_{(ab)}$ are usually assumed to be independent and identically distributed random vectors with distribution given by a mixture of two different (unobserved) distributions: the former represents the pairs (a,b) which actually are the same unit, the m distribution; the latter represents the pairs (a,b) which actually belong to different units, the u distribution. The mixture weight p represents the marginal probability that a random pair of records (a,b) is a true match, i.e. it may be interpreted as the percentage of overlapping of the two data sets.

The estimation of the mixture weight p and the two distributions m and u requires the use of iterative methods, generally the EM algorithm or its generalizations. Also, in the standard setting, the matching variables are assumed independent of each other. Several extensions of this basic set-up have been proposed, mainly by introducing potential interactions among key variables, see for example (Winkler, 1995; Larsen and Rubin, 2001).

Once the two distributions $m(\gamma_{(ab)})$ and $u(\gamma_{(ab)})$ are estimated, a given pair should be allocated to M or U on the basis of the likelihood ratio, also called composite matching weight:

$$r_{(ab)} = \frac{\hat{m}(\gamma_{(ab)})}{\hat{u}(\gamma_{(ab)})}.$$

It is also possible to assign the pairs on the basis of a posterior probability that the pair is a match:

$$r^*_{(ab)} = \frac{\hat{p} * \hat{m}(\gamma_{(ab)})}{\hat{p} * \hat{m}(\gamma_{(ab)}) + (1 - \hat{p})\,\hat{u}(\gamma_{(ab)})}.$$

In general, we declare as matches the pairs of records with likelihood ratio r - or posterior probability r^* - above a fixed threshold. In practice, the choice of the threshold can be problematic, as illustrated, for example, in Belin and Rubin (1995). In this context, optimization techniques may be helpful to solve the multiple matches issue, that is the possibility that a single unit in data set A is linked with more than one unit in data set B. This will be discussed in the next subsection.

The optimization problem in record linkage

In several applications, the record linkage aims at recognizing exactly and univocally the same units and to establish only unique or "1 to 1" links. In other words, the linkage result must satisfy the constraint that one record on file A can be assigned to one and only one record on file B, and vice-versa. This kind of applications requires several constraints and is a complex problem of optimization. For instance, when comparing Population Census with Post Enumeration Survey, one is interested in "1 to 1" links in order to recognize people caught in the two occasions; moreover when linking tax register and income survey again "1 to 1" links are the expected output in order to enrich the available information in the input sources. On the other side, when hospital admissions forms are linked to a patient list, multiple linkages ("n to 1" links) are admissible. Finally "n to m" links are expected when linking, for instance, Employees and Employers files.

To achieve "1 to 1" links in the Fellegi-Sunter setting that considers the cross product of possible pairs, Jaro (1989) suggested to formulate it as a linear programming problem: once the matching weight r is assigned to each pair, the identification of "1 to 1" links can be solved maximizing the objective function given by the sum of weights for the link pairs, under the constraints given by the fact that each unit of A can be linked at most with one unit of B and vice-versa. According to Jaro (1989), this is a degenerate transportation problem, and the use of such a linear programming model represents an advance with respect to other ad hoc assignment methods. In order to formulate the problem, let r_{ab} be the matrix containing the composite weights for all pairs, the maximizing function is:

$$Z = \sum_{a=1}^{n_A} \sum_{b=1}^{n_B} r_{ab} X_{ab}$$

under the $n_A + n_B$ constrains:

$$\sum_{a=1}^{n_A} X_{ab} \leqslant 1 \qquad b = 1, 2, \ldots, n_B$$

$$\sum_{b=1}^{n_B} X_{ab} \leqslant 1 \qquad a = 1, 2, \ldots, n_A$$

where X_{ab} is a matrix with entries corresponding to indicator variables, equal to 1 if record a in A is linked with record b in B .

It is worth noting that the size of the X_{ab} matrix increases quadratically in the size of the input files, with dramatic effects on the memory usage and computation time. For instance, when both input files contain 100 records, the X matrix contains 10 thousand cells; when both input files consist of 1000 records, the matrix becomes 1 million entries. Traditionally, in record linkage problems, the computation issues related to the input size are managed via the so called blocking procedures (Gu et al., 2003; Baxter et al., 2003); in the optimization step, the complexity related to the input size is also managed restricting to the "most likely" pairs. Indeed, the matching weight r represents the ratio between the likelihood that a pair belongs to the set of Matches and the likelihood that the pair belongs to the set of Non-Matches. Similarly, r^* represents the posterior probability that a pair belongs to the set of Matches. It is clear that for most pairs, the matching weights r_{ab} and r^*_{ab} take very small values (close to zero), since considering two input files of 1000 records and the 1 million pairs they generate, at most 1000 pairs will be true matches with expected high values of r_{ab} and r^*_{ab}. So, a common practice for solving the "1 to 1" links is to reduce the complexity by eliminating from the optimization analysis the pairs with a value of r_{ab} (similarly, r^*_{ab}) below a certain threshold. A common choice of the threshold for r_{ab} is 1, meaning that we disregard the pairs for which the likelihood of belonging to M is lower than the likelihood of belonging to U. For r^*_{ab}, the most proper choice seems 0.5, as it is a posterior probability. The role of r_{ab} and r^*_{ab} will be further discussed in the experimental section.

It is worth mentioning that in the Bayesian approach to record linkage, the "1 to 1" constraint is solved directly in the model rather than in an ex-post adjustment (Tancredi and Liseo, 2013; Stoerts et al., 2017; Sadinle, 2017); however, to the best of our knowledge, the Bayesian record linkage is still affected by a certain lack of scalability, so we do not consider it in this analysis.

The Relais toolkit

The abovementioned Relais is a toolkit developed and used in Istat and in other NSIs to face record linkage problems. It is implemented in two programming languages, Java and R; moreover, the relational database MySql is used to manage datasets and temporary results. The R language is used for the key statistical phases of Relais: the estimation of the parameters p, m, and u of the probabilistic decision model, based on Fellegi-Sunter approach (Fellegi and Sunter, 1969), and the optimization algorithm to obtain the "1 to 1" linkage results.

The "1 to 1" reduction in Relais

The "1 to 1" reduction in RELAIS uses the linear programming problem approach proposed by Jaro (1989) and defined in the previous section. The R module uses the **LpSolve** package. In the following, the core of the current R source:

```
# pairs is the output of the decision model
# colnames(pairs) <- c("a", "b", "gamma", "r", "r*")


# command1: application of a preliminary filter to the input data
filtered=pairs[pairs[,5]>0.5,]

# command2: input preprocessing
# counting of unique identifiers of records
nA= length(unique(filtered[,1]))
nB= length(unique(filtered[,2]))
A=cbind(a=unique(filtered[,1]),A=1:nA)
B=cbind(b=unique(filtered[,2]),B=1:nB)
filtered =merge(B, filtered)
filtered =merge(A, filtered)
dat=t(filtered)

# command3: preparing constraint matrix
constr=array(rep(0,(nA+nB)*ncol(dat)), dim=c(nA+nB,ncol(dat)))
p=rbind(matrix(rep(dat[2,],nA),c(nA,ncol(constr)),byrow=TRUE),
        matrix(rep(as.numeric(dat[4,])+nA,nB),c(nB,ncol(constr)),byrow=TRUE))
constr [as.numeric(p)==row(constr)]=1

# command4: preparing other LP parameters
diseq=rep('<=',nA+nB)
ones=rep(1,nA+nB)
# target function
coeff=dat[6,]

# command5: LP execution
library("lpSolve")
ret=lp ("max", coeff, constr, diseq, ones)
# preparing the reduced set of pairs
reduc <- t(dat[,ret$solution>0.9])
```

The command1 is the preliminary filter useful to reduce the size of the input pairs: each pair is an entry of the constraint matrix. The filter is $r^* \geqslant 0.5$, as explained at the end of the previous section. Despite this simplification, the method may not work when processing medium/large sized datasets, i.e. when the input files A and B consist of more than 5000 records. In this case, the Relais tool offers an alternative algorithm (the greedy one) which, however, does not guarantee an optimal result. The size of the datasets that are treatable with the procedure depends on several causes. First of all, it depends on the workstation system 32-bit o 64-bit for Windows platform. Other relevant parameters are the size of the RAM, the version of the Operating System, the version of R, others software running on the workstation, the processor, etc. As shown in the next paragraph, we investigated the limits of this algorithm in two typical PC configurations (32-bit and 64-bit), furthermore we propose some improvements aimed at increasing its efficiency and performances.

Algorithms and R packages for LP solver

A recent analysis and comparison of different algorithms for linear programming is in Gearhart et al. (2013). This work is an inspiring starting point for our analysis aimed at investigating the best freely available open-source implementation in terms of performance and memory usage. Gearhart et al. (2013) compare four different open-source solvers facing a collection of linear programming problems; Gearhart et al. (2013) also consider the IBM ILOG CPLEX Optimizer (CPLEX), an industrial standard.

In this work, we compare linear programming solvers with the specific purpose of optimizing the identification of "1 to 1" links: to this end, we compare the current implementation available in Relais, the **lpSolve** (Berkelaar and others, 2013) R package, with the **Rglpk** (Theussl and Hornik, 2014) and **ROI.plugin.clp** (Thieurmel, 2017) R packages. These two R packages are wrappers of C libraries, corresponding to the two methodologies, GLPK and COIN-OR respectively. The comparison of these packages in other contexts, with different optimization problems, is out of the scope of this work. At a very first stage, we also considered the **intpoint** (del Rio, 2012) R package, that has been discarded because of the low performance in memory management compared to the previous ones. We also developed a Java procedure that implements the Hungarian Method Karmarkar (Karmarkar, 1984) but it was discarded because it did not bring improvements.

The comparison of the proposed solvers is influenced by the several configuration parameters of the personal computers, the specific hand code for the input preparation, and other characteristics that have been fixed in the experimental settings, described in the following section.

Experiments

In this paragraph, we resume the experiments about the "1 to 1" reduction procedures with the aim of measuring their execution time and their ability to handle large size data. Moreover, some upgrades of the code are proposed and described in detail: the upgrades are evaluated by comparing their performances with the current version. As previously specified, the current Relais procedure successfully solves optimization problems when the input files are smaller than 5000 records each. We investigate improvements with the first objective of enlarging these sizes; in the set of solutions which enable to manage larger datasets, we evaluate the best performances in terms of execution time. As already mentioned, Relais also proposes another method, the greedy one, but this is not compared with the optimization algorithms because it follows a different rationale not aimed at global optimization of the result. In addition, currently in Relais the greedy algorithm is not implemented as an R module. In short, we have observed that when the greedy algorithm finds links other than those of the optimal algorithms, these links are not correct; however, this rarely happens, about 1 in 1000 proposed pairs.

Experiment setup

For the comparison of the algorithms, we used two typical PC configurations, the 32-bit and 64-bit R respectively. Details on the PC configurations are shown in table 1.

Config.	OS	System	Processor	RAM	R version
32-bit	Windows 7	32-bit	Pentium Dual-Core 2.7 Ghz	4 Gb	3.4.0
64-bit	Windows 7	64-bit	Intel 3.5 Ghz	16 Gb	3.4.2

Table 1: Experiment PC configurations

To evaluate the performances in terms of time and memory usage, we used ten different linkage exercises summarized in table 2. Each exercise is composed by two datasets to integrate, the entities object of the linkage can be people or companies. The first seven problems are quite standard, i.e. the size of the two datasets and the number the matches are balanced; on the other hand, the last three exercises are less common, i.e. the size of datasets are lopsided or there are few matches. In the first column of table (Exercise), we mark the exercise on the basis of the size of the datasets, with "L" as suffix for lopsided exercises; this mark is also used in the next tables. The datasets reproduce real data; the linking variables are the true ones either have been artificially generated mimicking true data, as reported in the second column. The size of datasets and the number of real matches vary across the exercises, as shown in the last two columns of table 2.

Exercise	Linking variables	Entity	Datasets size	True matches
1K	Real	People	1,165x1,165	1,141
4K	Artificial	Companies	4,012x3,988	3,187
5K	Real	People	5,851x5,748	5,543
8K	Real	People	8,518x7,596	6,893
15K	Artificial	Companies	14,998x15,003	11,281
25K	Real	People	25,343x24,613	24,043
40K	Artificial	Companies	40,048x39,952	36,033
10KL	Artificial	Companies	9,963x1,018	1,015
20KL	Artificial	Companies	658x20,052	625
55KL	Artificial	Companies	55,210x5,083	2,189

Table 2: Experiment datasets

Experiments report

The current procedure encounters two critical phases, in which the memory used risks exceeding the memory available for the R task, respectively commands 3 (preparing constraint matrix) and 5 (LP execution). In both cases, the issue is represented by the size of the constraint matrix X_{ab}. As mentioned before, when the sizes of the input files do not allow the execution of the R commands, currently in Relais the users are suggested to apply greedy techniques.

So, firstly we focused on modifying command 3 to overcome the memory problem. The most promising solution is to reformulate the constraint matrix as a vector of constraints, where only the non-zero values of matrix X_{ab} appear. This structure requires much less memory than the previous solution, especially when the size of the inputs increases. The `lp` function of the **lpSolve** package admits the `dense.const` parameter which allows us to use a vector instead of a matrix to express the constraints for our maximization problems. In this case commands 3 and 5 become as follows:

```
# command3.1: preparing constraint vector
constr.vec <- matrix(c(as.numeric(dat[2,]), as.numeric(dat[4,])+nA,
                       rep(1:ncol(dat),2), rep(1,(2*ncol(dat)))), ncol=3)


# command5.1: LP execution
```

```
library("lpSolve")
ret=lp ("max", coeff, , diseq, ones, dense.const=constr.vec)
# preparing the reduced set of pairs
reduc <- t(dat[,ret$solution>0.9])
```

Table 3 shows the results of the 10 exercises, in the two options. The column 'Matrix' shows the execution time of the current R module with constraints expressed by a matrix. The column 'Vector' shows the execution time of the R module with commands 3.1 and 5.1 in place of commands 3 and 5, i.e. constraints expressed by a vector. The 'KO' entry means that the execution is aborted due to memory error.

The results shown by table 3 fully meet our expectations. In fact, the current code can process only 1K, 4K and lopsided datasets using the 32-bit configuration and reaches up to 15K datasets in the 64-bit configuration. The new code can process all datasets even in the worst configuration. In addition, there is also a great improvement in the execution times.

In a second phase, we concentrated our efforts on evaluating the use of the alternative packages identified and abovementioned, i.e. **ROI.plugin.clp** and **Rglpk**.

The two solvers accept the constraint parameter as vector, using the structure `simple_triplet_matrix` defined in the **slam** (Hornik et al., 2014) package.

Then, command 3 becomes

```
# command3.2: preparing constraint parameter
constr <- simple_triplet_matrix(c(as.numeric(dat[2,]),as.numeric(dat[4,])+n),
                rep(1:ncol(dat),2), rep(1,(2*ncol(dat))), nrow=(n+m), ncol=ncol(dat))
```

In the case of **ROI.plugin.clp** solver, command 5 becomes:

```
# command5.2: LP execution
LP <- ROI::OP(as.numeric(coeff), ROI::L_constraint(L = constr, dir = diseq, rhs = ones),
                 max = TRUE)
ret <- ROI::ROI_solve(x = LP, solver = "clp")
# preparing the reduced set of pairs
reduc <- t(dat[,ret$solution>0.9])
```

In the case of **Rglpk** solver, command 5 is:

```
# command5.3: LP execution
ret <- Rglpk_solve_LP(coeff,constrv,diseq,ones,types="I",max=TRUE)
# preparing the reduced set of pairs
reduc <- t(dat[,ret$solution>0.9])
```

Table 4 compares the execution times in seconds, required for the complete execution of the R module using the three different packages:

From table 4, the first remark underlines that the use of constraints as vectors allows all packages to manage all the tested datasets even in the worst configuration. Moreover, for this type of problem, it is quite clear that the **ROI.plugin.clp** solver guarantees the best performances, overtaking the other packages especially with large datasets. The gain with **ROI.plugin.clp** is more evident in the 32-bit configuration. The second best performer is **Rglpk**, however it is at least one order of magnitude slower than the previous one. Finally, **lpSolve** presents the worst performances, particularly in the 32-bit configuration, whilst the differences with **Rglpk** are reduced in the case of the 64-bit configuration. These results are substantially valid for both balanced data sets and lopsided data sets. We note that in lopsided cases all solutions seem to perform better. In fact, the complexity of the problem is mainly due to the number of pairs proposed to the reduction algorithm rather than to the size of the input data. In typical "1 to 1" record linkage projects, the number of pairs depends more on the size of the smallest dataset than on the largest one. The **Roi.plugin.clp** greatly improves in the case of sparse matrices; the difference with **Rglpk** is reduced with lopsided data. In our opinion, a large part of the improvements with **Roi.plugin.clp** is due to the COIN-OR Optimization algorithm written in C and the use of a good wrapper for R; there is a part of the code where the R language communicates with a buffer with a procedure compiled in C language. **Rglpk** is also based on an algorithm written in C, but this is probably less powerful in this type of problem. Instead, **lpSolve** is written entirely in R, it makes an intensive use of the memory, and it is therefore less efficient than the other tested packages.

Further improvements and concluding remarks

The vector representation of the constraints and the adoption of the **ROI.plugin.clp** package provide the expected important improvements in the module's performance, i.e. the size of the managed pairs, memory usage and the execution speed. The achieved results encourage us to overcome the current preliminary filter that allows processing only pairs (ab) with posterior probability $r^*_{ab} > 0.5$, see command 1. This filter was included to overcome the previous performance issue in terms of memory usage, however, from a statistical perspective, it risks compromising the results of the statistical model by deterministically removing possible matches. Obviously, the more restrictive the filter, the more likely it is that possible links are missing. So, to partially reduce this drawback, we intend to apply the filter $r_{ab} > 1$ instead of $r^*_{ab} > 0.5$. In fact, $r_{ab} > 1$ is less restrictive than $r^*_{ab} > 0.5$. The filters $r_{ab} > 1$ and $r^*_{ab} > 0.5$ are "theoretically" justified, as they represent, respectively, the likelihood ratio and the posterior probability of a pair to be a match.

We resume the 10 exercises proposed in the previous section and apply the "1 to 1" reduction with the proposed preliminary filters. In each run we measure the execution time and the quality of the output in terms of recall. The recall is a standard quality measure used in record linkage that compares the number of true matches identified by the linkage process with the number of true matches. The closer the value is to 1, the more effective the procedure is. The following table 5 compares the obtained measures. We only report the values obtained from the **ROI.plugin.clp** package in the 32-bit configuration. As above clarified, our first interest is to evaluate the improvements in the recall and to verify that memory problems do not recur.

Table 5 firstly shows that, with the filter $r_{ab} > 1$, there is always a small improvement in the recall. Secondly, the algorithm is always successfully executable and the execution time doesn't generally increase significantly. The exercise 40K is an exception: the execution time goes from 1.7 to 7.3 seconds. In this case, we have verified that the modification of the preliminary filter has a significant impact: in fact, the number of considered pairs increases from about 26,000 to over 100,000, with remarkable effects on the recall. The 40K experiment proves, on one hand, that this adjustment can have important positive effects for the linkage process and, on the other hand, ensures that the procedure can be successfully executed also with a large number of input pairs.

```
# command1: application of a preliminary filter to the input data
filtered=pairs[pairs[,6]>1,]

# command2: input preprocessing
# counting of unique identifiers of records
n= length(unique(filtered[,1]))
m= length(unique(filtered[,2]))
A=cbind(I=unique(filtered[,1]),A=1:n)
B=cbind(J=unique(filtered[,2]),B=1:m)
filtered =merge(B, filtered)
filtered =merge(A, filtered)
dat=t(filtered)

# command3: preparing constraint parameter
constr <- simple_triplet_matrix(c(as.numeric(dat[2,]),as.numeric(dat[4,])+n), rep(1:ncol(dat),2),
                                 rep(1,(2*ncol(dat))), nrow=(n+m), ncol=ncol(dat))

# command4: preparing other LP parameters
diseq=rep('<=',m+n)
ones=rep(1,m+n)
# coefficients for the target function
coeff=dat[6,]

# command5: LP execution
LP <- ROI::OP(as.numeric(coeff),
                         ROI::L_constraint(L = constr,  dir = diseq,  rhs = ones), max = TRUE)
ret <- ROI::ROI_solve(x = LP, solver = "clp")
# prepare the reduced set of pairs
reduc <- t(dat[,ret$solution>0.9])
```

To conclude, the main advantages of the proposed improvements relate to the successful execution of the "1 to 1" reduction for large datasets, as well as the gain in the execution time. The above reported new code for the implementation of the optimization step will replace the previous one in the new release of Relais. We are mostly satisfied with the achieved improvements, as they will simplify and enhance the future linkage strategies. For instance, with the improvements studied in this paper, we will be able to easily manage the linkage between statistical registers and social sample surveys. In particular, we will be able to manage the "1 to 1" optimization step for the current social surveys involving about 40,000 units, as in the experiment 40K. Moreover, we will also manage within its main spatial domains, the largest Italian sample survey, the Labour Force Survey (LFS). In fact, currently, the Italian LFS involves up to 25,000 units, in the NUTS2 domain.

Bibliography

R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. *ACM SIGKDD*, 3:25–27, 2003. [p187]

T. R. Belin and D. B. Rubin. A method for calibrating false- match rates in record linkage. *Journal of the American Statistical Association*, (90):694–707, 1995. [p186]

M. Berkelaar and others. *lpSolve: Interface to Lpsolve v. 5.5 to Solve Linear Integer Programs*, 2013. URL http://CRAN.R-project.org/package=lpSolve. R package version 5.6.7. [p189]

N. Cibella, M. Fortini, R. Spina, M. Scannapieco, L. Tosco, and T. Tuoto. Relais: An open source toolkit for record linkage. *Rivista di Statistica Ufficiale*, (2-3):55–68, 2007. [p185]

N. Cibella, M. Scannapieco, L. Tosco, T. Tuoto, and L. Valentino. Record linkage with relais: Experiences and challenges. *Estadistica espanola*, 54(179):311–328, 2012. [p185]

A. Q. del Rio. *Intpoint: Linear Programming Solver by the Interior Point Method and Graphically (Two Dimensions)*, 2012. URL http://CRAN.R-project.org/package=intpoint. R package version 1.0. [p189]

I. Fellegi and A. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, (64):1183–1210, 1969. [p185, 186, 188]

J. L. Gearhart, K. L. Adair, J. D. Detry Richard J.and Durfee, K. A. Jones, and N. Martin. Comparison of open-source linear programming solvers. *Sandia National Laboratories Report*, pages 4–62, 2013. [p189]

L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: Current practice and future directions. *CSIRO Mathematical and Information Sciences Technical Report*, 3(83), 2003. [p187]

M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge purge problem. *Journal Data Mining and Knowledge Discovery*, 2(1):9–37, 1998. [p185]

K. Hornik, D. Meyer, and C. Buchta. *Slam: Sparse Lightweight Arrays and Matrices*, 2014. URL http://CRAN.R-project.org/package=slam. R package version 0.1-32. [p191]

ISTAT. Download RELAIS on Istat Website, 2015. URL https://www.istat.it/en/methods-and-tools/methods-and-it-tools/process/processing-tools/relais. [p185]

M. A. Jaro. Advances in record linkage methodologies as applied to matching the 1985 census of tampa, florida. *Journal of American Statistical Society*, 84(84):414–420, 1989. [p185, 187, 188]

JOINUP. Download RELAIS on Joinup Website, 2015. URL https://joinup.ec.europa.eu/solution/relais-record-linkage-istat. [p185]

N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4): 373–395, 1984. [p189]

M. D. Larsen and D. B. Rubin. Iterative automated record linkage using mixture models. *Journal of the American Statistical Association*, (79):32–41, 2001. [p186]

H. Newcombe, J. Kennedy, S. Axford, and A. James. Automatic linkage of vital records. *Science*, 130(130):954–959, 1959. [p185]

M. Sadinle. Bayesian estimation of bipartite matchings for record linkage. *Journal of the American Statistical Association*, 112(518):600–612, 2017. [p187]

R. Stoerts, R. Hall, and S. Fienberg. A bayesian approach to graphical record linkage and deduplication. *Journal of the American Statistical Association*, pages 1660–1672, 2017. [p187]

A. Tancredi and B. Liseo. A hierarchical bayesian approach to record linkage and population size problems. *The Annals of Applied Statistics*, pages 1553–1585, 2013. [p187]

S. Theussl and K. Hornik. *Rglpk: R/GNU Linear Programming Kit Interface*, 2014. URL http://CRAN.R-project.org/package=Rglpk. R package version 0.6-0. [p189]

B. Thieurmel. *ROI.plugin.clp: 'Clp (Coin-or Linear Programming)' Plugin for the 'R' Optimization Interface*, 2017. URL https://CRAN.R-project.org/package=ROI.plugin.clp. R package version 0.4. [p189]

W. E. Winkler. Matching and record linkage. *Business Survey Methods (B.G. Cox et al, ed.)*, pages 355–384, 1995. [p186]

Diego Moretti
Istat - Italian National Institute of Statistics
via C. Balbo 16
Italy
dimorett@istat.it

Luca Valentino
Istat - Italian National Institute of Statistics
via C. Balbo 16
Italy
luvalent@istat.it

Tiziana Tuoto
Istat - Italian National Institute of Statistics
via C. Balbo 16
Italy
tuoto@istat.it

Config.	RAM	Exercise	Matrix	Vector
32-bit	4 Gb	1K	4.3	0.5
32-bit	4 Gb	4K	28.0	2.0
32-bit	4 Gb	5K	KO	5.7
32-bit	4 Gb	8K	KO	9.0
32-bit	4 Gb	15K	KO	32.9
32-bit	4 Gb	25K	KO	82.5
32-bit	4 Gb	40K	KO	213.3
32-bit	4 Gb	10KL	0.7	0.7
32-bit	4 Gb	20KL	1.9	0.8
32-bit	4 Gb	55KL	3.7	1.9
64-bit	16 Gb	1K	2.2	0.3
64-bit	16 Gb	4K	14.6	1.0
64-bit	16 Gb	5K	45.3	2.7
64-bit	16 Gb	8K	69.1	3.8
64-bit	16 Gb	15K	206.2	11.6
64-bit	16 Gb	25K	KO	27.6
64-bit	16 Gb	40K	KO	77.7
64-bit	16 Gb	10KL	0.2	0.2
64-bit	16 Gb	20KL	0.7	0.4
64-bit	16 Gb	55KL	1.4	0.8

Table 3: Use of the constraint matrix against the constraint vector

Config.	RAM	Exerc.	lpSolve	ROI.plugin.clp	Rglpk
32-bit	4 Gb	1K	0.5	0.1	0.4
32-bit	4 Gb	4K	2.0	0.2	1.0
32-bit	4 Gb	5K	5.7	0.3	2.7
32-bit	4 Gb	8K	9.0	0.3	3.5
32-bit	4 Gb	15K	32.9	0.6	9.6
32-bit	4 Gb	25K	82.5	0.7	27.7
32-bit	4 Gb	40K	213.3	1.3	73.2
32-bit	4 Gb	10KL	0.7	0.2	0.2
32-bit	4 Gb	20KL	0.8	0.2	0.3
32-bit	4 Gb	55KL	1.9	0.2	0.8
64-bit	16 Gb	1K	0.3	0.2	0.1
64-bit	16 Gb	4K	1.0	0.2	0.4
64-bit	16 Gb	5K	2.7	0.2	1.1
64-bit	16 Gb	8K	3.8	0.2	1.5
64-bit	16 Gb	15K	11.6	0.3	4.1
64-bit	16 Gb	25K	27.6	0.3	9.8
64-bit	16 Gb	40K	77.7	0.4	22.6
64-bit	16 Gb	10KL	0.2	0.1	0.1
64-bit	16 Gb	20KL	0.4	0.1	0.2
64-bit	16 Gb	55KL	0.8	0.2	0.3

Table 4: Reduction procedure using the three different packages

Exercise	Filter $r^*_{ab} > 0.5$		Filter $r_{ab} > 1$	
	Time	Recall	Time	Recall
1K	0.1	0.985	0.1	0.997
4K	0.2	0.943	0.2	0.976
5K	0.3	0.966	0.3	0.971
8K	0.3	0.944	0.3	0.950
15K	0.6	0.927	0.7	0.980
25K	0.7	0.698	0.7	0.710
40K	1.3	0.688	7.3	0.945
10KL	0.2	0.956	0.3	0.983
20KL	0.2	0.944	0.5	0.958
55KL	0.2	0.888	0.6	0.938

Table 5: Recall measure using different filters

MDFS: MultiDimensional Feature Selection in R

by Radosław Piliszek, Krzysztof Mnich, Szymon Migacz, Paweł Tabaszewski, Andrzej Sułecki, Aneta Polewko-Klim, and Witold Rudnicki

Abstract Identification of informative variables in an information system is often performed using simple one-dimensional filtering procedures that discard information about interactions between variables. Such an approach may result in removing some relevant variables from consideration. Here we present an R package **MDFS** (MultiDimensional Feature Selection) that performs identification of informative variables taking into account synergistic interactions between multiple descriptors and the decision variable. **MDFS** is an implementation of an algorithm based on information theory (Mnich and Rudnicki, 2017). The computational kernel of the package is implemented in C++. A high-performance version implemented in CUDA C is also available. The application of **MDFS** is demonstrated using the well-known Madelon dataset, in which a decision variable is generated from synergistic interactions between descriptor variables. It is shown that the application of multidimensional analysis results in better sensitivity and ranking of importance.

Introduction

Identification of variables that are related to the decision variable is often the most important step in dataset analysis. In particular, it becomes really important when the number of variables describing the phenomena under scrutiny is large.

Methods of feature selection fall into three main categories (Guyon and Elisseeff, 2003):

- filters, where the identification of informative variables is performed before data modelling and analysis,
- wrappers, where the identification of informative variables is achieved by analysis of the models,
- embedded methods, which evaluate utility of variables in the model and select the most useful variables.

Filters are designed to provide a quick answer and therefore are the fastest. On the other hand, their simplicity is also the source of their errors. The rigorous univariate methods, such as *t*-test, do not detect interactions between variables. Heuristical methods that avoid this trap, such as Relief-f algorithm (Kononenko, 1994), may be biased towards weak and correlated variables (Robnik-Šikonja and Kononenko, 2003). Interesting heuristical filter based on decision trees – Monte Carlo Feature Selection (MCFS) (Dramiński et al., 2007; Dramiński and Koronacki, 2018) – avoids this pitfall. However, it may fail to find purely synergistic variables. Several filtering methods are designed to return only the non-redundant subset of variables (Zhao and Liu, 2007; Peng et al., 2005; Wang et al., 2013). While such methods may lead to very efficient models, their selection may be far from the best when one is interested in deeper understanding of the phenomena under scrutiny.

The wrapper algorithms are designed around machine learning algorithms such as SVM (Cortes and Vapnik, 1995), as in the SVM-RFE algorithm (Guyon et al., 2002), or random forest (Breiman, 2001), as in the Boruta algorithm (Kursa et al., 2010). They can identify variables involved in non-linear interactions. Unfortunately, for systems with tens of thousands of variables they are slow. For example, the Boruta algorithm first expands the system with randomised copies of variables and then requires numerous runs of the random forest algorithm.

The embedded methods are mostly limited to linear approximations and are part of a modelling approach where the selection is directed towards the utility of the model. Therefore, variables that are relevant for understanding the phenomena under scrutiny may be omitted and replaced by variables more suitable for building a particular model.

Here we introduce an R package implementing a filter based on information theory. The algorithm can identify synergistic relevant variables by performing an exhaustive search of low-dimensional combinations of variables.

Theory

Kohavi and John proposed that a variable $x_i \in X$, where X is a set of all descriptive variables, is weakly relevant if there exists a subset of variables $X_{sub} \subset X : x_i \notin X_{sub}$ that one can increase information on the decision variable y by extending this subset with the variable x_i (Kohavi and John, 1997). Mnich and Rudnicki introduced the notion of k-weak relevance, that restricts the original definition by Kohavi and John to $(k-1)$-element subsets X_{sub} (Mnich and Rudnicki, 2017).

The algorithm implements the definition of k-weak relevance directly by exploring all possible k-tuples of variables $x_i \cup \{x_{m_1}, x_{m_2}, \ldots, x_{m_{k-1}}\}$ for k-dimensional analysis. For example, in 2 dimensions we explore a set of all possible pairs of variables. For each variable x_i we check whether adding it to another variable x_k adds information to the system. If there exists such x_k, then we declare x_i as 2-weakly relevant.

The maximum decrease in conditional information entropy upon adding x_i to description, normalized to sample size, is used as the measure of x_i's relevance:

$$IG^k_{max}\left(y; x_i\right) = N \max_m \left(H\left(y|x_{m_1}, x_{m_2}, \ldots, x_{m_{k-1}}\right) - H\left(y|x_i, x_{m_1}, x_{m_2}, \ldots, x_{m_{k-1}}\right)\right), \quad \text{(F.2.1)}$$

where H is (conditional) information entropy and N is the number of observations. Difference in (conditional) information entropy is known as (conditional) mutual information. It is multiplied by N to obtain the proper null-hypothesis distribution. To name this value we reused the term information gain (IG) which is commonly used in information-theoretic context to denote different values related to mutual information.

To declare a variable k-weakly relevant it is required that its $IG^k_{max}(y; x_i)$ is statistically significant. This can be established via a comparison:

$$IG^k_{max}\left(y; x_i\right) \geqslant IG_{lim}, \quad \text{(F.2.2)}$$

where IG_{lim} is computed using a procedure of fitting the theoretical distribution to the data.

For a sufficiently large sample, the value of IG for a non-informative variable, with respect to a single k-tuple, follows a χ^2 distribution. $IG^k_{max}(y; x_i)$, which is the maximum value of IG among many trials, follows an extreme value distribution. This distribution has one free parameter corresponding to the number of independent tests which is generally unknown and smaller than the total number of tests. The parameter is thus computed empirically by fitting the distribution to the irrelevant part of the data (Mnich and Rudnicki, 2017). This allows to convert the IG^k_{max} statistic to its p-value and then to establish IG_{lim} as a function of significance level α. Since many variables are investigated, the p-value should be adjusted using well-known FWER (Holm, 1979) or FDR (**?**) control technique. Due to unknown dependencies between tests, for best results we recommend using Benjamini-Hochberg-Yekutieli method (Benjamini and Yekutieli, 2001)[1] when performing FDR control.

In one dimension ($k = 1$) Equation F.2.1 reduces to:

$$IG^1_{max}\left(y; x_i\right) = N\left(H\left(y\right) - H\left(y|x_i\right)\right), \quad \text{(F.2.3)}$$

which is a well-known G-test statistic (**?**).

All variables that are weakly relevant in one-dimensional test should also be discovered in higher-dimensional tests, nevertheless their relative importance may be significantly influenced by interactions with other variables. Often the criterium for inclusion to further steps of data analysis and model building is simply taking the top n variables, therefore the ordering of variables due to importance matters as well.

Algorithm and implementation

The **MDFS** package (Piliszek et al., 2018) consists of two main parts. One is an R interface to two computational engines. These engines utilise either CPU or NVIDIA GPU and are implemented in standard C++ and in CUDA C, respectively. Either computational engine returns the IG^k_{max} distribution for a given dataset plus requested details which may pose an interesting insight into data. The other part is a toolkit to analyse results. It is written entirely in R. The version of **MDFS** used and described here is 1.0.3. The term 'MDFS' (MultiDimensional Feature Selection) is used to denote the analysis, method and algorithm presented in this article as well.

The IG^k_{max} for each variable is computed using a straightforward algorithm based on Equation F.2.1. Information entropy (H) is computed using discretised descriptive variables. Discretisation is performed using customisable randomised rank-based approach. To control the discretisation process we use a concept of range. Range is a real number between 0 and 1 affecting the share each discretised variable class has in the dataset. Each share is sampled from a uniform distribution on

[1] Method `"BY"` for `p.adjust` function.

the interval $(1 - \text{range}, 1 + \text{range})$. Hence, $\text{range} = 0$ results in an equipotent split, $\text{range} = 1$ equals a completely random split. Let us assume that there are N objects in the system and we want to discretise a variable to c classes. To this end, $(c - 1)$ distinct integers from the range $(2, N)$ are obtained using computed shares. Then, the variable is sorted and values at positions indexed by these integers are used to discretise the variable into separate classes. In most applications of the algorithm there is no default best discretisation of descriptive variables, hence multiple random discretisations are performed. The IG^k_{max} is computed for each discretisation, then the maximum IG^k_{max} over all discretizations is returned. Hence, the final returned IG^k_{max} is a maximum over both tuples and discretisations.

The problem of selecting the right amount of classes (the right value of c) is similar to bias–variance tradeoff but more subtle. The statistic is better when there are less classes (binary being the best case) but the shape ought to be better when there are more classes as it improves the resolution. When the right split is known (as we show later with Madelon), it is best to use it. Otherwise we recommend to try different numbers of classes and do several random discretizations for each.

Conditional information entropy is obtained from the experimental probabilities of a decision class using the following formula:

$$H\left(y|x_1, \ldots, x_k\right) = - \sum_{d=0,1} \sum_{i_1=1:c} \cdots \sum_{i_k=1:c} p^d_{i_1,\ldots,i_k} \log\left(p^d_{i_1,\ldots,i_k}\right), \qquad \text{(F.3.1)}$$

where $p^d_{i_1,\ldots,i_k}$ denotes the conditional probability of class d in a k-dimensional voxel with coordinates i_j. Note that the number of voxels in k dimensions is c^k, where c is the number of classes of discretised descriptive variables. To this end, one needs to compute the number of instances of each class in each voxel. The conditional probability of class d in a voxel is then computed as

$$p^d_{i_1,\ldots,i_k} = \frac{N^d_{i_1,\ldots,i_k} + \beta^d}{N^0_{i_1,\ldots,i_k} + \beta^0 + N^1_{i_1,\ldots,i_k} + \beta^1}, \qquad \text{(F.3.2)}$$

where $N^d_{i_1,\ldots,i_k}$ is the count of class d in a k-dimensional voxel with coordinates i_j and β^d is a pseudocount corresponding to class d:

$$\beta^d = \xi \frac{N^d}{\min\left(N^0, N^1\right)}, \qquad \text{(F.3.3)}$$

where $\xi > 0$ can be supplied by the user. The default value is set to 0.25. It was obtained in an experimental process to achieve the best fit to χ^2 distribution. Usual usage should not mandate the need to change ξ.

The implementation of the algorithm is currently limited to binary decision variables. The analysis for information systems that have more than two categories can be performed either by executing all possible pairwise comparisons or one-vs-rest. Then all variables that are relevant in the context of a single pairwise comparison should be considered relevant. In the case of continuous decision variable one must discretise it before performing analysis. In the current implementation all variables are discretised into an equal number of classes. This constraint is introduced for increased efficiency of computations, in particular on a GPU.

Another limitation is the maximum number of dimensions set to 5. This is due to several reasons. Firstly, the computational cost of the algorithm is proportional to number of variables to power equal the dimension of the analysis, and it becomes prohibitively expensive for powers larger than 5 even for systems described with a hundred of variables. Secondly, analysis in higher dimensions requires a substantial number of objects to fill the voxels sufficiently for the algorithm to detect real synergies. Finally, it is also related to the simplicity of efficient implementation of the algorithm in CUDA.

The most time-consuming part of the algorithm is computing the counters for all voxels. Fortunately, this part of the computations is relatively easy to parallelise, as the exhaustive search is very well suited for GPU. Therefore, a GPU version of the algorithm was developed in CUDA C for NVIDIA GPGPUs and is targeted towards problems with a very large number of features. The CPU version is also parallelised to utilise all cores available on a single node. The 1D analysis is available only in the CPU version since there is no benefit in running this kind of analysis on GPU.

Package functions introduction

There are three functions in the package which are to be run directly with the input dataset: `MDFS`, `ComputeMaxInfoGains`, and `ComputeInterestingTuples`. The first one, `MDFS`, is our recommended function for new users, since it hides internal details and provides an easy to use interface for basic end-to-end analysis for current users of other statistical tests (e.g., `t.test`) so that the user can straightforwardly get the statistic values, p-values, and adjusted p-values for variables from input. The other two functions are interfaces to the IG-calculating lower-level C++ and CUDA C++ code. `ComputeMaxInfoGains` returns the max IGs, as described in the theory section. It can optionally provide information about the tuple in which this max IG was observed. On the other hand, one might be interested in tuples where certain IG threshold has been achieved. The `ComputeInterestingTuples` function performs this type of analysis and reports which variable in which tuple achieved the corresponding IG value.

The `ComputePValue` function performs fitting of IGs to respective statistical distributions as described in the theory section. The goodness of fit is tested using Kolmogorov-Smirnov one-sample test and a warning is emitted if the threshold is exceeded. `ComputePValue` returns an object of the `"MDFS"` class which contains, in particular, p-values for variables. This class implements various methods for handling output of statistical analysis. In particular they can plot details of IG distribution, output p-values of all variables, and output relevant variables. `ComputePValue` is implemented in a general way, extending beyond limitations of the current implementation of `ComputeMaxInfoGains`. In particular, it can handle multi-class problems and different number of divisions for each variable.

The `AddContrastVariables` is an utility function used to construct contrast variables (Stoppiglia et al., 2003; Kursa et al., 2010). Contrast variables are used solely for improving reliability of the fit of statistical distribution. In the case of fitting distribution to contrast variables we know exactly how many irrelevant variables there are in the system. The contrast variables are not tested for relevance and hence not used when adjusting p-values to not decrease the sensitivity without reason.

Canonical package usage

As mentioned earlier, the recommended way to use the package is to use the `MDFS` function. It uses the other packaged functions to achieve its goal in the standard and thoroughly tested way, so it may be considered the canonical package usage pattern. The `MDFS` function is general in terms of contrast variables being optional, hence let us examine a simplified version of it assuming the contrast variables are actually being used. We also neglect the setting of seed but we recommend it to be set so that the result is reproducible. The `MDFS` wrapper does accept a seed and saves it with the result.

The first step is to build the contrast variables:

```
contrast <- AddContrastVariables(data, n.contrast)
```

In the next step, the compute-intensive computation of IGs is executed:

```
MIG.Result <- ComputeMaxInfoGains(contrast$x, decision,
  dimensions = dimensions, divisions = divisions,
  discretizations = discretizations, range = range, pseudo.count = pseudo.count)
```

The first two positional parameters are respectively the feature data (plus contrast variables) and the decision. The other parameters decide on the type of computed IGs: `dimensions` controls dimensionality, `divisions` controls the number of classes in the discretisation (it is equal to `divisions+1`), `discretizations` controls the number of discretisations, `range` controls how random the discretisation splits are, and `pseudo.count` controls the regularization parameter ξ for pseudocounts.

Finally, the computed IGs are analysed and a statistical result is computed:

```
fs <- ComputePValue(MIG.Result$IG,
  dimensions = dimensions, divisions = divisions,
  contrast.mask = contrast$mask,
  one.dim.mode = ifelse (discretizations==1, "raw",
                         ifelse(divisions*discretizations<12, "lin", "exp")))

statistic <- MIG.Result$IG[!contrast$mask]
p.value <- fs$p.value[!contrast$mask]
```

```
adjusted.p.value <- p.adjust(p.value, method = p.adjust.method)
relevant.variables <- which(adjusted.p.value < level)
```

The `one.dim.mode` parameter controls the expected distribution in 1D. The rule states that as long as we have 1 discretisation the resulting distribution is chi-squared, otherwise, depending on the product of `discretizations` and `divisions`, the resulting distribution might be closer to a linear or exponential, as in higher dimensions, function of chi-squared distributions. This is heuristic and might need to be tuned. Features with adjusted *p*-values below some set level are considered to be relevant.

Testing the Madelon dataset

For demonstration of the **MDFS** package we used the training subset of the well-known Madelon dataset (Guyon et al., 2007). It is an artificial set with 2000 objects and 500 variables. The decision was generated using a 5-dimensional random parity function based on variables drawn from normal distribution. The remaining variables were generated in the following way. Fifteen variables were obtained as linear combinations of the 5 input variables and remaining 480 variables were drawn randomly from the normal distribution. The data set can be accessed from the UCI Machine Learning Repository (Dheeru and Karra Taniskidou, 2017) and it is included in **MDFS** package as well.

We conducted the analysis in all possible dimensionalities using both CPU and GPU versions of the code. Additionally, a standard *t*-test was performed for reference. We examined computational efficiency of the algorithm and compared the results obtained by performing analysis in varied dimensionalities.

In the first attempt we utilised the given information on the properties of the dataset under scrutiny. We knew in advance that Madelon was constructed as a random parity problem and that each base variable was constructed from a distinct distribution. Therefore, we could use one discretisation into 2 equipotent classes. In the second attempt the recommended 'blind' approach in 2D was followed, which utilises several randomized discretisations.

For brevity, in the following examples the set of Madelon independent variables is named `x` and its decision is named `y`:

```
x <- madelon$data
y <- madelon$decision
```

For easier comparison we introduce a helper function to obtain, from MDFS analysis, the relevant features' indices in decreasing relevance (increasing *p*-value) order:

```
GetOrderedRelevant <- function (result) {
  result$relevant.variables[order(result$p.value[result$relevant.variables])]
}
```

One can now obtain *p*-values from *t*-test, adjust them using Holm correction (one of FWER corrections, the default in the `p.adjust` function), take relevant with level 0.05, and order them:

```
> tt <- ttests(x, ina = y+1)[,2] # we only use $p$-values (2nd column)
> tt.adjusted <- p.adjust(tt, method = "holm")
> tt.relevant <- which(tt.adjusted < 0.05)
> tt.relevant.ordered <- tt.relevant[order(tt.adjusted[tt.relevant])]
> tt.relevant.ordered
[1] 476 242 337  65 129 106 339  49 379 443 473 454 494
```

A FWER correction is used because we expect strong separation between relevant and irrelevant features in this artificial dataset. We used the `ttests` function from the **Rfast** (Papadakis et al., 2018) package as it is a version of *t*-test optimized for this purpose.

To achieve the same with **MDFS** for 1, 2, and 3 dimensions one can use the wrapper `MDFS` function:

```
> d1 <- MDFS(x, y, n.contrast = 0, dimensions = 1, divisions = 1, range = 0)
> d1.relevant.ordered <- GetOrderedRelevant(d1)
> d1.relevant.ordered
[1] 476 242 339 337  65 129 106  49 379 454 494 443 473

> d2 <- MDFS(x, y, n.contrast = 0, dimensions = 2, divisions = 1, range = 0)
> d2.relevant.ordered <- GetOrderedRelevant(d2)
> d2.relevant.ordered
[1] 476 242  49 379 154 282 434 339 494 454 452  29 319 443 129 473 106 337  65

> d3 <- MDFS(x, y, n.contrast = 0, dimensions = 3, divisions = 1, range = 0)
> d3.relevant.ordered <- GetOrderedRelevant(d3)
> d3.relevant.ordered
[1] 154 434 282  49 379 476 242 319  29 452 494 106 454 129 473 443 339 337  65 456
```

The changes in the relevant variables set can be examined with simple `setdiff` comparisons:

```
> setdiff(tt.relevant.ordered, d1.relevant.ordered)
integer(0)
> setdiff(d1.relevant.ordered, tt.relevant.ordered)
integer(0)
> setdiff(d1.relevant.ordered, d2.relevant.ordered)
integer(0)
> setdiff(d2.relevant.ordered, d1.relevant.ordered)
[1] 154 282 434 452  29 319
> setdiff(d2.relevant.ordered, d3.relevant.ordered)
integer(0)
> setdiff(d3.relevant.ordered, d2.relevant.ordered)
[1] 456
```

One may notice that ordering by importance leads to different results for these 4 tests.

In the above the knowledge about properties of the Madelon dataset was used: that there are many random variables, hence no need to add contrast variables, and that the problem is best resolved by splitting features in half, hence one could use 1 discretisation and set range to zero.

However, one is usually not equipped with such knowledge and then may need to use multiple random discretisations. Below an example run of ‘blind’ 2D analysis of Madelon is presented:

```
> d2b <- MDFS(x, y, dimensions = 2, divisions = 1, discretizations = 30, seed = 118912)
> d2b.relevant.ordered <- GetOrderedRelevant(d2b)
> d2b.relevant.ordered
[1] 476 242 379  49 154 434 106 282 473 339 443 452  29 454 494 319  65 337 129
> setdiff(d2b.relevant.ordered, d2.relevant.ordered)
integer(0)
> setdiff(d2.relevant.ordered, d2b.relevant.ordered)
integer(0)
```

This demonstrates that the same variables are discovered, yet with a different order.

Performance

The performance of the CPU version of the algorithm was measured on a computer with two Intel Xeon E5-2650 v2 processors, running at 2.6 GHz. Each processor has eight physical cores. Hyperthreading was disabled.

The GPU version was tested on a computer with a single NVIDIA Tesla K80 accelerator. The K80 is equipped with two GK210 chips and is therefore visible to the system as two separate GPGPUs. Both were utilised in the tests.

The Madelon dataset has moderate dimensionality for modern standards, hence it is amenable to high-dimensional analysis. The CPU version of the code handles analysis up to four dimensions in a reasonable time, see Table 1.

The performance gap between CPU and GPU versions is much higher than suggested by a simple comparison of hardware capabilities. This is due to two factors. Firstly, the GPU version has been highly optimised towards increased efficiency of memory usage. The representation of the data by bit-vectors and direct exploitation of the data locality allows for much higher data throughput. What is more, the bit-vector representation allows for using very efficient *popcnt* instruction for counter updates. On the other hand the CPU version has been written mainly as a reference version using a straightforward implementation of the algorithm and has not been strongly optimised.

	t-test	1D	2D	3D	4D	5D
CPU	0.01s	0.01s	0.44s	42s	1h:58m	249h
GPU	-	-	0.23s	0.2s	9.8s	59m:37s

Table 1: Execution times for the Madelon dataset.

	1D	2D	3D	4D	5D
CPU	0.35s	5.8s	37m:11s	92h	-
GPU	-	2.9s	3.3s	7m:36s	42h

Table 2: Execution times for the Madelon dataset with 30 random discretisations.

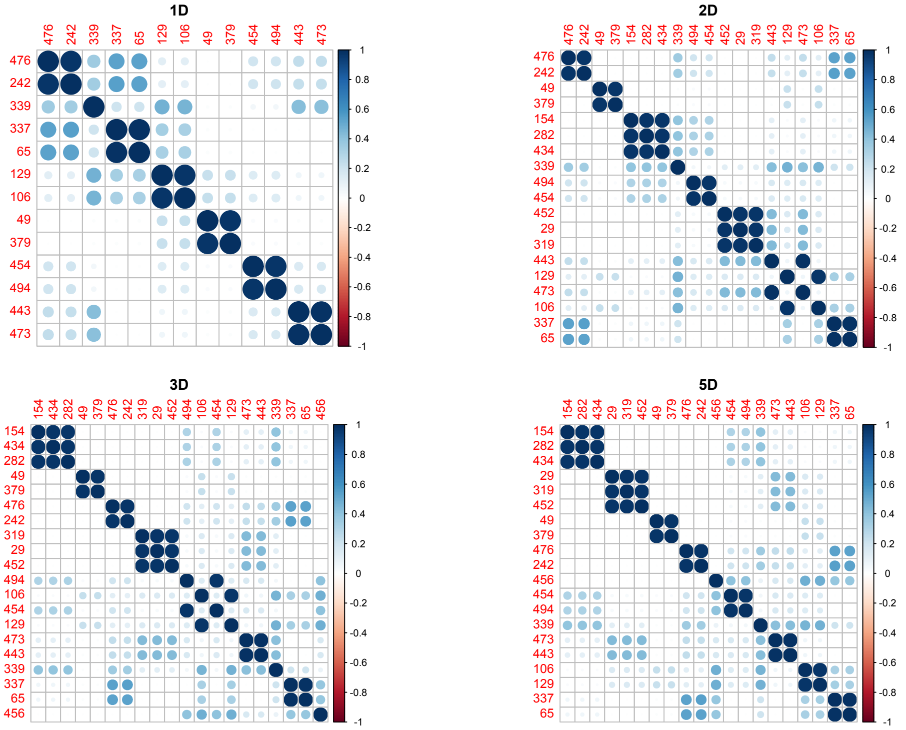


Figure 1: Correlation plots for relevant variables discovered in 1-, 2-, 3-, and 5-dimensional analysis of the Madelon dataset with one deterministic discretisation with division in the middle. The variables are ordered by IG.

Structure of the Madelon dataset revealed by MDFS analysis

The twenty relevant variables in Madelon can be easily identified by analysis of histograms of variables, their correlation structure and by a priori knowledge of the method of construction of the

Cluster	Members
154	154, 282, 434
29	29, 319, 452
49	49, 379
242	476, 242
456	456
454	454, 494
339	339
443	473, 443
106	106, 129
65	337, 65

Table 3: Discovered variable clusters (as seen in correlation plots) ordered by descending maximum relevance (measured with 5D IG), identified by the variable with the lowest number.

dataset. In particular, base variables, i.e. these variables that are directly connected to a decision variable, have the unique distribution that has two distinct peaks. All other variables have smooth unimodal distribution, hence identification of base variables is trivial. What is more, we know that remaining informative variables are constructed as linear combinations of base variables, hence they should display significant correlations with base variables. Finally, the nuisance variables are completely random, hence they should not be correlated neither with base variables nor with their linear combinations. The analysis of correlations between variables reveals also that there are several groups of very highly correlated ($r > 0.99$) variables, see Figure 1. All variables in such a group can be considered as a single variable, reducing the number of independent variables to ten. The entire group is further represented by the variable with the lowest number. The clusters are presented in Table 3.

This clear structure of the dataset creates an opportunity to confront results of the MDFS analysis with the ground truth and observe how the increasing precision of the analysis helps to discover this structure without using the a priori knowledge on the structure of the dataset.

One-dimensional analysis reveals 13 really relevant variables (7 independent ones), both by means of the *t*-test and using the information gain measure, see Table 4. Three-dimensional and higher-dimensional analyses find all 20 relevant variables. Additionally, with the exception of one-dimensional case, in all cases there is a clear separation between IG obtained for relevant and

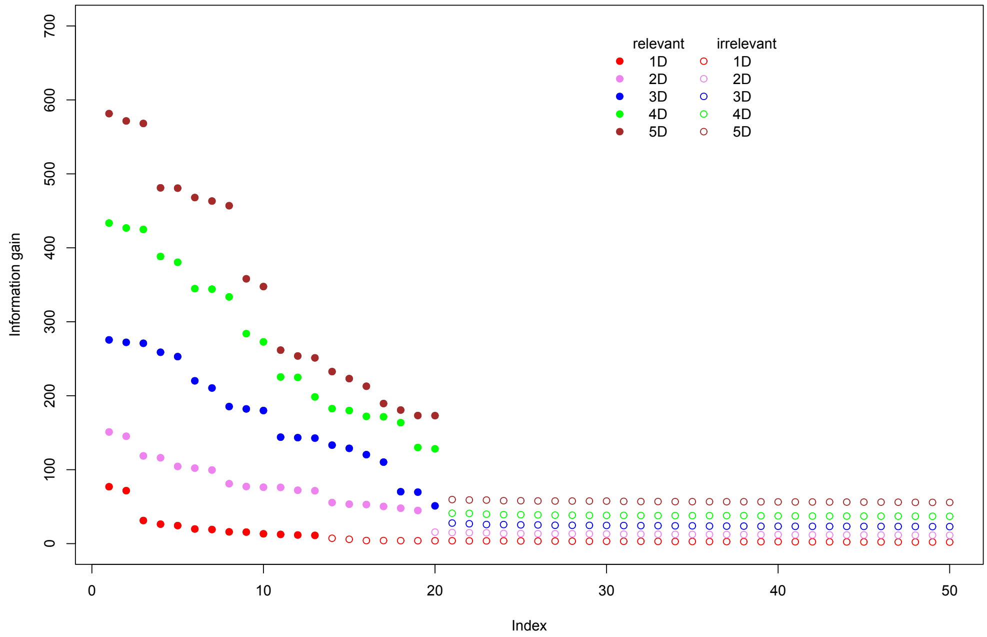


Figure 2: Information gain obtained by the MDFS algorithm using 1-, 2-, 3-, 4-, and 5-dimensional variants of the algorithm for the Madelon dataset with one deterministic discretisation with division in the middle. Full circles represent variables deemed relevant. All variables are sorted by IG. Margin between irrelevant and relevant features grows with dimensionality.

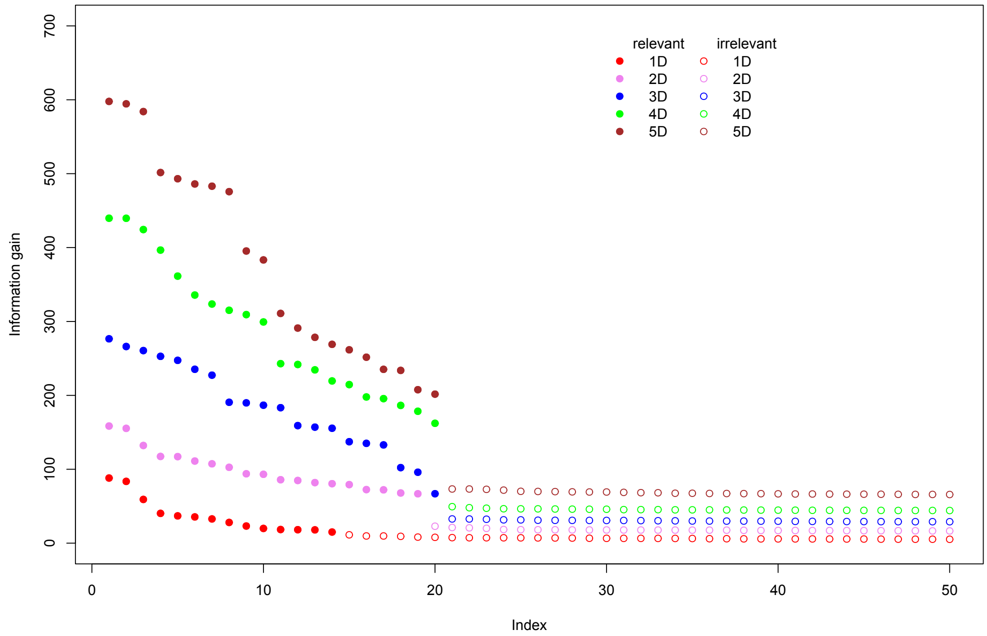


Figure 3: Information gain obtained by the MDFS algorithm using 1-, 2-, 3-, 4-, and 5-dimensional variants of the algorithm for the Madelon dataset with 30 random discretisations. Full circles represent variables deemed relevant. All variables are sorted by IG. The margin between irrelevant and relevant features grows with dimensionality.

Table 4: Summary of results for the Madelon dataset with one deterministic discretisation with division in the middle. The variable clusters are ordered by descending IG. The numbers of base variable clusters are highlighted in boldface. Clusters represented by 65 and 106, displayed in italic font, are deemed highly relevant in 1D analyses and the least relevant in 5D analysis.

	t-test	1D	2D	3D	4D	5D
1.	**242**	**242**	**242**	**154**	**154**	**154**
2.	*65*	339	**49**	**49**	**49**	**29**
3.	*106*	*65*	**154**	**242**	**29**	**49**
4.	339	*106*	339	**29**	**242**	**242**
5.	**49**	**49**	454	454	454	**456**
6.	443	454	**29**	*106*	339	454
7.	454	443	443	443	*106*	339
8.	-	-	*106*	339	**456**	443
9.	-	-	*65*	*65*	443	*106*
10.	-	-	-	**456**	*65*	*65*

irrelevant variables, see Figure 2. This translates into a significant drop of p-value for the relevant variables.

Five variables, namely $\{29, 49, 154, 242, 456\}$, are clearly orthogonal to each other, hence they are the base variables used to generate the decision variable. Five other variables are correlated with base variables and with each other, and hence they are linear combinations of base variables. The one-dimensional analyses, both t-test and mutual-information approach, find only two base variables, see Table 4. What is more, while one of them is regarded as highly important (lowest p-value), the second one is considered only the 5th most important out of 7. Two-dimensional analysis finds 4 or 5 base variables, depending on the method used. Moreover, the relative ranking of variables is closer to intuition, with three base variables on top. The relative ranking of importance improves with increasing dimensionality of the analysis. In 5-dimensional analysis all five base variables are scored higher than any linear combination. In particular, the variable 456, which is identified by 3D analysis as the least important, rises to the eighth place in 4D analysis and to the fifth in 5D. Interestingly, the variable 65, which is the least important in 5D analysis is the second most important variable in t-test and the third most important variable in 1D.

Table 5: Summary of results for the Madelon dataset with 30 random discretisations. The variable clusters are ordered by descending IG. The numbers of base variable clusters are highlighted in boldface. Similar behaviour with 65 and 106 is observed as in the single discretisation case. Note the irrelevant variable 205 (underlined) discovered in 1D as relevant due to small margin between relevant and irrelevant features.

	t-test	1D	2D	3D	4D	5D
1.	**242**	**242**	**242**	**154**	**154**	**154**
2.	*65*	339	**49**	**49**	**49**	**29**
3.	*106*	*65*	**154**	**242**	**29**	**49**
4.	339	443	*106*	**29**	**242**	**242**
5.	**49**	*106*	443	*106*	454	454
6.	443	454	339	454	*106*	**456**
7.	454	**49**	**29**	443	339	443
8.	-	205	454	339	443	339
9.	-	-	*65*	*65*	**456**	*106*
10.	-	-	-	**456**	*65*	*65*

Conclusion

We have introduced a new package for identification of informative variables in multidimensional information systems which takes into account interactions between variables. The implemented method is significantly more sensitive than the standard *t*-test when interactions between variables are present in the system. When applied to the well-known five-dimensional problem—Madelon—the method not only discovered all relevant variables but also produced the correct estimate of their relative relevance.

Acknowledgments

The research was partially funded by the Polish National Science Centre, grant 2013/09/B/ST6/01550.

Bibliography

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. [p199]

L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001. URL https://doi.org/10.1023/a:1010933404324. [p198]

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. URL https://doi.org/10.1007/bf00994018. [p198]

D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml. [p202]

M. Dramiński and J. Koronacki. Rmcfs: An r package for monte carlo feature selection and interdependency discovery. *Journal of Statistical Software, Articles*, 85(12):1–28, 2018. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v085.i12. [p198]

M. Dramiński, A. Rada-Iglesias, S. Enroth, C. Wadelius, J. Koronacki, and J. Komorowski. Monte carlo feature selection for supervised classification. *Bioinformatics*, 24(1):110–117, 2007. ISSN 1367-4803. [p198]

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003. URL https://doi.org/10.1162/153244303322753616. [p198]

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002. URL https://doi.org/10.1023/a:1012487302797. [p198]

I. Guyon, J. Li, T. Mader, P. A. Pletscher, G. Schneider, and M. Uhr. Competitive Baseline Methods Set New Standards for the NIPS 2003 Feature Selection Benchmark. *Pattern Recognition Letters*, 28(12):1438–1444, 2007. URL https://doi.org/10.1016/j.patrec.2007.02.014. [p202]

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979. ISSN 03036898, 14679469. URL https://doi.org/10.2307/4615733. [p199]

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2):273–324, 1997. ISSN 0004-3702. URL https://doi.org/10.1016/s0004-3702(97)00043-x. [p199]

I. Kononenko. Estimating attributes: Analysis and extensions of relief. In *European Conference on Machine Learning*, pages 171–182, 1994. URL https://doi.org/10.1007/3-540-57868-4_57. [p198]

M. B. Kursa, A. Jankowski, and W. R. Rudnicki. Boruta – a system for feature selection. *Fundamenta Informaticae*, 101(4):271–285, 2010. URL https://doi.org/10.3233/fi-2010-288. [p198, 201]

K. Mnich and W. R. Rudnicki. All-relevant feature selection using multidimensional filters with exhaustive search. *CoRR*, abs/1705.05756, 2017. URL http://arxiv.org/abs/1705.05756. [p198, 199]

M. Papadakis, M. Tsagris, M. Dimitriadis, S. Fafalios, I. Tsamardinos, M. Fasiolo, G. Borboudakis, J. Burkardt, C. Zou, K. Lakiotaki, and C. Chatzipantsiou. *Rfast: A Collection of Efficient and Extremely Fast R Functions*, 2018. URL https://CRAN.R-project.org/package=Rfast. R package version 1.9.2. [p202]

H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005. URL https://doi.org/10.1109/tpami.2005.159. [p198]

R. Piliszek, K. Mnich, P. Tabaszewski, S. Migacz, A. Sułecki, and W. R. Rudnicki. *MDFS: MultiDimensional Feature Selection*, 2018. URL https://CRAN.R-project.org/package=MDFS. R package version 1.0.3. [p199]

M. Robnik-Šikonja and I. Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine Learning*, 53(1-2):23–69, 2003. URL https://doi.org/10.1023/a:1025667309714. [p198]

H. Stoppiglia, G. Dreyfus, R. Dubois, and Y. Oussar. Ranking a random feature for variable and feature selection. *Journal of Machine Learning Research*, 3(7-8):1399–1414, 2003. URL https://doi.org/10.1162/153244303322753733. [p201]

G. Wang, Q. Song, B. Xu, and Y. Zhou. Selecting feature subset for high dimensional data via the propositional foil rules. *Pattern Recognition*, 46(1):199–214, 2013. URL https://doi.org/10.1016/j.patcog.2012.07.028. [p198]

Z. Zhao and H. Liu. Searching for interacting features. In *IJCAI*, volume 7, pages 1156–1161, 2007. [p198]

Radosław Piliszek
Computational Centre, University of Bialystok
Konstantego Ciolkowskiego 1M, 15-245 Bialystok
Poland
0000-0003-0729-9167
r.piliszek@uwb.edu.pl

Krzysztof Mnich
Computational Centre, University of Bialystok
Konstantego Ciolkowskiego 1M, 15-245 Bialystok
Poland
0000-0002-6226-981X
k.mnich@uwb.edu.pl

Szymon Migacz
Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw
Pawińskiego 5A, 02-106 Warsaw
Poland

Paweł Tabaszewski
Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw
Pawińskiego 5A, 02-106 Warsaw
Poland

Andrzej Sułecki
Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw
Pawińskiego 5A, 02-106 Warsaw
Poland

Aneta Polewko-Klim
Institute of Informatics, University of Bialystok
Konstantego Ciolkowskiego 1M, 15-245 Bialystok
Poland
0000-0003-1987-7374
anetapol@uwb.edu.pl

Witold Rudnicki
Institute of Informatics, University of Bialystok
Konstantego Ciolkowskiego 1M, 15-245 Bialystok
Poland
and
Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw
Pawińskiego 5A, 02-106 Warsaw
Poland
0000-0002-7928-4944
w.rudnicki@uwb.edu.pl

fclust: An R Package for Fuzzy Clustering

by Maria Brigida Ferraro, Paolo Giordani and Alessio Serafini

Abstract Fuzzy clustering methods discover fuzzy partitions where observations can be softly assigned to more than one cluster. The package **fclust** is a toolbox for fuzzy clustering in the R programming language. It not only implements the widely used fuzzy k-means (FkM) algorithm, but also many FkM variants. Fuzzy cluster similarity measures, cluster validity indices and cluster visualization tools are also offered. In the current version, all the functions are rewritten in the C++ language allowing their application in large-size problems. Moreover, new fuzzy relational clustering algorithms for partitioning qualitative/mixed data are provided together with an improved version of the so-called Gustafson-Kessel algorithm to avoid singularity in the cluster covariance matrices. Finally, it is now possible to automatically select the number of clusters by means of the available fuzzy cluster validity indices.

Introduction

Standard clustering algorithms assign a set of observations to a limited number of clusters such that observations belonging to the same cluster are more similar to each other than to those in other groups. Generally speaking, such algorithms usually produce a *hard* partition of the observations, i. e. every observation is assigned to one and only one cluster. This may often be too strict leading to unfeasible partitions. The well-known Butterfly dataset (Ruspini, 1970) helps to clarify the problem. It is available in the matrix `butterfly` of the package **fclust**, provided that the first and the last rows of the matrix are removed.

```
> data("butterfly", package = "fclust")
> butterfly <- butterfly[-c(1,17),]
> rownames(butterfly) <- as.character(rep(1:nrow(butterfly)))
> plot(butterfly, type = "n", xlab = "Var. 1", ylab="Var. 2")
> text(butterfly[,1], butterfly[,2], labels = rownames(butterfly), cex = 0.7, lwd = 2)
```

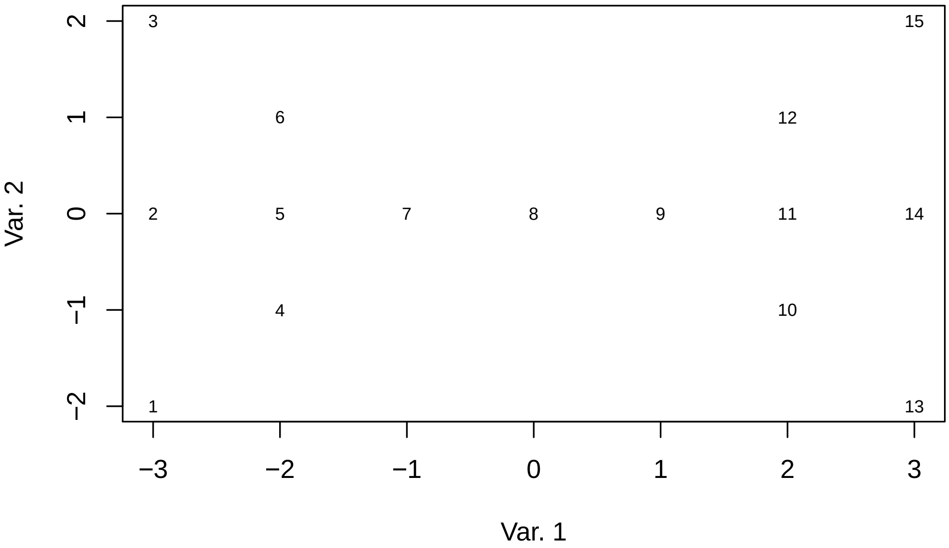


Figure 1: Scatterplot of the Butterfly data.

The Butterfly data refer to 15 observations and 2 variables. Two clusters corresponding to the left and right wings (observations n.1-n.7 and n.9-n.15, respectively) of the butterfly can be graphically depicted without any need of clustering tools. The assignment of observation n.8 (the body of the butterfly) is a much more complex issue because it is at the halfway between the two clusters. Standard algorithms fail to assign properly such an observation. For instance, let us consider the (hard) k-means (kM) algorithm (Hartigan and Wong, 1979), the most known clustering algorithm. We run the kM algorithm (using the function `kmeans`) a large number of times (`nt`).

```
> set.seed(12)
> nt <- 1000
> ca8 <- rep(NA,nt)
> lfv <- rep(NA,nt)
> for (n in 1: nt){
+   km.butterfly <- kmeans(butterfly, centers = 2, iter.max = 1000, nstart = 10)
+   lfv[n] <- km.butterfly[[5]]
+   if (km.butterfly$cluster[8] == km.butterfly$cluster[1]){
+     ca8[n] <- 1
+   }else{
+     ca8[n] <- 2
+   }
+ }

> summary(lfv)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  31.43   31.43   31.43   31.43   31.43   31.43

> table(ca8)

ca8
  1   2
560 440
```

We find (details not reported) that the same two clusters are always obtained (the one formed by observations n-1-n.7, labelled Cluster 1, and the other one by observations n-9-n.15, labelled Cluster 2), whilst observation n.8 is assigned to one of the two clusters (`ca8`) by chance without affecting the loss function value (`lfv`), i. e. the total within sum of squares.

The difficulty in the assignment of observation n.8 depends on the fact that it shares the features of both the groups. This situation frequently occurs in real life applications. In general, it may exist observations without clear assignments to the clusters. In such cases, it would be desirable to assign them to the clusters to a certain extent. For instance, in the butterfly problem, observation n.8 should be assigned to the two clusters with the same degree. This goal can be achieved by the *fuzzy* approach to clustering where observations belong to clusters with the so-called membership degrees in $[0, 1]$ and, for each observation, the sum of the membership degrees must be equal to 1. Therefore, in fuzzy clustering, the observations are not strictly assigned to one and only one cluster, but can share their memberships to more than one cluster. To differentiate the fuzzy approach from the standard hard one, it may also be referred to as *soft* clustering.

The most known fuzzy clustering algorithm is the fuzzy k-means (FkM), proposed by Bezdek (1981), which is the fuzzy counterpart of kM. It has been implemented in several functions in different R packages: we mention **cluster** (Maechler et al., 2017), **clue** (Hornik, 2005), **e1071** (Meyer et al., 2017), **skmeans** (Hornik et al., 2012), **vegclust** (De Caceres et al., 2010), **ppclust** (Cebeci et al., 2018) and **fclust** (Ferraro and Giordani, 2015). Among them, **fclust** offers a general toolbox for partitioning data using fuzzy clustering algorithms, computing cluster validity indices and visualizing clustering results. The current version (version 2.1.1) of the package has been deeply improved with respect to the previous ones. It contains several new routines and performance improvements. As a first improvement, all the functions have been rewritten in the C++ language using **Rcpp** (Eddelbuettel, 2013) and **RcppArmadillo** (Eddelbuettel and Sanderson, 2014) to enhance their computational efficiency. In addition, all the clustering algorithms can automatically select the optimal number of clusters using the cluster validity indexes available in the package. All the functions usually require data organized by quantitative variables and observations (object data). To increase the usability of the package, another relevant improvement is the implementation of fuzzy clustering algorithms for relational data (Davé and Sen, 2002). Relational data come from measures of dissimilarity between observations. Two clustering methods proposed by Davé and Sen (2002) have been considered. They do not require any assumption about the adopted measure of dissimilarity. Thus, such algorithms can be applied to a wide range of data. Specifically, whilst all the functions for object data require quantitative variables, those for relational data can handle all kinds of data (quantitative, qualitative or mixed) provided that a suitable measure of dissimilarity/distance is selected, e. g. the Gower distance (Gower, 1971) specifying the option `metric="gower"` of the function `daisy` in the package **cluster**. Finally, new functions to compare two partitions in a fuzzy environment have been implemented. Namely, the fuzzy versions of the Rand index (RI; Rand, 1971), the adjusted Rand index (ARI; Hubert and Arabie, 1985), and the Jaccard index (Jaccard, 1901; Halkidi et al., 2001) have been developed by Campello (2007). The standard indexes are implemented in different packages (see, for instance, Scrucca et al., 2016; Chung et al., 2018). The fuzzy vartiants are now available in **fclust**.

In this paper, after reviewing the most relevant fuzzy clustering methods, we recall some of the original functions to present the new improvements and we introduce the new functions by examples. We assume that the latest version of **fclust** available on CRAN is already installed with

```
> install.packages("fclust")
```

and loaded into the R session using

```
> library(fclust)
```

Fuzzy clustering

In this section, we recall the fuzzy k-means algorithm (Bezdek, 1981) and its extension suggested by Gustafson and Kessel (1979). Whilst the former detects spherical clusters, the latter allows for clusters with ellipsoidal shape. Then, a fuzzy clustering algorithm for relational data is described (Davé and Sen, 2002)

Fuzzy k-means algorithm

The most known and used fuzzy clustering algorithm is the fuzzy k-means (FkM) (Bezdek, 1981). The FkM algorithm aims at discovering the best fuzzy partition of n observations into k clusters by solving the following minimization problem:

$$\begin{aligned} \min_{\mathbf{U},\mathbf{H}} J_{\mathrm{F}k\mathrm{M}} &= \sum_{i=1}^{n} \sum_{g=1}^{k} u_{ig}^{m} d^{2}\left(\mathbf{x}_i, \mathbf{h}_g\right), \\ \text{s.t.} \quad u_{ig} &\in [0,1], \sum_{g=1}^{k} u_{ig} = 1, \end{aligned} \tag{H.2.1}$$

where $d\left(\mathbf{x}_i, \mathbf{h}_g\right)$ is the Euclidean distance. In (H.2.1), u_{ig} is the generic element of the membership degree matrix $\mathbf{U}$ of order $(n \times k)$, taking values in the interval [0,1] and representing the membership degree of observation i to cluster g. The row-wise sums of $\mathbf{U}$ are equal to 1. The propotypes (centroids) $\mathbf{h}_g = [h_{g1}, h_{g2}, \cdots, h_{gp}]$ $(g = 1, \cdots, k)$ are stored in the matrix $\mathbf{H}$ of order $(k \times p)$, being p the number of variables. Finally, the parameter m tunes the fuzziness of the obtained solution.

Gustafson-Kessel extensions of the FkM algorithm

The FkM algorithm, as the standard k-Means, can determine only spherical shaped clusters. This depends on the use of the Euclidean distance to measure the dissimilarity between observations. This limits its applicability when non-spherical clusters are expected. In order to overcome this drawback, Gustafson and Kessel (1979) extend the FkM algorithm by replacing the Euclidean distance with a cluster-specific Mahalanobis distance:

$$d_M^2\left(\mathbf{x}_i, \mathbf{h}_g\right) = \left(\mathbf{x}_i - \mathbf{h}_g\right)^{'} \mathbf{M}_g \left(\mathbf{x}_i - \mathbf{h}_g\right), \tag{H.2.2}$$

where $\mathbf{M}_g$ is a symmetric and positive-definite matrix of order p. When $\mathbf{M}_g = \mathbf{I}$, (H.2.2) is equvalent to the Euclidean distance. The Gustafson-Kessel FkM (briefly, GK-FkM) consists in solving the following minimizatin problem

$$\begin{aligned} \min_{\mathbf{U},\mathbf{H},\mathbf{M}_1,\ldots,\mathbf{M}_k} J_{\text{GK-F}k\text{M}} &= \sum_{i=1}^{n} \sum_{g=1}^{k} u_{ig}^{m} d_M^{2}\left(\mathbf{x}_i, \mathbf{h}_g\right), \\ \text{s.t.} \quad u_{ig} &\in [0,1], \sum_{g=1}^{k} u_{ig} = 1, |\mathbf{M}_g| = \rho_g. \end{aligned} \tag{H.2.3}$$

The constraints in (H.2.3) are similar to those in (H.2.1) except for the new ones on $\mathbf{M}_g$ ($|\mathbf{M}_g| = \rho_g > 0$, with ρ_g fixed for each g), added to avoid a trivial solution with $\mathbf{M}_g = \mathbf{0}$, that would be obtained since $J_{\text{GK-F}k\text{M}}$ is linear in $\mathbf{M}_g$.

For the generic g-th cluster, the iterative solution of $\mathbf{M}_g$ is $[\rho_g|\mathbf{V}_g|]^{\frac{1}{n}}\mathbf{V}_g^{-1}$, where $\mathbf{V}_g$ is the fuzzy covariance matrix for cluster g, defined as

$$\mathbf{V}_g = \frac{\sum_{i=1}^{n} u_{ig}^m (\mathbf{x}_i - \mathbf{h}_g)(\mathbf{x}_i - \mathbf{h}_g)'}{\sum_{i=1}^{n} u_{ig}^m}, \quad g = 1, \ldots, k. \tag{H.2.4}$$

The eigenvalues and eigenvectors of $\mathbf{V}_g$ describe the shape and orientation of the g-th cluster. When an eigenvalue is equal to 0 or when the condition number of $\mathbf{V}_g$ (i. e. the ratio between its maximum and minimum eigenvalue) is very large, the matrix is nearly singular, hence $\mathbf{V}_g^{-1}$ cannot be calculated. The condition $|\mathbf{V}_g| = \rho_g$ cannot overcome this drawback, as the determinant becomes 0. Babuska et al. (2002) propose to avoid these numerical problems by constraining the condition number of $\mathbf{V}_g$ to be smaller than a predefined threshold. Since this might lead to overfit the data, the update of $\mathbf{V}_g$ can be regularized by considering the covariance matrix of the whole dataset. See, for more details, Babuska et al. (2002).

Fuzzy clustering algorithms for relational data

In practical applications it may occur that only relational data are available. Relational data consist in the pairwise relations (similarities or dissimilarities) between observations, stored in a square matrix, say $\mathbf{D}$, not necessarily based on object data. In fact, $\mathbf{D}$ can be built either by computing the dissimilarity (or similarity) between all the pairs of observations on which a set of variables are collected (indirect relational data) or according to subjective expert knowledge, e. g. a teacher expresses her/his subjective degrees of dissimilarity for all the pair of pupils in her/his classroom (direct relational data). In the latter case, fuzzy clustering algorithms for object data can no longer be applied. In the former case, fuzzy clustering algorithms for object data should be preferred to those for relational data due to their computational efficiency. Nonetheless, fuzzy clustering algorithms usually assume to deal with quantitative variables preventing their applicability in case of qualitative/mixed variables. In such a case, fuzzy clustering methods for relational data can be fruitfully applied provided that a suitable dissimilarity measure for qualitative/mixed variables is used.

In the literature, there exist several proposals of fuzzy clustering algorithms for relational data. Among them, a valuable choice is represented by the class of algorithms proposed by Davé and Sen (2002), which are suitable for all kinds of dissimilarity. We assume that $\mathbf{D}$ is a dissimilarity matrix. If it contains similarities, these should be converted into dissimilarities. For this purpose, e. g. the function `sim2diss` of the package **smacof** (de Leeuw and Mair, 2009) can be used. The non-Euclidean fuzzy relational data clustering algorithm (NEFRC) consists in solving the following minimization problem:

$$\min_{\mathbf{U}} J_{\text{NEFRC}} = \sum_{g=1}^{k} \frac{\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} u_{ig}^m u_{jg}^m d(\mathbf{x}_i,\mathbf{x}_j)}{2\sum\limits_{t=1}^{n} u_{tg}^m},$$

$$\text{s.t.} \quad u_{ig} \in [0,1]\,, \sum_{g=1}^{k} u_{ig} = 1.$$

Notice that the NEFRC algorithm differs from the famous FANNY algorithm proposed by Kaufman and Rousseeuw (1990) since a general fuzzifier m is used and it is suitable for all kinds of dissimilarity.

The package also offers a robust variant of NEFRC involving the concept of noise cluster. It is an additional cluster such that the outliers are assigned to it with high membership degrees. It is not a cluster in a strict sense because the outliers are not necessarily similar to each other. Its role is that the membership degrees of the outliers to the standard clusters tend to be low without affecting the obtained partition. The robust version of NEFRC has been implemented in the current version of **fclust** and represents the relational counterpart of the FkM algorithm with noise cluster, already available in the package.

The package

In this section we present the main features of the package **fclust** with particular emphasis on the more recent updates. The list of algorithms with the corresponding functions is reported in Table 1. Apart from some peculiarities, all the available functions in the package require the same input arguments, involving the set-up of the clustering algorithms, i. e. number of starts, convergence criterion, data standardization. The user is not requested to specify such arguments because default options are specified. Obviously, the dataset to be clustered must be given.

Differently from the previous versions, the number of groups `k` is no longer required. Of course, the user can select the integer value of `k`, otherwise the optimal number of clusters is automatically chosen by maximizing or minimizing one of the available fuzzy cluster validity indices (see Table 2) to be specified in the option `index` (default `"SIL.F"`). By default the possible number of clusters is in the vector `k=2:6`, unless a different integer vector is selected by the user.

A fast way to apply one of the available algorithms is represented by the function `Fclust`:

```
> Fclust (X, k, type, ent, noise, stand, distance)
```

In `Fclust` to choose a given algorithm, the options `type`, `ent`, `noise` and `distance` should be set. `type` is a character string specifying the type of algorithm to be used. The currently available options are `"standard"` (the default option for FKM-type algorithms, provided that `distance = FALSE`), `"polynomial"`, `"gk"`, `"gkb"`, `"medoids"`. `ent` (default `FALSE`) and `noise` (default `FALSE`) are logical values indicating, respectively, whether the entropy regularization and the noise cluster should be used. Morever, `distance` (default `FALSE`) is another logical value indicating whether the data in `X` are distances/dissimilarities. When `distance = TRUE`, `type` is constrained to be `"standard"` and NEFRC-type algorithms are run. Finally, `stand` is used for standardization (default: no standardization) and `k` indicates the desired number of clusters (only for this function, the default value is `2`). For instance, the researcher interested in applying the FkM algorithm with noise cluster with $k = 3$ clusters to `X` can digit:

```
> Fclust (X = X, k = 3, type = "standard", noise = TRUE)
```

In the following we are going to present the main features and improvements of the package by considering the standard FkM algorithm (function `FKM`), the Gustafson–Kessel extension of FkM according to the Babuska et al. (2002) variant (function `FKM.gkb`), and the clustering algorithm for relational data (function `NEFRC`).

Fuzzy k-means (`FKM`)

The `FKM` function is applied to the NBA dataset available in **fclust**. The dataset contains some statistics on 30 NBA teams for the regular season 2017-2018 (source: https://stats.nba.com/teams/traditional/): number of wins (`W`), field goals made (`FGM`), field goals attempted (`FGA`), field goal percentage (`FGP`), 3 point field goals made (`3PM`), 3 point field goals attempted (`3PA`), 3 point field goals percentage (`3PP`), free throws made (`FTM`), free throws attempted (`FTA`), free throw percentage (`FTP`), offensive rebounds (`OREB`), defensive rebounds (`DREB`), assists (`AST`), turnovers (`TOV`), steals (`STL`), blocks (`BLK`), blocked field goal attempts (`BLKA`), personal fouls (`PF`), personal fouls drawn (`PFD`) and points (`PTS`). In addition, two more variables are available indicating the conference (`Conference`) and the playoff appearance (`playoff`). Both the variables are objects of class `factor` with two levels.

The dataset can be loaded as following:

```
> data("NBA")
```

A subset of variables is considered for clustering purposes. The raw values of field goals, point field goals and free throws are removed (only the percentage values are considered), as well as the wins and the personal fouls.

```
> X.NBA <- NBA[,c(4,7,10,11,12,13,14,15,16,17,20)]
```

We apply the function `FKM` to the obtained dataset. The parameter of fuzziness `m` is set to `m = 1.2` (the default value `m = 2` was too high producing an extremely fuzzy partition with membership degrees not far from 0.5) and the number of starts is fixed to 50 (`RS = 50`) to avoid local optima. The number of clusters is automatically selected using the fuzzy silhouette index (`index = "SIL.F"`). Notice that the fuzzy silhouette index represents a fuzzy extension of the well-known silhouette (Kaufman and Rousseeuw, 1990) involving the use of the membership degree information (for further details, refer to Campello, 2007). Finally, we set `stand = 1` in order to standardize the data before running `FKM`:

```
> fkm.NBA <-  FKM(X = X.NBA, m = 1.2, RS = 50, stand = 1, index = "SIL.F")
```

The `summary` method returns the most relevant information:

```
> summary(fkm.NBA)
```

```
 Fuzzy clustering object of class 'fclust' 

 Number of objects:
 30

 Number of clusters:
 2

 Cluster sizes:
Clus 1 Clus 2 
    18     12 


 Clustering index values:
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6 
0.2994904 0.2508281 0.2558217 0.2586680 0.2700120 


 Closest hard clustering partition:
      Houston Rockets       Toronto Raptors  Golden State Warriors 
                    2                     2                      2 
       Boston Celtics    Philadelphia 76ers    Cleveland Cavaliers 
                    1                     2                      2 
Portland Trail Blazers        Indiana Pacers   New Orleans Pelicans 
                    1                     2                      2 
 Oklahoma City Thunder             Utah Jazz Minnesota Timberwolves 
                    1                     2                      2 
    San Antonio Spurs        Denver Nuggets             Miami Heat 
                    1                     2                      1 
      Milwaukee Bucks    Washington Wizards            LA Clippers 
                    2                     2                      1 
      Detroit Pistons     Charlotte Hornets     Los Angeles Lakers 
                    1                     1                      1 
      New York Knicks         Brooklyn Nets          Chicago Bulls 
                    1                     1                      1 
     Sacramento Kings         Orlando Magic          Atlanta Hawks 
                    1                     1                      1 
     Dallas Mavericks     Memphis Grizzlies           Phoenix Suns 
                    1                     1                      1 

 Cluster memberships:
  Clus 1 
  [1] "Boston Celtics"         "Portland Trail Blazers"
 [3] "Oklahoma City Thunder"  "San Antonio Spurs"     
 [5] "Miami Heat"             "LA Clippers"           
 [7] "Detroit Pistons"        "Charlotte Hornets"     
 [9] "Los Angeles Lakers"     "New York Knicks"       
[11] "Brooklyn Nets"          "Chicago Bulls"         
[13] "Sacramento Kings"       "Orlando Magic"         
[15] "Atlanta Hawks"          "Dallas Mavericks"      
[17] "Memphis Grizzlies"      "Phoenix Suns"          
 Clus 2 
  [1] "Houston Rockets"        "Toronto Raptors"       
 [3] "Golden State Warriors"  "Philadelphia 76ers"    
 [5] "Cleveland Cavaliers"    "Indiana Pacers"        
 [7] "New Orleans Pelicans"   "Utah Jazz"             
 [9] "Minnesota Timberwolves" "Denver Nuggets"        
[11] "Milwaukee Bucks"        "Washington Wizards"    

 Number of objects with unclear assignment (maximal membership degree <0.5):
 0

 Membership degree matrix (rounded):
```

```
                       Clus 1 Clus 2
Houston Rockets          0.02   0.98
Toronto Raptors          0.01   0.99
Golden State Warriors    0.02   0.98
Boston Celtics           0.92   0.08
Philadelphia 76ers       0.11   0.89
Cleveland Cavaliers      0.05   0.95
Portland Trail Blazers   0.95   0.05
Indiana Pacers           0.34   0.66
New Orleans Pelicans     0.00   1.00
Oklahoma City Thunder    0.78   0.22
Utah Jazz                0.14   0.86
Minnesota Timberwolves   0.12   0.88
San Antonio Spurs        0.77   0.23
Denver Nuggets           0.03   0.97
Miami Heat               1.00   0.00
Milwaukee Bucks          0.03   0.97
Washington Wizards       0.03   0.97
LA Clippers              0.96   0.04
Detroit Pistons          1.00   0.00
Charlotte Hornets        0.98   0.02
Los Angeles Lakers       0.93   0.07
New York Knicks          0.96   0.04
Brooklyn Nets            0.99   0.01
Chicago Bulls            1.00   0.00
Sacramento Kings         0.98   0.02
Orlando Magic            1.00   0.00
Atlanta Hawks            0.98   0.02
Dallas Mavericks         0.97   0.03
Memphis Grizzlies        0.99   0.01
Phoenix Suns             0.99   0.01

 Cluster summary:
       Cl.size Min.memb.deg. Max.memb.deg. Av.memb.deg. N.uncl.assignm.
Clus 1      18          0.77             1         0.95               0
Clus 2      12          0.66             1         0.92               0

 Euclidean distance matrix for the prototypes (rounded):
       Clus 1
Clus 2   2.91

 Available components:
 [1] "U"         "H"         "F"         "clus"      "medoid"
 [6] "value"     "criterion" "iter"      "k"         "m"
[11] "ent"       "b"         "vp"        "delta"     "stand"
[16] "Xca"       "X"         "D"         "call"
```

According to `SIL.F`, we select the solution with $k = 2$ clusters. The obtained clusters can be plotted on the plane spanned by the first two principal components. This can be done by using the method `plot` associated to an **fclust** object specifying the option `pca = TRUE`.

We can see that the first component is able to distinguish the two clusters. Teams with high first component scores belong to Cluster 2 and those with low scores to Cluster 1. The first component loadings are (the script is omitted):

```
  FGP    3PP    FTP   OREB   DREB    AST    TOV    STL    BLK   BLKA
0.455  0.305  0.278 -0.157  0.158  0.395  0.071  0.160  0.370 -0.338
  PTS
0.369
```

Hence, it appears that the clusters are mainly distinguish in terms of `FGP`, `AST`, `BLK`, `PTS`, `BLKA`, `3PP` and `FTP`, i. e. the variables with the highest first component loadings in absolute value.

In order to interpret the clusters, we inspect the prototypes. To this purpose, we apply the function `Hraw` to visualize the prototypes by using the original units of measurement.

```
> plot(fkm.NBA, pca = TRUE)
```

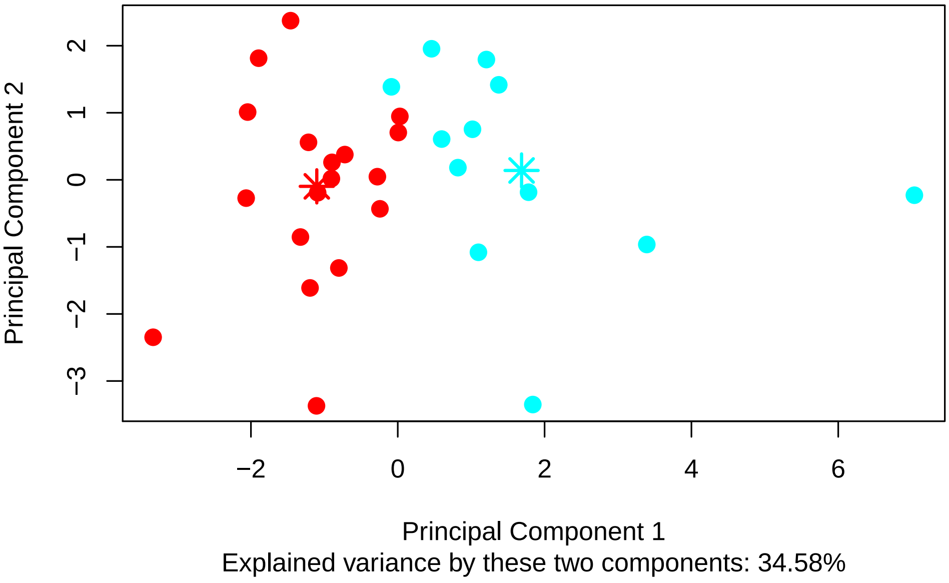


Figure 2: Scatterplot of the NBA teams on the plane spanned by the first two principal components. Points are marked according to the obtained partition (Cluster 1: red, Cluster 2: cyan).

```
> round(Hraw(X = X.NBA, H = fkm.NBA$H), 3)

         FGP   3PP   FTP  OREB   DREB    AST    TOV   STL   BLK  BLKA
Clus 1 0.451 0.358 0.759 9.830 33.757 22.483 14.252 7.446 4.581 4.996
Clus 2 0.474 0.367 0.780 9.513 33.891 24.425 14.305 8.096 5.145 4.548
           PTS
Clus 1 104.362
Clus 2 109.357
```

We can see that Cluster 2 recognizes the best teams. In fact, the values of the prototype of Cluster 2 are better than the corresponding ones of Cluster 1, except for a few variables such as OREB and BLKA. To further characterize the obtained clusters, we consider the variables Conference and Playoff. In particular, we aim at discovering whether the two clusters can be interpreted in terms of the geographical location and/or the playoff appearance. From a statistical point of view, this consists in comparing the fuzzy partition resulting from FKM with the hard ones corresponding to the classification variables Conference or Playoff. For this purpose, the fuzzy cluster similarity measures available in the package are considered. Such measures, proposed by Campello (2007), are summarized in Table 3.

To report the values of the three measures, the function Fclust.compare can be used. The input required by Fclust.compare (and similarly for RI.F, ARI.F and JACCARD.F) is a fuzzy membership degree matrix (U) and a vector of class labels (VC).

```
> Fclust.compare(VC = NBA$Playoff, U = fkm.NBA$U)

    ARI.F      RI.F JACCARD.F
0.3077549 0.6537339 0.4825140

> Fclust.compare(VC = NBA$Conference, U = fkm.NBA$U)

      ARI.F        RI.F   JACCARD.F
-0.02547957  0.48701485  0.31724090
```

It is clear that the clusters cannot be interpreted from a geographical point of view, whilst, to some extent, they are related to the playoff appearance. Such a comment holds especially for Cluster 2. In fact, 11 out of 12 teams belonging to Cluster 2 reached the playoff stage. The only exception is Denver Nuggets, which was one of the best teams in terms of number of wins (W) but not qualified to the playoff stage, because the number of wins was not sufficient to reach the playoff stage in the Western conference.

Gustafson-Kessel extensions of the FKM algorithm (FKM.gk and FKM.gkb)

The Gustafson-Kessel extension of the FkM algorithm is implemented in the functions FKM.gk and FKM.gkb. The former implements the GK-FkM algorithm in the original proposal, whilst the latter, recently added to the package, considers the computational improvement suggested by Babuska et al. (2002). A simulated dataset similar to the one in Babuska et al. (2002) is used to show the differences between the two functions. Three different clusters with different size (100, 80, and 60) in two-dimensional space are generated as follows:

$$y = \begin{cases} 6 - 2.0x & \text{with } x \sim U(1,3) \text{ for Cluster 1,} \\ -5 + 1.5x & \text{with } x \sim U(3.2,6) \text{ for Cluster 2,} \\ 3x & \text{with } x \sim U(-1,1) \text{ for Cluster 3.} \end{cases} \tag{H.3.1}$$

Data can be found in **fclust** and loaded with the following command:

```
> data(synt.data2)
```

Figure 3 shows the scatterplot of the simulated data. In this case the cluster covariance matrices are singular because the two variables are perfectly collinear in all the clusters.

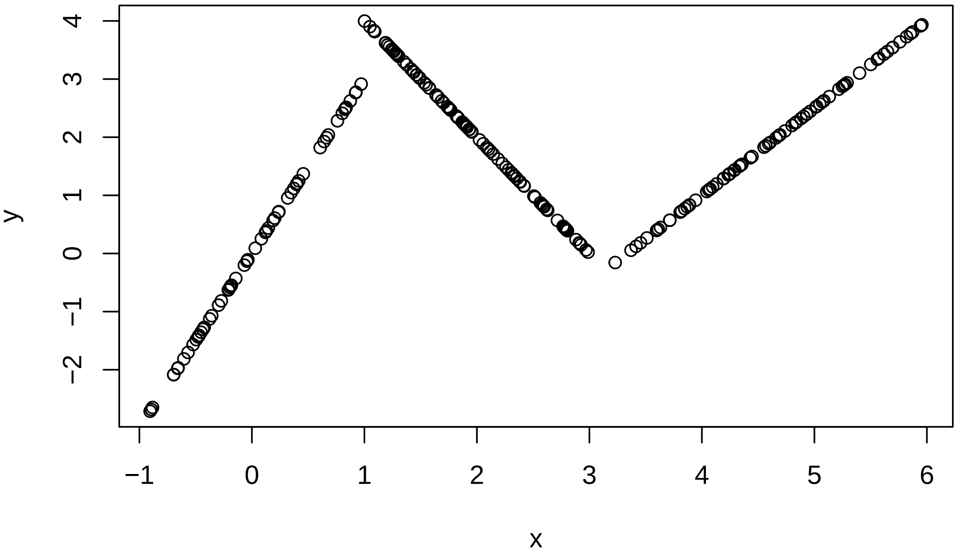


Figure 3: Scatterplot of the synt.data2 dataset.

By employing the standard function FKM.gk numerical problems occur. By setting m = 2, k = 3 and RS = 1, we have:

```
> fkm.gk.synt <- FKM.gk(X = synt.data2, k = 3, RS = 1)
```

The following warning message appears:

```
Warning message:
In FKM.gk(X = synt.data2, k = 3, RS = 1) :
  When k=3, at least one cluster covariance matrix seems to be singular.
  Increase the number of starts RS or use FKM.gkb
```

Thus, we can see that the algorithm stops because at least one cluster covariance matrix is singular. In this case, the function returns the standard object of class fclust containing the sub-optimal solution at the previous iteration, i. e. the one with no singular cluster covariance matrices. By studying the number of iterations and the loss function value of such a local optimum solution, we get:

```
> fkm.gk.synt$iter
```

```
Start 1
     13

> fkm.gk.synt$value

   Start 1
0.06044555
```

For comparative purpose, we run the recommended function FKM.gkb using the same start:

```
> fkm.gkb.synt <- FKM.gkb(X = synt.data2, k = 3, RS = 1, seed = 123)


> fkm.gkb.synt$iter

Start 1
     16


> fkm.gkb.synt$value

     Start 1
1.482029e-05
```

The method required two more iterations for convergence. The obtained solution is characterized by a lower loss function value and is not affected by singularity problems.

Fuzzy clustering for indirect relational data (dichotomous variables)

The NEFRC algorithm can be applied using the function NEFRC. Differently from the other functions for clustering object data, it requires distances/dissimilarities as input argument. Consistently with the other functions, the available clustering indices (except for the Xie and Beni one) can be used to select the optimal number of clusters k. In particular, the silhouette index (SIL) and its fuzzy extension (SIL.F) have been rearranged for relational data. Specifically, the input of NEFRC is employed to compute the silhouette and the fuzzy silhouette indices. This is the default option when SIL.F is called by NEFRC. In order to use the distance/dissimilarity matrix for computing the fuzzy silhouette index, the option distance = TRUE in SIL.F should be set. Generally speaking, the fuzzy silhouette index can be applied for any kind of data (quantitative or qualitative or mixed) provided that a suitable distance/dissimilarity matrix is used as input.

The function NEFRC is presented by considering the congressional voting records data (Schlimmer, 1987) available in **fclust**. The data collect 1984 United Stated voting records for 475 U.S. House of Representative congressmen on 16 key votes identified by the Congressional Quartely Almanac (CQA). The congressmen are split into Democrats and Republicans (variable class). The 16 key votes are objects of class factor with three levels according to the CQA scheme: y refers to the types of votes "voted for", "paired for" and "announced for"; n to "voted against", "paired against" and "announced against"; yn to "voted present", "voted present to avoid conflict of interest" and "did not vote or otherwise make a position known".

The dataset can be loaded as follows:

```
> data("houseVotes")
```

It contains the following variables:

```
> colnames(houseVotes)

 [1] "class"                                   "handicapped-infants"
 [3] "water-project-cost-sharing"              "adoption-of-the-budget-resolution"
 [5] "physician-fee-freeze"                    "el-salvador-aid"
 [7] "religious-groups-in-schools"             "anti-satellite-test-ban"
 [9] "aid-to-nicaraguan-contras"               "mx-missile"
[11] "immigration"                             "synfuels-corporation-cutback"
[13] "education-spending"                      "superfund-right-to-sue"
[15] "crime"                                   "duty-free-exports"
[17] "export-administration-act-south-africa"
```

Since the level yn might indicate unknown preferences for some key votes, these values are considered as missing and, therefore, the rows with at least one yn value are removed:

```
> level.drop <- droplevels(houseVotes, exclude = "yn")
> houseVotesComplete <- level.drop[complete.cases(level.drop),]
```

The research interest relies in discovering whether a two-cluster structure exists and a relationship between the political position and the system of voting emerges. Even if the dataset is not relational, NEFRC is the only one R routine for getting a fuzzy partition based on qualitative variables. For this purpose, the Gower distance, implemented in the function daisy of the package **cluster**, is used to generate the dissimilarity matrix:

```
> X.houseVotesComplete <- houseVotesComplete[,-1]
> library(cluster)
> D.houseVotes <- daisy(x = X.houseVotesComplete, metric = "gower")
```

The standard algorithm for relational data is employed by running the function NEFRC setting m = 1.5 and k = 2 in order to assess whether the clusters are related to the parties (class).

```
> nefrc.houseVotes <- NEFRC(D = D.houseVotes, k  = 2, m = 1.5, index = "SIL.F")
```

The summary method is similar to that of FKM and hence not reported. The two clusters can be interpreted in terms of the parties. In fact, we get the following cluster similarity measures:

```
> Fclust.compare(VC = houseVotesComplete$class, U = nefrc.houseVotes$U)

    ARI.F      RI.F JACCARD.F
0.4871095 0.7435544 0.5914710
```

Morevover, we have:

```
> table(nefrc.houseVotes$clus[,1], houseVotesComplete$class)


    democrat republican
  1       19        101
  2      105          7
```

Therefore, Cluster 1 and Cluster 2 refer to the Republicans and Democrats, respectively. In Figure 4 the clusters are plotted in the low dimensional space spanned by the first two components. Note that the plot method for relational data is based on non-metric multidimensional scaling (Kruskal, 1964) by calling the function isoMDS of the package **MASS** (Venables and Ripley, 2002).

```
> plot(nefrc.houseVotes)
```

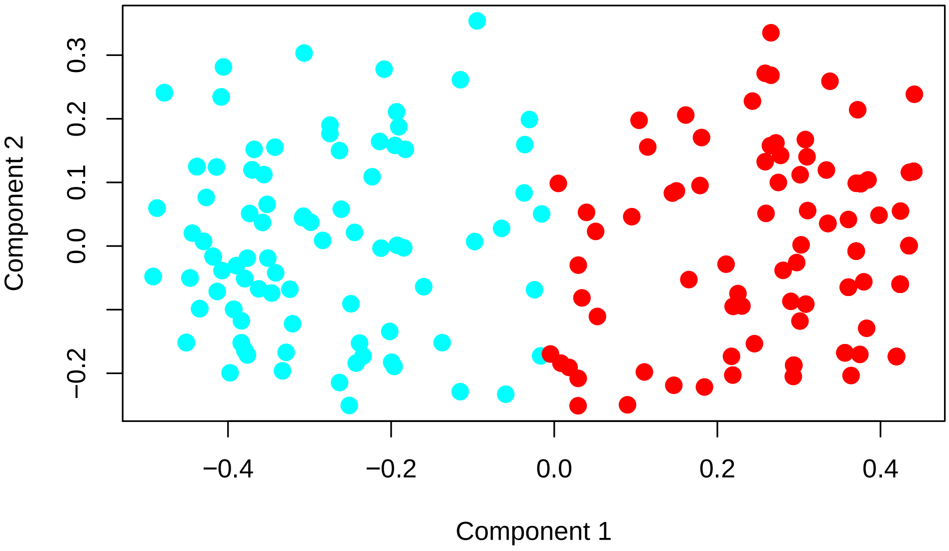


Figure 4: Scatterplot of relational data with plot method. Points are marked according to the obtained classification (Cluster 1: red, Cluster 2: cyan).

To further interpret the clusters, Figure 5 displays the barplots of the 16 key votes for the two clusters (by considering the closest hard clustering partition). We can observe that the votes are highly connected with the Congressmen political positions. This holds for almost all the 16 key votes with particular reference to, e. g. "adoption-of-the-budget", "education-spending" and "anti-satellite-test-ban".

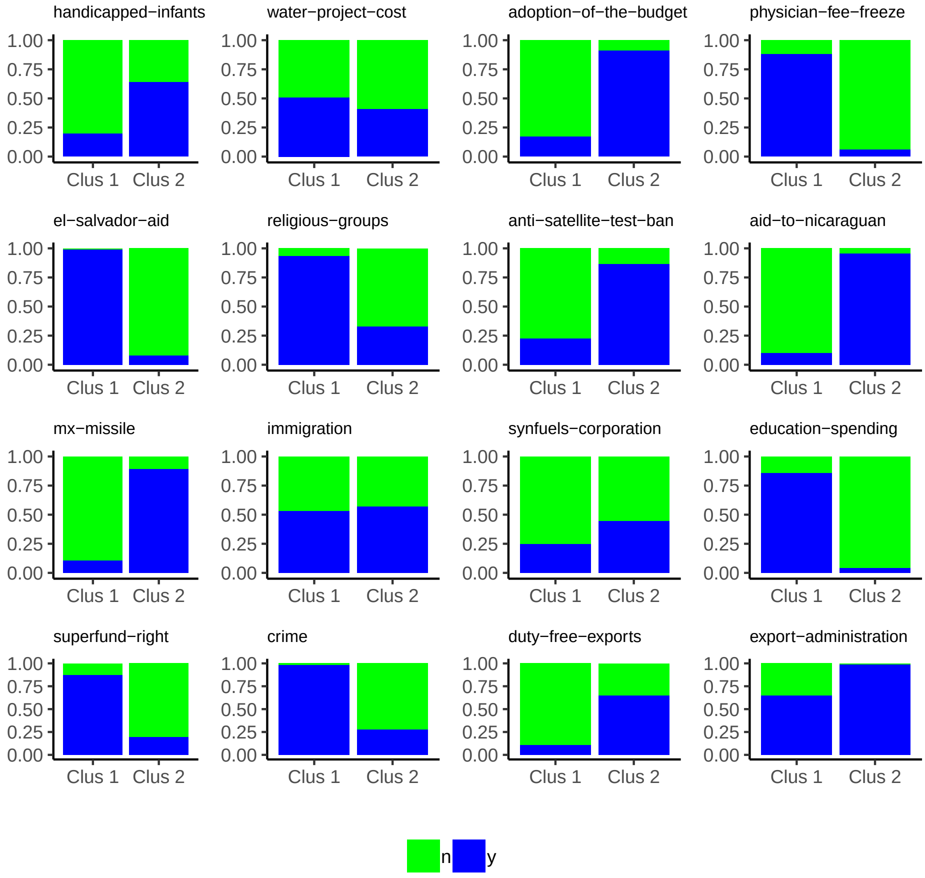


Figure 5: Barplot of the 16 key votes for the two clusters (n: green, y: blue).

Fuzzy clustering for indirect relational data (ordinal variables)

In this section, a dataset with ordinal data is analyzed by using NEFRC. The data refer to the Math Anxiety Scale Survey administered to 20 students in a statistics course (Bai et al., 2009). In the survey, each student answers 14 questions by using a Likert scale with five levels ("Strongly Disagree", "Disagree", "Neutral", "Agree", "Strongly Agree"). First, we load the dataset:

```
> library(likert)
> data("mass")
```

Then, we compute the dissimilarity matrix by using the Gower distance. When applied to ordinal variables, such a distance is based on ranks. Note that the first variable of mass is Gender, not useful for clustering purposes and, thus, omitted in the computation of the dissimilarity matrix. We have:

```
> library(cluster)
> D.mass <- daisy(x = mass[,-1], metric = "gower")
```

Finally, we run the function NEFRC automatically selecting the number of clusters by means of SIL.F:

```
> nefrc.mass <- NEFRC(D = D.mass, index = "SIL.F")
```

The fuzzy silhouette values, employed to select the number of clusters, are:

```
> nefrc.mass$criterion
```

```
SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.5330319 0.4623684 0.4039311 0.4428360 0.4685703
```

Hence, $k = 2$ clusters are suggested. Since the default options are used, the solution could also be obtained by considering the function `Fclust`:

```
> nefrc.mass <- Fclust(X = D.mass,  k = 2, noise = FALSE, distance = TRUE)
```

The clusters can be interpreted according to the observed qualitative variables. For this purpose, we calculate the p-values resulting from the χ^2 tests by which we study the independence between the closest hard clustering partition and every observed variables. The p-values are stored in the vector `PV`:

```
> PV <- rep(NA,ncol(mass))
> for (j in 1:ncol(mass)) PV[j] <- chisq.test(nefrc.mass$clus[,1], mass[,j])$p.value
```

At the significance level $\alpha = 0.05$, we are interested in those variables such that the corresponding p-value is lower than α:

```
> alpha <- 0.05
> names(mass)[PV < alpha]

 [1] "I find math interesting."
 [2] "I get uptight during math tests."
 [3] "Mind goes blank and I am unable to think clearly when doing my math test."
 [4] "I worry about my ability to solve math problems."
 [5] "I get a sinking feeling when I try to do math problems."
 [6] "I find math challenging."
 [7] "Mathematics makes me feel nervous."
 [8] "Mathematics makes me feel uneasy."
 [9] "Math is one of my favorite subjects."
[10] "I enjoy learning with mathematics."
[11] "Mathematics makes me feel confused."
```

We inspect the contingency tables (not reported here) between such a subset of observed variables and the closest hard clustering partition and we find that Cluster 1 is characterized by large frequencies for the modalities `"Strongly Disagree"` and `"Disagree"` with respect to the variables `"I find math interesting."`, `"Math is one of my favorite subjects."` and `"I enjoy learning with mathematics."` and large frequencies for the modalities `"Agree"` and `"Strongly Agree"` with respect to the variables `"I get uptight during math tests."`, `"Mind goes blank and I am unable to think clearly when doing my math test."`, `"I worry about my ability to solve math problems."`, `"I get a sinking feeling when I try to do math problems."`, `"I find math challenging."`, `"Mathematics makes me feel nervous."`, `"Mathematics makes me feel uneasy."` and `"Mathematics makes me feel confused."`. Of course, the opposite comment holds for Cluster 2. Therefore, the partition distinguishes the students liking math (assigned to Cluster 2) from those who experience feelings of stress when faced with math (assigned to Cluster 1).

Fuzzy clustering for direct relational data

In the previous two subsections, `NEFRC` is applied in order to discover homogeneous clusters of observations on which qualitative variables are collected. In these cases, suitable dissimilarity matrices are built before running `NEFRC`. In the current subsection, we consider the case where variables are not available and the only information about the observations is expressed in terms of their dissimilarities or distances. The data are stored in the following object of class `dist`:

```
> library(smacof)
> data("FaceExp")
```

`FaceExp` contains the dissimilarities between pairs of 13 facial expressions related to particular stimuli:

```
> labels(FaceExp)

 [1] "Grief at death of mother"         "Savoring a Coke"
 [3] "Very pleasant surprise"           "Maternal love-baby in arms"
 [5] "Physical exhaustion"              "Something wrong with plane"
 [7] "Anger at seeing dog beaten"       "Pulling hard on seat of chair"
 [9] "Unexpectedly meets old boyfriend" "Revulsion"
[11] "Extreme pain"                     "Knows plane will crash"
[13] "Light sleep"
```

The dissimilarities have been calculated in a psychological experiment where a set of subjects were invited to judge how much two pictures of emotional expressions differ. Thus, all the possible pairs of emotional expressions were compared by the subjects and the dissimilarities were derived. See, for further details, Abelson and Sermat (1962) .

By means of NEFRC the aim is to discover whether similar facial expressions are perceived by the subjects in connection with similar emotions intended by the stimuli. In this case, we do not know the number of clusters and, therefore, we determine it according to SIL.F.

```
> nefrc.FaceExp <- NEFRC(D = FaceExp, index = "SIL.F")
```

We find that $k = 3$ should be set:

```
> nefrc.FaceExp$criterion

SIL.F k=2 SIL.F k=3 SIL.F k=4 SIL.F k=5 SIL.F k=6
0.5298465 0.5929045 0.5470887 0.5436513 0.4003177
```

The interpretation of the clusters can be done by seeking a common feature for the facial expressions, i. e. the stimuli, assigned to the same cluster. We have:

```
> round(nefrc.FaceExp$clus[(nefrc.FaceExp$clus[,1] == 1), 2], 2)

Savoring a Coke   Very pleasant surprise  Maternal love-baby in arms
       0.64                0.85                       0.75
Pulling hard on seat    Unexpectedly meets old boyfriend
        0.59                          0.94

> round(nefrc.FaceExp$clus[(nefrc.FaceExp$clus[,1] == 2), 2], 2)

Grief at death of mother      Physical exhaustion    Revulsion
    0.79                           0.81                0.69
Extreme pain    Light sleep
    0.56            0.64


> round(nefrc.FaceExp$clus[(nefrc.FaceExp$clus[,1] == 3), 2], 2)

Something wrong with plane      Anger at seeing dog beaten     Knows plane will crash
       0.52                              0.93                           0.78
```

Cluster 1 groups pleasant stimuli with the only exception of "Pulling hard on seat of chair" for which the membership degree is however the lowest one (0.59). The facial expressions showing pain belong to Cluster 2. "Light sleep" is also assigned to the cluster. It follows that the subjects tend to associate such an expression with suffering. Finally, anxiety characterizes Cluster 3.

Conclusion

In this paper we have described the main features of the package **fclust**. **fclust** represents a toolbox for fuzzy cluster analysis. The functions in the package offer a wide range of fuzzy clustering algorithms, fuzzy cluster validity indices, measures of similarity for comparing hard and fuzzy partitions and visualization tools for fuzzy clustering results. Particular attention has been paid to the new improvements and implementations available in the current version of the package (version 2.1.1). First of all, the functions have been updated by using the C++ language, with a remarkable reduction in computation time. Furthermore, the package now includes some fuzzy clustering algorithms for relational data, allowing the user to perform a fuzzy clustering analysis when the variables are qualitative or mixed. In such cases, a dissimilarity matrix can be built by using the existing R functions (e. g. dist or daisy in the package **cluster**) and the available functions for relational data (NEFRC and NEFRC.noise) can then be applied. As far as we know, NEFRC and NEFRC.noise represent the first available R functions for fuzzy clustering of qualitative or mixed variables. All the functions have been revised in such a way that the number of clusters can be automatically selected. This might increase the computation time, but it is crucial in order to spread the use of fuzzy clustering methods especially for non-expert users. In this connection, the function Fclust for running the available algorithms using the default options and specifying the desired number of clusters is also offered.

Bibliography

R. P. Abelson and V. Sermat. Multidimensional scaling of facial expressions. *Journal of Experimental Psychology*, 63(6):546–554, 1962. URL http://dx.doi.org/10.1037/h0042280. [p224]

R. Babuska, P. J. Van der Veen, and U. Kaymak. Improved covariance estimation for gustafson-kessel clustering. In *Proceedings of the 2002 IEEE International Conference on Fuzzy Systems*, page 1081–1085, 2002. URL https://doi.org/10.1109/FUZZ.2002.1006654. [p214, 215, 219, 228]

H. Bai, L. Wang, W. Pan, and M. Frey. Measuring mathematics anxiety: Psychometric analysis of a bidimensional affective scale. *Journal of Instructional Psychology*, 36(3):185–193, 2009. URL https://eric.ed.gov/?id=EJ952267. [p222]

J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981. [p212, 213, 228]

R. J. Campello. A fuzzy extension of the Rand index and other related indexes for clustering and classification assessment. *Pattern Recognition Letters*, 28(7):833–841, 2007. URL https://dx.doi.org/10.1016/j.patrec.2006.11.010. [p212, 215, 218]

Z. Cebeci, F. Yildiz, A. T. Kavlak, C. Cebeci, and H. Onder. *ppclust: Probabilistic and Possibilistic Cluster Analysis*, 2018. URL https://CRAN.R-project.org/package=ppclust. R package version 0.1.1. [p212]

N. C. Chung, B. Miasojedow, M. Startek, and A. Gambin. *Jaccard: Test Similarity Between Binary Data Using Jaccard/Tanimoto Coefficients*, 2018. URL https://CRAN.R-project.org/package=jaccard. R package version 0.1.0. [p213]

R. N. Davé. Characterization and detection of noise in clustering. *Pattern Recognition Letters*, 12 (11):657–664, 1991. URL https://doi.org/10.1016/0167-8655(91)90002-4. [p228]

R. N. Davé and S. Sen. Robust fuzzy clustering of relational data. *IEEE Transactions on Fuzzy Systems*, 10(6):713–727, 2002. URL https://dx.doi.org/10.1109/TFUZZ.2002.805899. [p212, 213, 214, 228]

M. De Caceres, X. Font, and F. Oliva. The management of vegetation classifications with fuzzy clustering. *Journal of Vegetation Science*, 21:1138–1151, 2010. URL https://doi.org/10.1111/j.1654-1103.2010.01211.x. [p212]

J. de Leeuw and P. Mair. Multidimensional scaling using majorization: SMACOF in R. *Journal of Statistical Software*, 31(3):1–30, 2009. URL http://www.jstatsoft.org/v31/i03/. [p214]

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. [p104, 212]

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics & Data Analysis*, 71:1054–1063, 2014. URL https://dx.doi.org/10.1016/j.csda.2013.02.005. [p212]

M. B. Ferraro and P. Giordani. A new fuzzy clustering algorithm with entropy regularization. In *Proceedings of the 9th Scientific Meeting of the Classification and Data Analysis Group (CLADAG 2013)*, 2013. ISBN 9788867871179. [p228]

M. B. Ferraro and P. Giordani. A toolbox for fuzzy clustering using the r programming language. *Fuzzy Sets and Systems*, 279:1–16, 2015. URL https://dx.doi.org/10.1016/j.fss.2015.05.001. [p212]

J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871, 1971. URL https://doi.org/10.2307/2528823. [p212]

D. E. Gustafson and W. C. Kessel. Fuzzy clustering with a fuzzy covariance matrix. In *Proceedings of the 1978 IEEE Conference on Decision and Control Including the 17th Symposium on Adaptive Processes*, page 761–766, 1979. URL https://doi.org/10.1109/CDC.1978.268028. [p213, 228]

M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001. URL https://doi.org/10.1023/A:1012801612483. [p212]

J. A. Hartigan and M. A. Wong. Algorithm as 136: A K-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. URL https://doi.org/10.2307/2346830. [p211]

K. Hornik. A CLUE for CLUster Ensembles. *Journal of Statistical Software*, 14(12):1–25, 2005. URL https://doi.org/10.18637/jss.v014.i12. [p37, 212]

K. Hornik, I. Feinerer, M. Kober, and C. Buchta. Spherical *k*-means clustering. *Journal of Statistical Software*, 50(10):1–22, 2012. URL https://doi.org/10.18637/jss.v050.i10. [p212]

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. URL https://dx.doi.org/10.1007/BF01908075. [p212]

P. Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901. [p212]

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, New York, 1990. URL https://doi.org/10.2307/2532178. [p214, 215]

R. Krishnapuram, A. Joshi, O. Nasraoui, and L. Yi. Low-complexity fuzzy relational clustering algorithms for web mining. *IEEE Transactions on Fuzzy Systems*, 9(4):595–607, 2001. URL https://doi.org/10.1109/91.940971. [p228]

J. B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2): 115–129, 1964. URL https://doi.org/10.1007/BF02289694. [p221]

R.-P. Li and M. Mukaidono. A maximum-entropy approach to fuzzy clustering. In *Proceedings of 1995 IEEE International Conference on Fuzzy Systems*, page 2227–2232, 1995. URL https://doi.org/10.1109/FUZZY.1995.409989. [p228]

R.-P. Li and M. Mukaidono. Gaussian clustering method based on maximum-fuzzy-entropy interpretation. *Fuzzy Sets and Systems*, 102(2):253–258, 1999. URL https://doi.org/10.1016/S0165-0114(97)00126-7. [p228]

M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik. *Cluster: Cluster Analysis Basics and Extensions*, 2017. URL https://cran.r-project.org/web/packages/cluster. R package version 2.0.6. [p212]

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. [p212]

W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(33):846–850, 1971. URL https://dx.doi.org/10.2307/2284239. [p212]

E. H. Ruspini. Numerical methods for fuzzy clustering. *Information Sciences*, 2(3):319–350, 1970. URL https://doi.org/10.1016/S0020-0255(70)80056-1. [p211]

J. C. Schlimmer. *Concept Acquisition through Representational Adjustment*. Department of Information and Computer Science, University of California, Irvine, 1987. [p220]

L. Scrucca, M. Fop, T. B. Murphy, and A. E. Raftery. mclust 5: Clustering, Classification and Density Estimation Using Gaussian Finite Mixture Models. *The R Journal*, 8(1):289–317, 2016. URL https://journal.r-project.org/archive/2016/RJ-2016-021/index.html. [p213]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, New York, 2002. URL https://doi.org/10.1007/978-0-387-21706-2. [p221]

R. Winkler, F. Klawonn, F. Höppner, and R. Kruse. Fuzzy cluster analysis of larger data sets. In *Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design*, page 302–331. IGI Global, Hershey, 2009. URL https://doi.org/10.4018/978-1-60566-858-1.ch012. [p228]

R. Winkler, F. Klawonn, and R. Kruse. Fuzzy clustering with polynomial fuzzifier function in connection with *m*-estimators. *Applied and Computational Mathematics*, 10(1):146–163, 2011. [p228]

Maria Brigida Ferraro
Department of Statistical Sciences, Sapienza University of Rome
P.le Aldo Moro 5, 00185 Rome, Italy
ORCiD: 0000-0002-7686-5938
mariabrigida.ferraro@uniroma1.it

Paolo Giordani
Department of Statistical Sciences, Sapienza University of Rome
P.le Aldo Moro 5, 00185 Rome, Italy
ORCiD: 0000-0003-4091-3165
paolo.giordani@uniroma1.it

Alessio Serafini
Department of Statistical Sciences, Sapienza University of Rome
P.le Aldo Moro 5, 00185 Rome, Italy
ORCiD: 0000-0002-8579-5695
alessio.serafini@uniroma1.it

Function	Algorithm
`FKM`	standard FkM algorithm (Bezdek, 1981)
`FKM.ent`	FkM with entropy regularization (Li and Mukaidono, 1995, 1999)
`FKM.noise`	FkM with noise cluster (Davé, 1991)
`FKM.ent.noise`	FkM with entropy regularization and noise cluster (Li and Mukaidono, 1999; Davé, 1991)
`FKM.gk`	Gustafson and Kessel extension of FkM (Gustafson and Kessel, 1979)
`FKM.gk.ent`	Gustafson and Kessel extension of FkM with entropy regularization (Ferraro and Giordani, 2013)
`FKM.gk.noise`	Gustafson and Kessel extension of FkM with noise cluster (Gustafson and Kessel, 1979; Davé, 1991)
`FKM.gk.ent.noise`	Gustafson and Kessel extension of FkM with entropy regularization and noise cluster (Ferraro and Giordani, 2013; Davé, 1991)
`FKM.gkb`	Gustafson, Kessel and Babuska extension of FkM (Babuska et al., 2002; Gustafson and Kessel, 1979)
`FKM.gkb.ent`	Gustafson, Kessel and Babuska extension of FkM with entropy regularization

Function	Index
`PC`	partition coefficient
`MPC`	modified partition coefficient
`PE`	partition entropy
`XB`	partition entropy
`SIL`	(crisp) silhouette
`SIL.F`	fuzzy silhouette

Table 2: List of fuzzy cluster validity indices available in the package **fclust**.

Function	Index
`RI.F`	Fuzzy version of Rand index
`ARI.F`	Fuzzy version of adjusted Rand index
`JACCARD.F`	Fuzzy version of Jaccard index

Table 3: List of fuzzy cluster similarity measures available in the package **fclust**.

Nowcasting: An R Package for Predicting Economic Variables Using Dynamic Factor Models

by Serge de Valk, Daiane de Mattos and Pedro Ferreira

Abstract The **nowcasting** package provides the tools to make forecasts of monthly or quarterly economic variables using dynamic factor models. The objective is to help the user at each step of the forecasting process, starting with the construction of a database, all the way to the interpretation of the forecasts. The dynamic factor model adopted in this package is based on the articles from Giannone et al. (2008) and Banbura et al. (2011). Although there exist several other dynamic factor model packages available for R, ours provides an environment to easily forecast economic variables and interpret results.

Introduction

Important economic decisions are made based on current and future conditions. Oftentimes, the variables used to measure such conditions are not available even for the recent past. This is, for instance, the case with US GDP that is published 45 days after the end of the quarter. Similarly, Brazilian GDP is published with a 60-day lag. There is therefore a need for forecasting the current value of given variables. To this end, Giannone et al. (2008) proposed a statistical model that allows quarterly variables, such as US GDP, to be forecast using a large set of monthly variables released with different lags. GDP forecasts for the current quarter are, furthermore, updated whenever new information is available. Different central banks have shown interest in this methodology, among them the European Central Bank (Angelini et al., 2008; Bańbura and Rünstler, 2011; Van Nieuwenhuyze et al., 2008), and the central banks of Ireland (D'Agostino et al., 2008), New Zealand (Matheson, 2010) and Norway (Aastveit and Trovik, 2012).

Factor models are designed to summarize the variation contained in a large dataset into only a few variables (Stock and Watson, 2006). In Giannone et al. (2008), the authors show how to reduce the information contained in dozens of monthly time series into only two dynamic factors. These two estimated factors, which are initially monthly, are then transformed into quarterly factors and used in a regression against GDP. Various other authors, such as Chauvet (2001); Marcellino et al. (2003); Forni et al. (2004); Boivin and Ng (2006); D'Agostino et al. (2006); Banbura et al. (2011); Dahlhaus et al. (2015); Stock and Watson (2016), have explored Dynamic Factor Models (DFMs) in time series forecasting and found promising results.

Given the publication lag of many variables, such as GDP, we can either forecast past, current or future values. In order to differentiate between those types of forecasts we adopt the terminology used in Giannone et al. (2008) and Banbura et al. (2011). Backcasting refers to forecasting the value of a yet unpublished variable for a past period, while nowcasting will be with respect to the current period. By way of illustration, suppose we want to forecast the GDP for the 2^{nd} quarter of 2018. If the exercise is made during the 2^{nd} quarter of 2018, then the forecast is classified as nowcasting. However, if the current date is before the 2^{nd} quarter of 2018, then the term used is forecasting. Finally, if the date is after the 2^{nd} quarter of 2018 and the GDP has not yet been released, then the forecast is classified as backcasting.

The aim of the package **nowcasting** is to offer the tools for the R user to implement dynamic factor models. The different steps in the forecasting process and the associated functions within the package are based on the literature. We have chosen to divide the process into 4 main steps: 1) constructing a dataset; 2) defining the model's initiation parameters; 3) forecasting; 4) presenting results. This particular division will be maintained in most sections.

This brings us to the article's sections that are organized as follows: 1) the theoretical framework is introduced; 2) the functions of our package are presented; 3) working examples of how to nowcast Brazilian GDP and of the New York FED nowcasting are given; 4) and finally the last section concludes with some considerations.

Methodology

Dynamic Factor Model

Let $x_t = (x_{1,t}, x_{2,t}, ..., x_{N,t})^{'}$ be the vector representing N monthly time series transformed to satisfy the weak stationarity assumption. The general specification of the dynamic factor model is given by:

$$x_t = \mu + \Lambda f_t + \varepsilon_t \tag{J.2.1}$$

$$f_t = \sum_{i=1}^{p} A_i f_{t-i} + Bu_t, \quad u_t \sim i.i.d.N(0, I_q) \tag{J.2.2}$$

In equation (J.2.1), the variables x_t are expressed as a function of an intercept μ and r unobserved common factors f_t. Since all variables x will later be demeaned, one may drop the unconditional means μ. The variables x_t will be loaded into the unobserved factors f_t through Λ. Equation (J.2.2) imposes the structure of a VAR(p) process on the factors f_t. Both ε_t and u_t are normal, allowing the use of the Kalman Filter. Furthermore, the vector of idiosyncratic component ε_t is unrelated to u_t at all lags, i.e., $E[\epsilon_t u'_{t-k}] = 0$ for any k. An interesting feature of equation (J.2.2) is that the number of shocks q to the factors need not be equal to the number of factors r. Structural breaks or lead/lag relationships of the r factors with q common shocks may motivate such a modeling choice (see Stock and Watson (2016) for more information).

In the so-called *exact dynamic factor model*, the error components from equation (J.2.1) are assumed to be mutually uncorrelated at all lags, i.e., $E[\varepsilon_{i,t}\varepsilon_{j,s}] = 0$ for $i \neq j$. However, following Banbura et al. (2011), the error term could be modeled as an AR(p') process:

$$\varepsilon_{i,t} = \sum_{j=1}^{p'} \alpha_{i,j}\varepsilon_{i,t-j} + e_{i,t}, \quad e_{i,t} \sim i.i.d.N(0, \sigma_i^2) \tag{J.2.3}$$

where $E[e_{i,t}e_{j,s}] = 0$ for $i \neq j$.

Following is an example, in matrix form, of equation (J.2.2) of the model for orders $r = 2$, $p = 2$ and $q = 2$.

$$\begin{bmatrix} f_{1,t} \\ f_{2,t} \\ f_{1,t-1} \\ f_{2,t-1} \end{bmatrix} = \begin{bmatrix} a^1_{1,1} & a^1_{1,2} & a^2_{1,1} & a^2_{1,2} \\ a^1_{2,1} & a^1_{2,2} & a^2_{2,1} & a^2_{2,2} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_{1,t-1} \\ f_{2,t-1} \\ f_{1,t-2} \\ f_{2,t-2} \end{bmatrix} + \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{1,t} \\ u_{2,t} \end{bmatrix} \tag{J.2.4}$$

$$F_t = \begin{bmatrix} A_1 & A_2 \\ I_2 & 0 \end{bmatrix} F_{t-1} + Bu_t \tag{J.2.5}$$

Quarterly and monthly variables

In order to predict a quarterly variable using monthly data, we construct a partially observed monthly counterpart of the quarterly variable as proposed in Mariano and Murasawa (2003). This allows, for instance, quarterly GDP to be explained by monthly variables. Continuing with this example, let Y_t^M be the level of the unobservable monthly GDP level and Y_t^Q the quarterly value of GDP for the partially observable monthly series. As is usual in the literature, we let quarterly GDP be observable in the third month of the quarter.

$$Y_t^Q = \begin{cases} Y_t^M + Y_{t-1}^M + Y_{t-2}^M, & t = 3, 6, 9, \ldots \\ \text{unobserved} & \text{otherwise} \end{cases} \tag{J.2.6}$$

The above accounting rule states that the quarterly GDP flow is equal to the sum of the monthly flows. Looking at the quarterly change, $y_t^Q = Y_t^Q - Y_{t-3}^Q$ [1], it is easy to show that it can be expressed as a function of the differences of the monthly variable, $y_t = Y_t^M - Y_{t-1}^M$, by using equation (J.2.6):

$$\begin{aligned} y_t^Q &= Y_t^Q - Y_{t-3}^Q \\ &= Y_t^Q + Y_{t-1}^Q - Y_{t-1}^Q + Y_{t-2}^Q - Y_{t-2}^Q - Y_{t-3}^Q \\ &= y_t + 2y_{t-1} + 3y_{t-2} + 2y_{t-3} + y_{t-4}, \quad t = 6, 9, \ldots \end{aligned} \tag{J.2.7}$$

Suppose that the variable of interest is a quarterly rate of change, x_t^Q, defined as:

$$x_t^Q \equiv \log(Y_t^Q) - \log(Y_{t-3}^Q) \tag{J.2.8}$$

Stating the approximation between the arithmetic and geometric means we have:

$$\frac{1}{3}[Y_t^M + Y_{t-1}^M + Y_{t-2}^M] \approx \sqrt[3]{Y_t^M Y_{t-1}^M Y_{t-2}^M} \tag{J.2.9}$$

Combining equations (J.2.8) and (J.2.9) we obtain the approximation from Mariano and Murasawa (2003) that expresses the quarterly growth rate of GDP as a function of the unobservable monthly growth rates x_t^M:

$$x_t^Q \approx \frac{1}{3}\left[x_t^M + 2x_{t-1}^M + 3x_{t-2}^M + 2x_{t-3}^M + x_{t-4}^M\right] \tag{J.2.10}$$

Suppose that the unobserved monthly growth rate x_t^M also admits the same factor representation as in equation (J.2.1) with loadings Λ_Q, then the quarterly GDP growth rate, x_t^Q, can be expressed as a function of monthly factors.

$$x_t^Q = \overline{\Lambda_Q}\begin{bmatrix} f_t' & \ldots & f_{t-4}' \end{bmatrix}' + \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \end{bmatrix}\begin{bmatrix} \varepsilon_t^M & \ldots & \varepsilon_{t-4}^M \end{bmatrix}' \tag{J.2.11}$$

where $\overline{\Lambda_Q} = [\Lambda_Q \; 2\Lambda_Q \; 3\Lambda_Q \; 2\Lambda_Q \; \Lambda_Q]$ is a restricted matrix of loadings on the factors and their lags. Note that the errors are normal in the exact dynamic factor model or have an AR(1) structure as in Banbura et al. (2011).

Determining the number of factors and shocks to the factors

We follow the papers by Bai and Ng (2002) and Bai and Ng (2007) to respectively define 1) the number r of factors in equation (J.2.1) and 2) the number of shocks q to the factors in equation (J.2.2).

Let $V(r, \hat{F}^r)$ be the sum of squared residuals when r factors are estimated using principal components. The the information criteria can then be written as follows:

$$IC_{r1}(r) = ln(V(r, \widehat{F}^r)) + r\left(\frac{N+T}{NT}\right) ln\left(\frac{NT}{N+T}\right) \tag{J.2.12}$$

$$IC_{r2}(r) = ln(V(r, \widehat{F}^r)) + r\left(\frac{N+T}{NT}\right) ln\Big(\min\{N, T\}\Big) \tag{J.2.13}$$

$$IC_{r3}(r) = ln(V(r, \widehat{F}^r)) + r\left(\frac{ln(\min\{N, T\})}{min\{N, T\}}\right) \tag{J.2.14}$$

[1] The aggregation scheme, and ensuing weights used for aggregating the monthly series, may differ according to the order of the difference taken. In the paper of Mariano and Murasawa (2003), the example is of a first difference of quarterly log GDP, which corresponds to a quarterly growth rate. In the case of an annual growth rate, $\Delta_{12} log(Y_t^Q) = log(Y_t^Q) - log(Y_{t-12}^Q)$, the aggregation weights would be different. Such cases are not considered here.

The chosen number of factors r^* will then correspond to $\arg\min_r IC_{ri}(r)$, for $i \in \{1, 2, 3\}$. Equations (J.2.12), (J.2.13), and (J.2.14) are asymptotically equivalent, but may nevertheless give significantly different results for finite samples. To this effect, observe that the penalty in equation (J.2.13) is highest when considering finite samples.

The number of shocks `q` can be lower than the number of factors `r`. Once the number of factors is determined, we use an information criterion from Bai and Ng (2007) to estimate the number of shocks q in equation (J.2.2). Let $\widehat{F}_t$ be the r factors estimated using principal components and let $\widehat{u}_t$ be the residuals from the VAR $A(l)\widehat{F}_t = \widehat{u}_t$. The idea is to check whether the eigenvalues of the variance-covariance matrix $\widehat{\sum}_u$ are different from 0. Numerically, we will therefore want to test whether a given eigenvalue is below a predefined tolerance level.To this end, define the eigenvalues $c_1 > c_2 \geqslant ... \geqslant c_r \geqslant 0$ of $\widehat{\sum}_u$ and define the kth normalization of the k+1th eigenvalue

$$\widehat{D}_k = \left(\frac{c^2_{(k+1)}}{\sum_{j=1}^{r} c_j^2} \right)^{1/2} \tag{J.2.15}$$

Then for some $0 < m < \infty$ and $0 < \delta < 1/2$ that set the tolerance level, define the vector K

$$K = \{k : \widehat{D}_k < m/\min[N^{1/2-\delta}, T^{1/2-\delta}]\} \tag{J.2.16}$$

where the estimated number of shocks to the factors will be $\hat{q} = \min\{k \in K\}$. This estimator will converge in probability towards the real number of shocks given that r is the real number of factors.

Estimation

We will describe two methodologies for estimating dynamic factors: *Two-Stage* and *Expectation-Maximization.*

1. *Two-Stage*: This approach is described in Giannone et al. (2008) and refers to the exact DFM. In the first stage, the parameters of the matrices Λ and f_t are estimated by Principal Components Analysis (PCA) using a standardized, balanced panel ($\overline{X}_t$), in which there are no missing values and outliers. Standardization is important as PCA is not scale invariant. The estimators $\widehat{\Lambda}$ and $\widehat{f}_t$ can be obtained by solving the following optimization problem:

$$\min_{f_1,...,f_T,\Lambda} \frac{1}{NT} \sum_{t=1}^{T} (\overline{X}_t - \Lambda f_t)'(\overline{X}_t - \Lambda f_t) \quad s.t. \quad N^{-1}\Lambda'\Lambda = I_r \tag{J.2.17}$$

The estimator for the variance and covariance matrix for ε_t is then given by

$$\widehat{\Psi} = diag\left(\frac{1}{T} \sum_{t=1}^{T} (\overline{X}_t - \widehat{\Lambda}\widehat{f}_t)(\overline{X}_t - \widehat{\Lambda}\widehat{f}_t)' \right) \tag{J.2.18}$$

According to Stock and Watson (2011), the solution to (J.2.17) is to set $\widehat{\Lambda}$ equal to the eigenvectors of the variance and covariance matrix of $\overline{X}_t$ associated with the r largest eigenvalues, from which it follows that the vector $\widehat{f}_t$ is the r first principal components of $\overline{X}_t$. The coefficients of the matrix A_i, $i = 1, 2, ..., p$, from equation (J.2.2), are estimated by OLS regression of f_t on $f_{t-1}, ..., f_{t-p}$. Finally, BB' is estimated as the covariance matrix of the residuals of this regression.

In the second stage, Kalman smoothing (Durbin and Koopman, 2012) is used to re-estimate the factors for the unbalanced panel x_t considering the parameters obtained in the previous step. There are some R packages that implemented the Kalman smoothing (Tusell, 2011). However, for convenience, in the **nowcasting** package, we used the routine provided by Giannone et al. (2008). Furthermore, two options are provided when estimating the factors:

   - *No aggregation*: No bridge equation, to obtain (J.2.19), is needed if both the dependent and the explanatory variables are monthly indicators. Hence, the aggregation procedure as set out in Mariano and Murasawa (2003) is not required. Similarly, if the explanatory variables have been transformed to represent quarterly quantities, the same aggregation procedure does not need to be implemented again on the factors.
   - *With aggregation*: This option is relevant when having a dependent variable `y` of lower frequency than the explanatory variables. Factors are estimated using the monthly explanatory variables x, after which the transformation from Mariano and Murasawa (2003) is applied in order to obtain factors representing quarterly quantities. Those will be used to forecast the dependent variable in the bridge equation (J.2.19).

$$y_t = \beta_0 + \beta' \widehat{f}_t + e_t \tag{J.2.19}$$

The parameters of equation (J.2.19) are estimated by OLS, and the forecast for y_{t+h} is given by

$$\widehat{y}_{t+h} = \widehat{\beta_0} + \widehat{\beta}' \widehat{f}_{t+h} \tag{J.2.20}$$

2. Expectation-Maximization: This estimation method is able to deal with arbitrary patterns of missing values as shown in Bańbura and Modugno (2014). It is therefore less restrictive than the *Two-Stage* method with regards to the frequencies of the variables and allows for a mixed frequency database. Following Banbura et al. (2011), factors can be defined for different subgroups of variables and no longer all need to be global as in the *Two-Stage* estimation method. Below, we illustrate a case where three factors are partitioned into three groups (global, real and nominal) as in Banbura et al. (2011). Rewriting equation (J.2.1) accordingly gives equation (J.2.21). As opposed to the *Two-Stage* estimation method that builds on an exact dynamic factor model, the error term is defined as an AR(1) process. A more restrictive assumption than the *Two-Stage* method is that the number of shocks to the factors `q` is set equal to the number of factors `r`.

$$x_t = \mu + \begin{pmatrix} \Lambda_{N,G} & \Lambda_{N,N} & 0 \\ \Lambda_{R,G} & 0 & \Lambda_{R,R} \end{pmatrix} \begin{pmatrix} f_t^G \\ z f_t^N \\ f_t^R \end{pmatrix} + \varepsilon_t \tag{J.2.21}$$

where

$$\begin{pmatrix} \Lambda_{N,G} & \Lambda_{N,N} & 0 \\ \Lambda_{R,G} & 0 & \Lambda_{R,R} \end{pmatrix} = \Lambda \tag{J.2.22}$$

$$\begin{pmatrix} f_t^G \\ f_t^N \\ f_t^R \end{pmatrix} = f_t \tag{J.2.23}$$

The global factor is estimated considering all the explanatory variables, while the estimates of the nominal and real factors only consider variable classified, respectively, as nominal and real. The parameter μ is a vector of constants of dimension N. As previously mentioned, the alternative proposed by Banbura et al. (2011) to the exact DFM, allows for serial autocorrelation among the error of equation (J.2.1) along an $AR(1)$ process:

$$\varepsilon_{i,t} = \alpha_i \epsilon_{i,t-1} + e_{i,t}, \quad e_{i,t} \sim i.i.d.N(0, \sigma_i^2) \tag{J.2.24}$$

where $E[e_{i,t} e_{j,s}] = 0$ for $i \neq j$.

In this model, the parameters, the unobserved common factors and the missing values are estimated through the *Expectation-Maximization* algorithm, which uses the following recursive structure:

- E-step: The conditional expectation of the likelihood function is calculated using the estimates of the static parameters (θ) from the previous iteration, θ_j;
- M-step: The new parameters, θ_{j+1} are estimated by maximizing the likelihood function from the previous step with respect to θ.

Convergence is achieved when the absolute change in the value of the log-likelihood function is less than 10^{-4}, the tolerance level used for this algorithm. The recursive process starts with the PCA estimates given in Giannone et al. (2008) (first stage of the *Two-Stage* method).

The R package

Working on the dataset

The first step in the nowcasting process is to prepare the data in a way that is compatible with the proposed models and estimation methods. One of the motivations of the presented models is the forecasting improvements that can be achieved by using higher frequency variables. More specifically, the gains that can be obtained in using monthly variables to forecast quarterly series. Hence, all functions require monthly `mts` objects. In practice, the quarterly variables are usually represented as monthly variables for which the last month of the quarter is observed. As illustrated in the working examples, such straightforward transformations from one frequency representation to another can be achieved by using the functions `qtr2month()` or `month2qtr()`.

With regards to the estimation methods, different inputs may have to be provided. As a matter of fact, the *Two-Stage* method is more restrictive on the format of the variables as it depends on principal components in the first stage. This requires a strategy to deal with missing values, which are not part of the jagged edge, beforehand. Giannone et al. (2008) propose to replace such missing values with the median of the series that are then smoothed with a moving average. Since such a strategy assigns a value that is independent of the information contained in other contemporaneous variables, it is advisable to exclude series with many missing values. The *EM algorithm*, however, is able to deal with missing values in a way that uses the information contained in other variables and might therefore not require discarding such variables. Finally, independently of the estimation method, stationary series are required. The usual transformations for making time series stationary and the different strategies to deal with missing values have been included in the function `Bpanel()` that prepares the database for the nowcasting function. Since these choices require careful attention, the function `Bpanel()` is explained in further detail.

```
Bpanel(base, trans, NA.replace = TRUE, aggregate = FALSE, k.ma = 3, na.prop = 1/3, h = 12)
```

`trans` is a vector indicating the transformations to be applied to the variables. For most cases, the available transformations are sufficient to make economic variables stationary. The transformation must be specified by using one of the following values for the argument `trans`:

`trans = 0`: the observed series is preserved;

`trans = 1`: monthly rate of change: $\frac{x_{i,t} - x_{i,t-1}}{x_{i,t-1}}$;

`trans = 2`: monthly difference: $x_{i,t} - x_{i,t-1}$;

`trans = 3`: monthly difference in year-over-year rate of change:

$$\frac{x_{i,t} - x_{i,t-12}}{x_{i,t-12}} - \frac{x_{i,t-1} - x_{i,t-13}}{x_{i,t-13}};$$

`trans = 4`: monthly difference in year-over-year difference:

$$(x_{i,t} - x_{i,t-12}) - (x_{i,t-1} - x_{i,t-13}).$$

`trans = 5`: year difference:

$$(x_{i,t} - x_{i,t-12})$$

`trans = 6`: year-over-year rate of change:

$$\frac{x_{i,t} - x_{i,t-12}}{x_{i,t-12}}$$

`trans = 7`: quarterly rate of change

$$\frac{x_{i,t} - x_{i,t-3}}{x_{i,t-3}}$$

`NA.replace` is a boolean to determine whether missing values should be replaced (`NA.replace = TRUE`) or not (`NA.replace = FALSE`).

`aggregate` is a boolean to indicate whether to aggregate the monthly variables to represent quarterly quantities. If `TRUE` the aggregation is made following the approximation of Mariano and Murasawa (2003).

`k.ma` is a numeric representing the degree of the moving average correction if `NA.replace = TRUE`.

`na.prop` is a number between 0 and 1 indicating the ratio of missing observations to the total number of observations beyond which series will be discarded. The default is 1/3, meaning that if more than 1/3 of the observations are missing the series will be discarded from the database.

`h` indicates how many periods should be added to the database. Default is 12. Those missing values will be predicted with the function `nowcast()`.

Determining the number of factors and shocks to the factors

As explained in the section on parameter estimation, the package offers different functions to estimate the number of factors r and of idiosyncratic shocks q of equations (J.2.1) and (J.2.2) respectively.

1. Function `ICfactors()` estimates the number of factors r^* according to an information criterion. The argument `x` is a balanced panel and `rmax` is an integer representing the maximum number of factors for which the information criterion should be calculated. The default value is 20. `type` indicates which of the information criterion from Bai and Ng (2002) to use. `type` $\in \{1, 2, 3\}$ with the default being 2 as explained in the methodological section. If `x` is not a balanced panel, the function will delete rows with missing values in order to use principal components.

   ```
   ICfactors(x, rmax = 20, type = 2)
   ```

2. Function `ICshocks()` estimates the number of idiosyncratic shocks given a number r of factors according to the information criterion introduced in the previous section. The argument `x` is a balanced panel. `delta` and `m` are parameters of the information criterion, where $0 < m < \infty$ and $0 < \delta < 1/2$. The default values are those from Bai and Ng (2007): $m = 1$ and $\delta = 0.1$. If the number of factors `r` is not specified it will be defined according to `ICfactors(x,rmax = 20,type = 2)`. `p` is the number of lags in the VAR of equation (J.2.2). If not specified, the default is the lowest most occurring value from the information criteria used within the function `VARselect()` from the package **vars**.

   ```
   ICshocks(x, r = NULL, p = NULL, delta = 0.1, m = 1)
   ```

Forecasts

An important feature of factor models is the dimensionality reduction of (many) original variables into a few common factors. Hence, the target variable `y` will be expressed as a function of a few factors extracted from the explanatory variables. This motivated the choice of the inputs for the `nowcast()` function. The formula format, which is well known to R users, captures this idea as `formula = y~.` can be understood as the projection of `y` on the information contained in the dataset. The model's parameters are estimated according to the selected method (`2s`, `2s_agg` and `EM`, which correspond, respectively, to "two-stage", "two-stage with factor aggregation" and "Expectation-Maximization algorithm") described in the section on estimation. The number `r` of dynamic factors, the number `q` of shocks to the factors, and the lag order `p` of the factors are determined beforehand as shown in the previous subsection. The argument `blocks` can be used with the `EM` method to estimate factors for different subgroups of variables. Finally, the argument `frequency` is necessary for all methods in order to identify the frequency of the variables.

```
nowcast(formula, data, q = NULL, r = NULL, p = NULL, method = 'EM', blocks = NULL,
        frequency = NULL)
```

In the first two methods (`2s` and `2s_agg`), the factors are calculated based on the monthly variables, on which the dependent variable `y` will be regressed. The difference between `2s` and `2s_agg` is that for the latter the monthly factors are transformed into quarterly quantities while in the former no such aggregation is used. A linear regression (*bridge equation* if `y` is quarterly) of `y` on the factors allows the former to be forecast.

In the third method (`EM`) no bridge equation is needed, as opposed to the *Two-Stage* method. In practice, the algorithm will estimate all the missing values respecting the restrictions imposed by equation (J.2.11). The forecasts of quarterly time series are defined as the estimated values of the third month of the out of sample quarters. As opposed to the *Two-Stage* method, the number of common shocks `q` can not be specified and is assumed to be equal to `r`, the number of factors in each block.

Analyzing the results

The function `nowcast.plot()` allows to plot several outputs from the function `nowcast()`.

```
nowcast.plot(out, type = "fcst")
```

The argument `out` is the output from the function `nowcast()`. The argument `type` can be chosen from the list {"fcst","factors","eigenvalues","eigenvectors"}:

- `"fcst"`: shows the `y` variable and its forecasts in sample and out of sample.
- `"factors"`: shows all the estimated factors.
- `"eigenvalues"`: indicates how much of the variability in the dataset is explained by each factor.
- `"eigenvectors"`: shows the importance of each variable in the first factor.

A working example of the *Two-Stage* method: nowcasting Brazilian GDP

Constructing the dataset

In this example we showcase how to nowcast Brazilian GDP using the *Two-Stage* estimation method. Most of the variables of interest can be downloaded from the Brazilian central bank using the function `BETSget()` from the package **BETS**. The variables and the associated codes can be found on the Brazilian central bank's website [2]. For the sake of simplicity we have included the database, and all relevant information within the package[3].

```
> library(nowcasting)
> data(BRGDP)
```

For this example we will construct a pseudo real-time dataset, using the function `PRTDB()`. Some variables, such as GDP, suffer revisions over time. Since we do not take revisions into account, we refer to such datasets as pseudo real-time (as opposed to vintages). The (approximate) delays in days are included in the `BRGDP` object and will be used to define if observations were available at a specific moment in time. The dataset is then treated for outliers and missing values that are not part of the jagged edges of the data, i.e., that are not due to the different publication lags of the variables. This is achieved through the function `Bpanel()`. Unless otherwise specified by the user, the function will also discard series with over 1/3 missing values.

```
> vintage <- PRTDB(mts = BRGDP$base, delay = BRGDP$delay, vintage = "2015-06-01")
> base <- window(vintage, start = c(2005,06), frequency = 12)
> x <- Bpanel(base = base, trans = BRGDP$trans)
```

The function `month2qtr()` transforms monthly time series into quarterly ones. In this case we want to use the value of the third month as the quarterly value.

```
> GDP <- base[,which(colnames(base) == "PIB")]
> window(GDP, start = c(2015,1))
        Jan    Feb    Mar    Apr    May    Jun
2015     NA     NA 170.68     NA     NA     NA

> GDP_qtr <- month2qtr(x = GDP, reference_month = 3)
> window(GDP_qtr, start = c(2015,1))
       Qtr1   Qtr2
2015 170.68     NA
```

The quarterly GDP indicator, in this example, is an index representing the seasonal quarterly product. $\Delta_4 Y_t$ deals with seasonality, while $\Delta\Delta_4 Y_t$ is necessary to obtain a stationary time series. To test the latter, one could look at tests for unit roots or serial auto correlation that are included in many R packages.

```
> y <- diff(diff(GDP_qtr,4))
> y <- qtr2month(y)
```

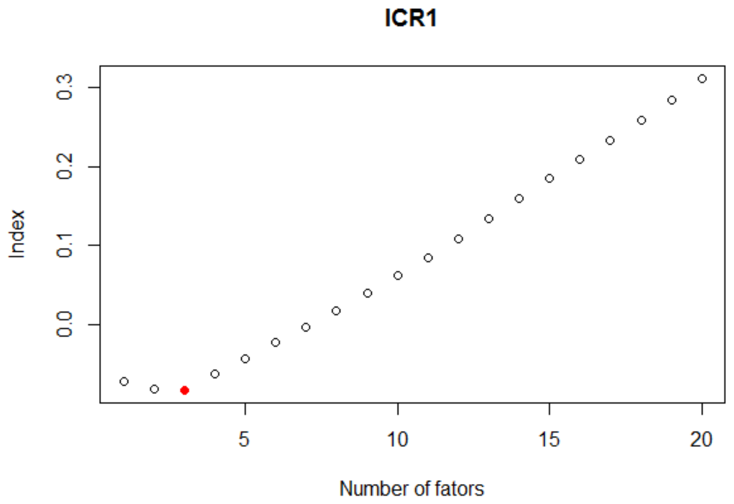
[2] see http://www4.bcb.gov.br/pec/series/port/aviso.asp

[3] The database is a random sample of 100 variables from our own database

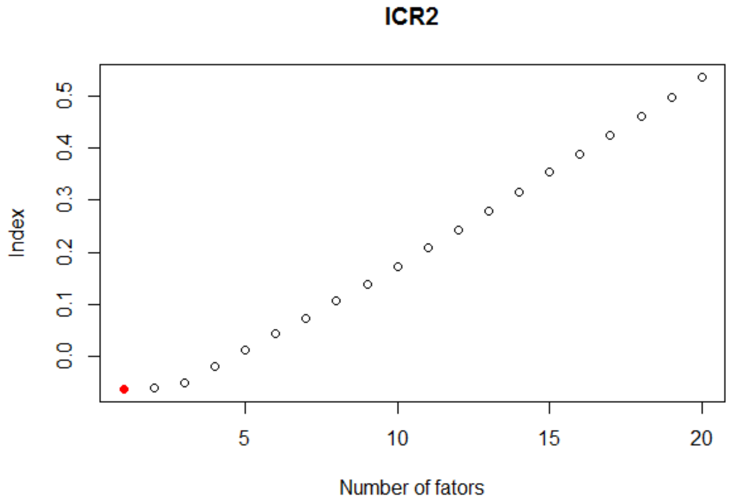
Determining the number of factors and shocks

The dataset `x`, which now only posses jagged edges, is well suited for the information criteria that make use of principal components. The estimated number of factors is given by the function `ICfactors()`. As explained in the previous section, the information criteria might give different results for finite samples.

```
> ICR1 <- ICfactors(x = x, type = 1)
```



```
> ICR2 <- ICfactors(x = x, type = 2)
```



Finally, given the chosen number of factors for our model, we can use an information criterion for determining the number of shocks to the factors.

```
> ICQ1 <- ICshocks(x = x, r = 2, p = 2)
> ICQ1$q_star
[1] 2
```

Forecasts

Let the object `data` be a monthly `mts` object where the first column is a partially observable stationary GDP series (`y`) and the remaining columns a balanced panel of stationary time series (`x`).

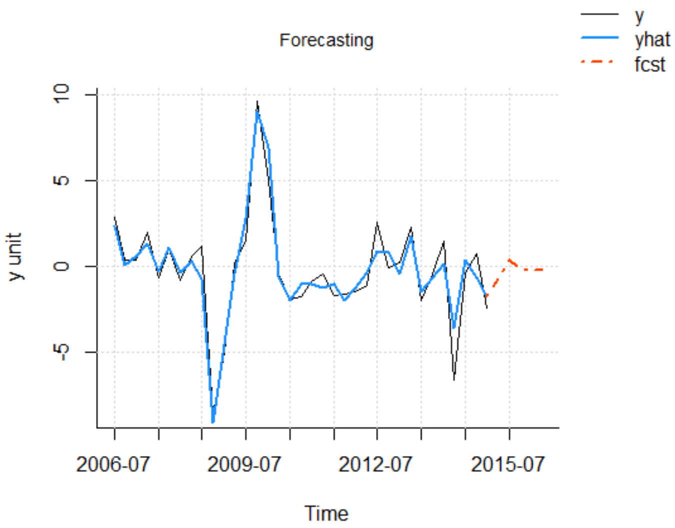
The `frequency` vector will be determined by the quarterly GDP series and the remaining monthly series. In this example the factors will be aggregated to obtain quarterly quantities by setting `method = "2s_agg"`.

```
> data <- cbind(y,x)
> frequency <- c(4,rep(12,ncol(x)))
> now <- nowcast(formula = y~., data = data, r = 2, q = 2 , p = 2, method = "2s_agg",
    frequency = frequency)
> summary(now$reg)

Call:
stats::lm(formula = Y ~ ., data = Balanced_panel)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.0248 -0.5679  0.1094  0.5835  1.8912 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.19526    0.16940  -1.153    0.258    
Factor1      0.22610    0.01456  15.528  < 2e-16 ***
Factor2      0.06135    0.01174   5.228 1.02e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.002 on 32 degrees of freedom
Multiple R-squared:  0.8995,Adjusted R-squared:  0.8932 
F-statistic: 143.1 on 2 and 32 DF,  p-value: < 2.2e-16
```

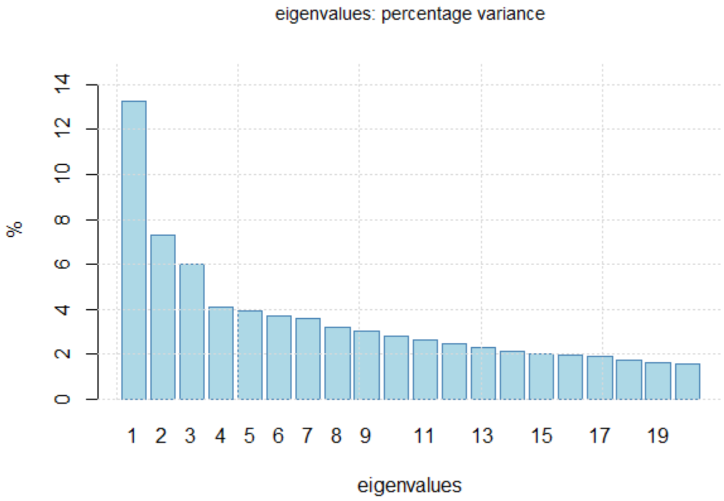
Results

The function `nowcast.plot()` enables the user to visualize some of the results. Say, for instance, that we want to look at fitted values and out-of-sample forecasts. This can be achieved by setting the type to `"fcst"`. We might also want to look at the eigenvalues of the normalized variance-covariance matrix of our balanced panel or at how variables enter the first factor.

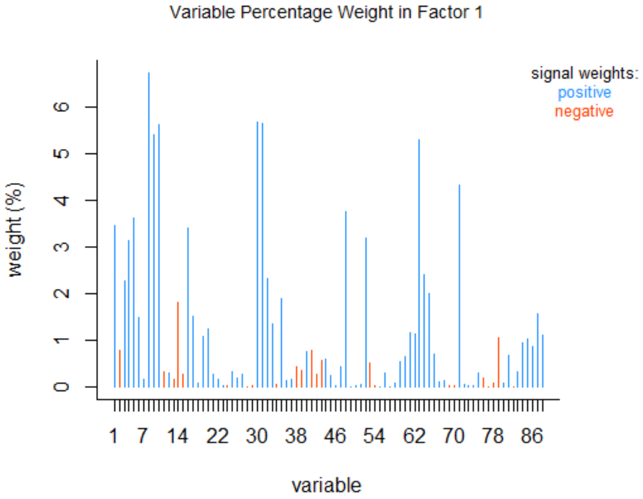
```
> nowcast.plot(now, type = "fcst")
```



```
> nowcast.plot(now, type = "eigenvalues")
```



```
> nowcast.plot(now, type = "eigenvectors")
```



Up until now, we have been forecasting GDP after transforming it into a stationary variable. We might want to transform the former back into a level variable in order to forecast the actual growth rate. Remember that we transformed GDP according to

$$\begin{aligned} diff(diff(GDP_t,4)) &= (GDP_t - GDP_{t-4}) - (GDP_{t-1} - GDP_{t-5}) \\ &= GDP_t + GDP_{t-5} - GDP_{t-1} - GDP_{t-4} \end{aligned} \tag{J.4.1}$$

that can be rewritten as

$$GDP_t = diff(diff(GDP_t,4)) - GDP_{t-5} + GDP_{t-1} + GDP_{t-4} \tag{J.4.2}$$

Equation (J.4.2) gives us the forecast of the new quarter GDP level. The variable `BRGDP$GDP` is the non-stationary GDP.

```
> level_forecast <- na.omit(now$yfcst[,3])[1] - tail(na.omit(GDP_qtr),5)[1] +
+   +   tail(na.omit(GDP_qtr),5)[5] + tail(na.omit(GDP_qtr),5)[2]
> level_forecast
[1] 170.4783

> position_q2_2015 <- which(time(BRGDP$GDP) == 2015.25)
> BRGDP$GDP[position_q2_2015]
[1] 169.24
```

A working example of the *EM* method: The NY FED nowcast

Constructing the dataset

In this example we work with the data the Federal Reserve of New York made available to reproduce its weekly nowcasting report[4]. The explanatory variables are mixed frequencies including both monthly and quarterly series.

```
> library(nowcasting)
> data(NYFED)
> NYFED$legend$SeriesName
 [1] "Payroll Employment"          "Job Openings"
 [3] "Consumer Price Index"        "Durable Goods Orders"
 [5] "Retail Sales"                "Unemployment Rate"
 [7] "Housing Starts"              "Industrial Production"
 [9] "Personal Income"             "Exports"
[11] "Imports"                     "Construction Spending"
[13] "Import Price Index"          "Core Consumer Price Index"
[15] "Core PCE Price Index"        "PCE Price Index"
[17] "Building Permits"            "Capacity Utilization Rate"
[19] "Business Inventories"        "Unit Labor Cost"
[21] "Export Price Index"          "Empire State Mfg Index"
[23] "Philadelphia Fed Mfg Index"  "Real Consumption Spending"
[25] "Real Gross Domestic Product"
```

Similarly to the previous working example, the object NYFED contains all the necessary information to run the nowcast() function. The time series, the block structure, the transformations to make the variables stationary and the variables' frequencies can be loaded as illustrated below.

```
> base <- NYFED$base
> blocks <- NYFED$blocks$blocks
> trans <- NYFED$legend$Transformation
> frequency <- NYFED$legend$Frequency
> delay <- NYFED$legend$delay
```

The dataset data can be prepared by using the function Bpanel(). Using the EM algorithm, there is no need to replace missing values that are not part of the jagged edges, as was the case with the *Two-Stage* method. This can be achieved by setting NA.replace to FALSE. In this case we do not want to discard series based on a particular ratio of missing values to total observations as was the case in the *Two-Stage* method. This is done by setting na.prop = 1, where 1 indicates that only series with more than 100% missing values will be discarded.

```
> data <- Bpanel(base = base, trans = trans, NA.replace = FALSE, na.prop = 1)
```

Forecasts

The model´s specifications are the same as those used by the NY FED. We therefore limit the number of factors, r, per block to one and define the factor process as a VAR(1), i.e., p = 1. The convergence of the log-likelihood function is displayed every 5 iterations.

```
> nowEM <- nowcast(formula = GDPC1~., data = data, r = 1, p = 1, method = "EM",
    blocks = blocks, frequency = frequency)
5th iteration:
The loglikelihood went from -2418.5983 to -2406.1482
...
65th iteration:
The loglikelihood went from -2354.084 to -2353.8435
```

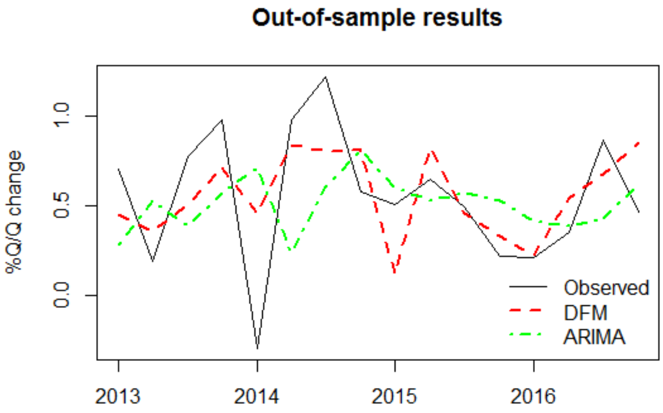
[4] https://www.newyorkfed.org/research/policy/nowcast

Results

Combining the functions `nowcast()` and `PRTB()` within a loop, we illustrate how a pseudo out-of-sample end-of-quarter nowcast can be made. The vector `fcst_dates` defines the last month of the quarters for which quarterly GDP growth will be nowcast. The vector `delay` contains approximate delays, in days, with which variables are published. This enables us to construct a pseudo real-time dataset for a given day.

```
> fcst_dates <- seq.Date(from = as.Date("2013-03-01"),to = as.Date("2017-12-01"),
        by = "quarter")
> fcst_results <- NULL
> for(date in fcst_dates){
+
+   vintage <- PRTDB(data, delay = delay, vintage = date)
+   nowEM <- nowcast(formula = GDPC1~., data = vintage, r = 1, p = 1, method = "EM",
        blocks = blocks, frequency = frequency)
+   fcst_results <- c(fcst_results,tail(nowEM$yfcst[,3],1))
+
+ }
```

The results of this out-of-sample nowcast example, as well as the results of an out-of-sample ARIMA, are displayed below.



The root mean square prediction error can easily be calculated for the 2013-2016 period. For this given example, when compared to one-period-ahead projections given by an ARIMA model, a Theil's U statistic of 0.70 is obtained, signaling a 30% improvement over the benchmark.

Summary

The package **nowcasting** was developed in order to facilitate the use of dynamic factor models for large datasets as set out in Giannone et al. (2008) and Banbura et al. (2011). The package offers functions at each step of the forecasting process to help the user treat data, choose and estimate the value of parameters, as well as interpret results. We provided a working example for nowcasting Brazilian GDP, illustrating each step and showing how to implement the various functions available. We also used the New York FED nowcasting exercise to illustrate the EM algorithm. We will, in the future, work on adding new tools for the user to better leverage the `EM` method by identifying the source of forecast revisions. As shown by the New York FED nowcasting report, this is an interesting policy instrument that helps contextualizing forecast updates.

Acknowledgements

We thank Daniel Mesquita for revising some of the codes and our colleagues from FGV-IBRE for helpful inputs. We also thank an anonymous referee and the R journal editor Olivia Lau for constructive comments. The authors are responsible for any errors in this paper. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Bibliography

K. A. Aastveit and T. Trovik. Nowcasting norwegian gdp: The role of asset prices in a small open economy. *Empirical Economics*, 42(1):95–119, 2012. ISSN 1435-8921. URL https://doi.org/10.1007/s00181-010-0429-9. [p230]

E. Angelini, G. Camba-Mendez, D. Giannone, L. Reichlin, and G. Rünstler. Short-Term Forecasts of Euro Area GDP Growth. Working Papers ECARES ECARES 2008-035, ULB – Universite Libre de Bruxelles, 2008. URL https://ideas.repec.org/p/eca/wpaper/2008_035.html. [p230]

J. Bai and S. Ng. Determining the number of factors in approximate factor models. *Econometrica*, 70(1):191–221, 2002. URL https://doi.org/10.1111/1468-0262.00273. [p232, 236]

J. Bai and S. Ng. Determining the number of primitive shocks in factor models. *Journal of Business & Economic Statistics*, 25(1):52–60, 2007. URL https://doi.org/10.1198/073500106000000413. [p232, 233, 236]

M. Banbura, D. Giannone, and L. Reichlin. Nowcasting. *Oxford Handbook on Economic Forecasting*, 2011. [p230, 231, 232, 234, 242]

M. Bańbura and M. Modugno. Maximum likelihood estimation of factor models on datasets with arbitrary pattern of missing data. *Journal of Applied Econometrics*, 29(1):133–160, 2014. URL https://doi.org/10.1002/jae.2306. [p234]

M. Bańbura and G. Rünstler. A look into the factor model black box: Publication lags and the role of hard and soft data in forecasting gdp. *International Journal of Forecasting*, 27(2):333–346, 2011. URL https://doi.org/10.1016/j.ijforecast.2010.01.011. [p230]

J. Boivin and S. Ng. Are more data always better for factor analysis? *Journal of Econometrics*, 132 (1):169–194, 2006. URL https://doi.org/10.1016/j.jeconom.2005.01.027. [p230]

M. Chauvet. A monthly indicator of brazilian gdp. *Brazilian Review of Econometrics*, 21(1):1–47, 2001. URL https://doi.org/10.12660/bre.v21n12001.3191. [p230]

A. D'Agostino, G. Domenico, and P. Surico. (Un)Predictability and Macroeconomic Stability. Research Technical Papers 5/RT/06, Central Bank of Ireland, 2006. URL https://ideas.repec.org/p/cbi/wpaper/5-rt-06.html. [p230]

A. D'Agostino, K. McQuinn, and D. O'Brien. Now-casting irish gdp. Research Technical Papers 9/RT/08, Central Bank of Ireland, 2008. URL https://doi.org/10.1787/19952899. [p230]

T. Dahlhaus, J.-D. Guenette, and G. Vasishtha. Nowcasting bric+m in real time. Staff working papers, Bank of Canada, 2015. URL https://doi.org/10.1016/j.ijforecast.2017.05.002. [p230]

J. Durbin and S. J. Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2 edition, 2012. URL https://EconPapers.repec.org/RePEc:oxp:obooks:9780199641178. [p233]

M. Forni, M. Hallin, M. Lippi, and L. Reichlin. The generalized dynamic factor model consistency and rates. *Journal of Econometrics*, 119(2):231–255, 2004. URL https://doi.org/10.1016/s0304-4076(03)00196-9. [p230]

D. Giannone, L. Reichlin, and D. Small. Nowcasting: The real-time informational content of macroeconomic data. *Journal of Monetary Economics*, 55(4):665–676, 2008. URL https://doi.org/10.1016/j.jmoneco.2008.05.010. [p230, 233, 234, 235, 242]

M. Marcellino, J. Stock, and M. Watson. Macroeconomic forecasting in the euro area: Country specific versus area-wide information. *European Economic Review*, 47(1):1–18, 2003. URL https://doi.org/10.1016/s0014-2921(02)00206-4. [p230]

R. S. Mariano and Y. Murasawa. A new coincident index of business cycles based on monthly and quarterly series. *Journal of applied Econometrics*, 18(4):427–443, 2003. URL https://doi.org/10.1002/jae.695. [p231, 232, 233, 235]

T. D. Matheson. An Analysis of the Informational Content of New Zealand Data Releases: The Importance of Business Opinion Surveys. *Economic Modelling*, 27(1):304–314, 2010. URL https://doi.org/10.1016/j.econmod.2009.09.010. [p230]

J. H. Stock and M. Watson. Dynamic factor models. *Oxford Handbook on Economic Forecasting*, 2011. [p233]

J. H. Stock and M. W. Watson. Forecasting with many predictors. *Handbook of economic forecasting*, 1:515–554, 2006. URL https://doi.org/10.1016/s1574-0706(05)01010-4. [p230]

J. H. Stock and M. W. Watson. *Dynamic Factor Models, Factor-Augmented Vector Autoregressions, and Structural Vector Autoregressions in Macroeconomics*, volume 2. Elsevier, 2016. URL https://doi.org/10.1016/bs.hesmac.2016.04.002. [p230, 231]

F. Tusell. Kalman filtering in r. *Journal of Statistical Software, Articles*, 39(2):1–27, 2011. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v039.i02. [p233]

C. Van Nieuwenhuyze, K. Ruth, A. Rua, P. Jelonek, A. Jakaitiene, A. Den Reijer, R. Cristadoro, G. Rünstler, S. Benk, and K. Barhoumi. Short-term forecasting of gdp using large monthly datasets: a pseudo real-time forecast evaluation exercise. Occasional Paper Series 84, European Central Bank, 2008. URL https://doi.org/10.1002/for.1105. [p230]

Serge de Valk
EPGE Brazilian School of Economics and Finance (FGV EPGE)
60 Barão de Itambi, Botafogo, Rio de Janeiro - RJ
Brazil
serge.valk@fgv.br

Daiane Marcolino de Mattos
FGV-IBRE
60 Barão de Itambi, Botafogo, Rio de Janeiro - RJ
Brazil
daiane.mattos@fgv.br

Pedro Guilherme Costa Ferreira
FGV-IBRE
60 Barão de Itambi, Botafogo, Rio de Janeiro - RJ
Brazil
pedro.guilherme@fgv.br

Connecting R with D3 for dynamic graphics, to explore multivariate data with tours

by Michael Kipp, Ursula Laa, Dianne Cook

Abstract The **tourr** package in R has several algorithms and displays for showing multivariate data as a sequence of low-dimensional projections. It can display as a movie but has no capacity for interaction, such as stop/go, change tour type, drop/add variables. The **tourrGui** package provides these sorts of controls, but the interface is programmed with the dated **RGtk2** package. This work explores using custom messages to pass data from R to D3 for viewing, using the Shiny framework. This is an approach that can be generally used for creating all sorts of interactive graphics.

Introduction

Did you know you can run *any javascript you like* in a Shiny application and you can pass *whatever you want including JSON* back and forth? This massively widens the scope of what you can do with Shiny, and generating a tour of multivariate data with this approach is a really good example of what is possible.

The tour algorithm (Asimov, 1985) is a way of systematically generating and displaying projections of high-dimensional spaces in order for the viewer to examine the multivariate distribution of data. It can do this either randomly, or by picking projections judged interesting according to some criterion or index function. The **tourr** package (Wickham et al., 2011) provides the computing and display in R to make several types of tours: grand, guided, little and local. The projection dimension can be chosen between one and the number of variables in the data. The display, though, has no capacity for interaction. The viewer can watch the tour like a movie, but not pause it and restart, or change tour type, or number of variables.

These interactive controls were provided with the **tourrGui** package (Huang et al., 2012), with was programmed with the **RGtk2** package (Lawrence and Temple Lang, 2010). This is not the toolkit of choice today, and has been superceded with primarily web-capable tools, like Shiny (Chang et al., 2017). To display dynamic graphics though, is not straight-forward. This paper explains how to use D3 (Bostock et al., 2011) as the display engine in a Shiny graphical user interface (GUI), using custom message passing between server and client.

Creating a tour, with the tourr package

The **tourr** package (Wickham et al., 2011) is an R implementation of the tour algorithms discussed in Cook et al. (2007). It includes methods for geodesic interpolation and basis generation, as well as an implementation of the simulated annealing algorithm to optimise projection pursuit indices for the guided tour. The tour can be displayed directly in the R graphics device, for example, the code below generates a 1D density tour. Figure 1 shows snapshots.

```
library(tourr)
# quartz() # to display on a Mac; X11() # For windows; The Rstudio graphics
# device is not advised
animate_dist(flea[, 1:6], center = TRUE)
```

A tour path is a smooth sequence of projection matrices, $p \times d$, that when combined with a matrix of n data points, $n \times p$, and a rendering method, produces a steady stream of d-dimensional views of the data. Each tour is initialised with the `new_tour()` method, which instantiates a tour object and takes as arguments the data X, the tour method, e.g. `guided_tour()`, and the starting basis. Once initialised, a new target plane is chosen, and a series of steps along a geodesic path from starting to target plane are generated by interpolation.

This requires a series of calls to the tour object producing the series of projections. The steps are discrete, of size given by ω/Δ, where ω denotes the angular velocity of the geodesic interpolation, and Δ is a parameter denoting frames per second, reflecting the rendering speed of the device in use. The Δ parameter can be thought of as the frames per second, while ω affects the speed at which the tour moves through the projection space. For our purposes, Δ, `fps` in the code, is set at 25, while the ω can be adjusted by the user.

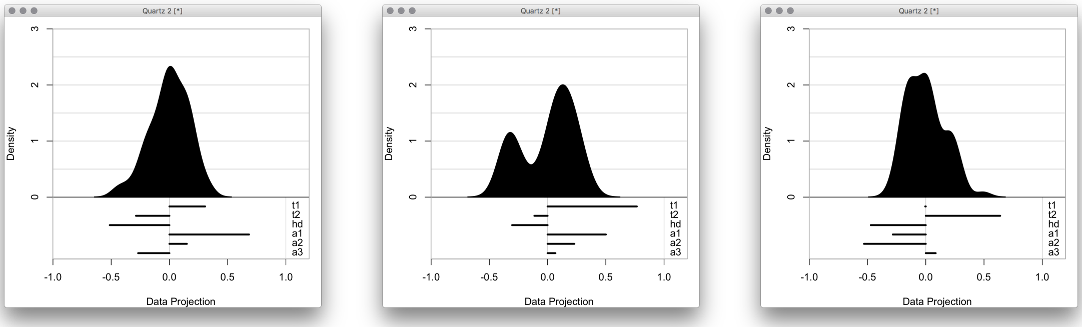


Figure 1: Three projections from a 1D tour of 6D data, displayed as a density. Full video can be seen at https://vimeo.com/255466661.

Connecting the tour projections to D3 display using `sendCustomMessage`

D3.js (Data-Driven Documents) (Bostock et al., 2011) is a JavaScript library for manipulating documents based on data. The advantages of D3 are similar to those provided by Shiny: namely, an industry standard with rich array of powerful, easy to use methods and widgets that can be displayed on a wide variety of devices, with a large user base. D3 works on data objects in the JavaScript Object Notation (JSON) format, which are then parsed and used to display customisable data visualisations.

The new implementation of the tour interface uses D3 to render each projection step returned by R, focusing on 2D projections as a test case. It does this by drawing and re-drawing a scatterplot with dots (or circles in D3 language) and providing SVG objects for the web browser to render. Figure 2 shows the new GUI.

The Shiny functions `session$sendCustomMessage()` and `Shiny.addCustomMessageHandler()` are provided to transport data between R and JavaScript. Whenever the former is executed in R, the latter function will execute a code block in JS. There are many examples of such functions being used to pass arbitrary data from an R app to a JS front-end, few examples exist of this basic functionality to update a D3 animation in real-time.

To set up the interface for the app, we need to load the relevant scripts into the Shiny app and assign a section for the resulting plots. This is done when setting up the user interface. We import D3 and our plotting code via the `tags$script` (for web links) and `includeScript` (for reading from a full path). We use `tags$div` to assign an id for the output section that can be accessed in the D3 code.

```
tags$script(src = "https://d3js.org/d3.v4.min.js"),
includeScript(system.file("js/d3anim.js", package = "tourrGUID3")),
tags$div(id = "d3_output")
```

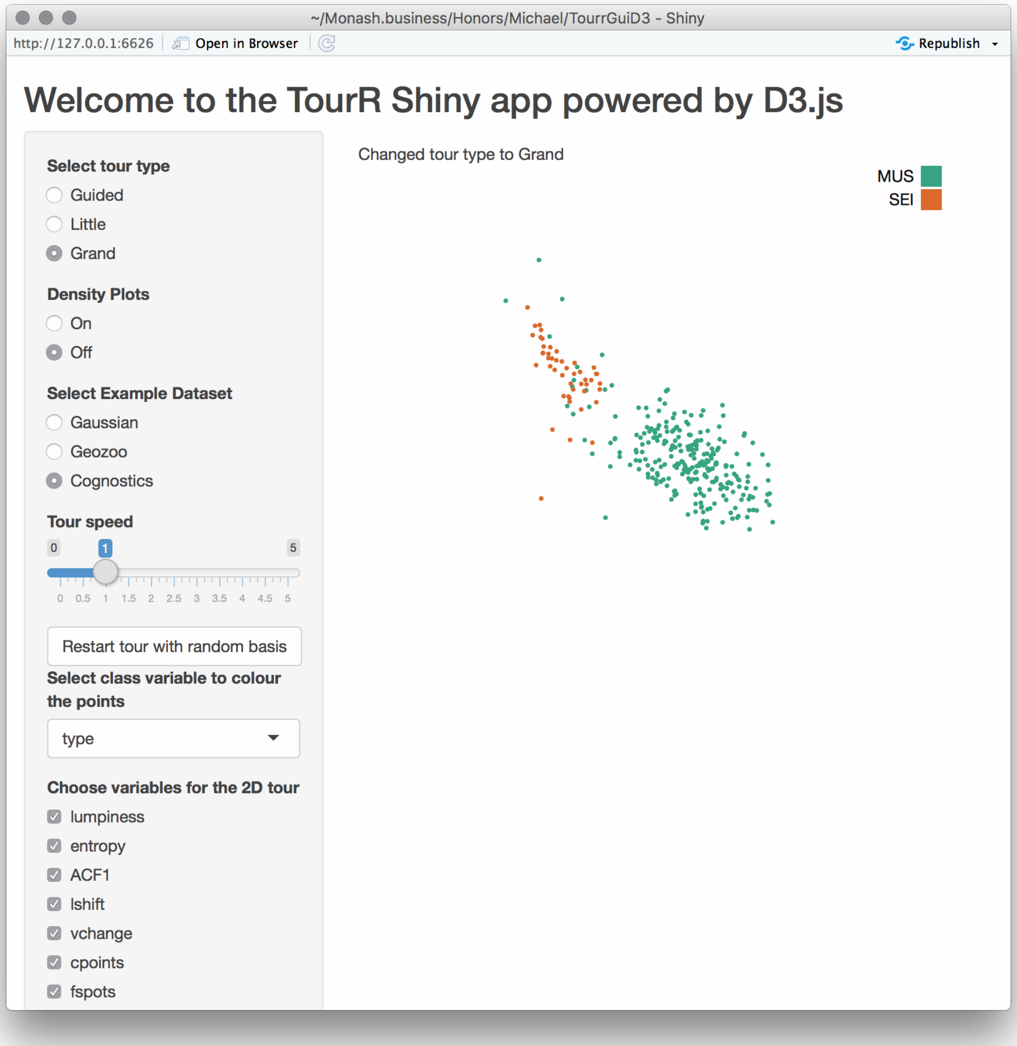
On the D3 side we can access the id defined in Shiny, and for example assign it to a scalable vector graphics (svg) object to be filled in D3 and rendered onto the Shiny app.

```
var svg = d3.select("#d3_output")
    .append("svg")
    .attr("width", w)
    .attr("height", h);
```

The data format expected by D3 is in JSON format, which combines two basic programming paradigms: a collection of name/value pairs, and an ordered list of values. R's preferred data formats include data frames, vectors and matrices. Every time a new projection has been calculated with the tour path, the resulting matrix needs to be converted to JSON and sent to D3. Using a named list we can send multiple JSON datasets to D3, e.g. to draw both the data points (stored in dataframe d) and the projection axes (stored in dataframe a). Converting dataframes will pass the column names to JSON. The code to send the D3 data looks like this:

```
session$sendCustomMessage(type = "data", message = list(d = toJSON(d), a = toJSON(a)))
```

This code is from the observe environment from the `server.R` file. It converts the matrix of projected data points to JSON format, and sends it to JavaScript with the id data. The list entries of



Welcome to the TourR Shiny app powered by D3.js

Select tour type
○ Guided
○ Little
● Grand

Density Plots
○ On
● Off

Select Example Dataset
○ Gaussian
○ Geozoo
● Cognostics

Tour speed: 1

Select class variable to colour the points: type

Choose variables for the 2D tour
☑ lumpiness
☑ entropy
☑ ACF1
☑ lshift
☑ vchange
☑ cpoints
☑ fspots

Changed tour type to Grand

Figure 2: Shiny GUI for the tour, with D3 as the display engine. GUI provides controls to select tour type, change speed, restart, and select variables to include.

the "message" can parsed in D3 by its `data()` method, e.g. `data(message.d)` to access the projected data points, and we can access each column through the column names assigned in the original dataframe, and loop over all rows for rendering. All of the code required to render the scatterplots and legends, along with colours, is JavaScript code in the file `d3anim.js`. In particular, the data from R is handled with the following code:

```
Shiny.addCustomMessageHandler("data",
    function(message) {
        /* D3 scatterplot is drawn and re-drawn using the
            data sent from the server. */
}
```

Every time the message is sent (25 times per second), the code-block is run.

Getting projections

The `observeEvent` Shiny method defines a code block to be run whenever some input value changes. The following code snippet restarts a tour using a random basis:

```
observeEvent(input$restart_random,
  {
    p <- length(input$variables)
    b <- matrix(runif(2*p), p, 2)
    rv$tour <-
      new_tour(as.matrix(rv$d[input$variables]),
              choose_tour(input$type,
              input$guidedIndex,
              c(rv$class[[1]])), b)
})
```

The projections are calculated using the tour object in an `observe()` environment, which re-executes the code whenever it is invalidated. The invalidation is either by a change in reactive value inside the code block, or we can schedule a re-execution by explicitly invalidating the observer after a selected interval using `invalidateLater()`. The projections are calculated using the following code block:

```
observe({
    if (length(rv$mat[1, ]) < 3) {
      session$sendCustomMessage(type = "debug",
      message = "Error: Need >2 variables.")
    }
    aps <- rv$aps
    tour <- rv$tour
    step <- rv$tour(aps / fps)
    invalidateLater(1000 / fps)
    j <- center(rv$mat %*% step$proj)
    j <- cbind(j, class = rv$class)
    colnames(j) <- NULL
    session$sendCustomMessage(type = "data",
         message = list(d = toJSON(data.frame(pL=rv$pLabel[,1], x=j[,2],
                                              y=j[,1], c=j[,3])),
                        a = toJSON(data.frame(n=rv$vars, y=step$proj[,1],
                                              x=step$proj[,2]))))
})
```

Try it

You can try the app yourself using this code:

```
devtools::install_github("uschiLaa/tourrGUID3")
library(tourrGUID3)
launchApp(system.file("extdata", "geozoo.csv", package = "tourrGUID3"))
```

Troubleshooting

Fixing bugs in the JavaScript code can be cumbersome, as R and Shiny will not report any errors. Tracing JavaScript errors can be done when using the JavaScript console in the web browser. For example, in Google Chrome the console can be accessed via the "Developer Tools" option found under "Moore Tools" in the control menu. Typical errors that we encountered were version dependent syntax in D3, e.g. for axis definitions or scaling.

Pros and cons

The D3 canvas makes for smooth drawing and re-drawing of the data projections. Adding a GUI around the display is straightforward with the Shiny package, e.g. control elements such as stop/go, increase/decrease speed, change tour type, add/remove variables from the mix.

The main disadvantage is that the speed is inconsistent, as server and client play tag to keep up with each other, and the display cannot handle many observations. Noticeable slow down was oberved with 2000 points, the main reason being the rendering time required for the large number of SVG circle elements. The situation can be improved when using a single HTML5 canvas element to draw the scatter points, significantly reducing the rendering time.

Another disadvantage is that the displays needs to be coded anew. D3 provides mostly primitives, and example code, to make scatterplots, and contours, but the data displays all need to be coded again.

Summary

The custom message tools from Shiny provide a way to share a tour path with the D3 renderer, and embed it in a Shiny GUI providing controls such as stop/go, increase/decrease speed, change tour type, add/remove variables. However, the approach doesn't provide the smooth motion that is needed for easy display of projections, and is slow for large numbers of observations.

Code

The code is available at https://github.com/uschiLaa/tourrGUID3, and the source material for this paper is available at https://github.com/dicook/paper-tourrd3.

Acknowledgements

Thanks to Yihui Xie for pointing out the custom message tools.

Bibliography

D. Asimov. The Grand Tour: A Tool for Viewing Multidimensional Data. *SIAM Journal of Scientific and Statistical Computing*, 6(1):128–143, 1985. URL https://doi.org/10.1137/0906011. [p245]

M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. URL https://doi.org/10.1109/TVCG.2011.185. [p245, 246]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *Shiny: Web Application Framework for R*, 2017. URL https://CRAN.R-project.org/package=shiny. R package version 1.0.5. [p245]

D. Cook, A. Buja, E. K. Lee, and H. Wickham. Grand Tours, Projection Pursuit Guided Tours and Manual Controls. Springer-Verlag, Berlin, Heidelberg, 2007. URL https://doi.org/10.1007/978-3-540-33037-0_13. [p245]

B. Huang, D. Cook, and H. Wickham. tourrGui: A gWidgets GUI for the Tour to Explore High-Dimensional Data Using Low-Dimensional Projections. *Journal of Statistical Software*, 49(6):1–12, 2012. URL https://doi.org/10.18637/jss.v049.i06. [p245]

M. Lawrence and D. Temple Lang. RGtk2: A Graphical User Interface Toolkit for R. *Journal of Statistical Software*, 37(8):1–52, 2010. URL https://doi.org/10.18637/jss.v037.i08. [p245]

H. Wickham, D. Cook, H. Hofmann, and A. Buja. tourr: An R Package for Exploring Multivariate Data with Projections. *Journal of Statistical Software*, 40(2):1–18, 2011. URL https://doi.org/10.18637/jss.v040.i02. [p245]

Michael Kipp
Monash University
Department of Econometrics and Business Statistics
mkipp271@gmail.com

Ursula Laa
Monash University
School of Physics and Astronomy
ursula.laa@monash.edu

Dianne Cook
Monash University
Department of Econometrics and Business Statistics
dicook@monash.edu

SimCorrMix: Simulation of Correlated Data with Multiple Variable Types Including Continuous and Count Mixture Distributions

by Allison Fialkowski and Hemant Tiwari

Abstract The **SimCorrMix** package generates correlated continuous (normal, non-normal, and mixture), binary, ordinal, and count (regular and zero-inflated, Poisson and Negative Binomial) variables that mimic real-world data sets. Continuous variables are simulated using either Fleishman's third-order or Headrick's fifth-order power method transformation. Simulation occurs at the component level for continuous mixture distributions, and the target correlation matrix is specified in terms of correlations with components. However, the package contains functions to approximate expected correlations with continuous mixture variables. There are two simulation pathways which calculate intermediate correlations involving count variables differently, increasing accuracy under a wide range of parameters. The package also provides functions to calculate cumulants of continuous mixture distributions, check parameter inputs, calculate feasible correlation boundaries, and summarize and plot simulated variables. **SimCorrMix** is an important addition to existing R simulation packages because it is the first to include continuous mixture and zero-inflated count variables in correlated data sets.

Introduction

Finite mixture distributions have a wide range of applications in clinical and genetic studies. They provide a useful way to describe heterogeneity in a population, e.g., when the population consists of several subpopulations or when an outcome is a composite response from multiple sources. In survival analysis, survival times in competing risk models have been described by mixtures of exponential, Weibull, or Gompertz densities (Larson and Dinse, 1985; Lau et al., 2009, 2011). In medical research, finite mixture models may be used to detect clusters of subjects (*cluster analysis*) that share certain characteristics, e.g., concomitant diseases, intellectual ability, or history of physical or emotional abuse (McLachlan, 1992; Newcomer et al., 2011; Pamulaparty et al., 2016). In schizophrenia research, Gaussian mixture distributions have frequently described the earlier age of onset in men than in women and the vast phenotypic heterogeneity in the disorder spectrum (Everitt, 1996; Lewine, 1981; Sham et al., 1994; Welham et al., 2000).

Count mixture distributions, particularly zero-inflated Poisson and Negative Binomial, are required to model count data with an excess number of zeros and/or overdispersion. These distributions play an important role in a wide array of studies, modeling health insurance claim count data (Ismail and Zamani, 2013), the number of manufacturing defects (Lambert, 1992), the efficacy of pesticides (Hall, 2000), and prognostic factors of Hepatitis C (Baghban et al., 2013). Human microbiome studies, which seek to develop new diagnostic tests and therapeutic agents, use RNA-sequencing (RNA-seq) data to assess differential composition of bacterial communities. The operational taxonomic unit (OTU) count data may exhibit overdispersion and an excess number of zeros, necessitating zero-inflated Negative Binomial models (Zhang et al., 2016). Differential gene expression analysis utilizes RNA-seq data to search for genes that exhibit differences in expression level across conditions (e.g., drug treatments) (Soneson and Delorenzi, 2013; Solomon, 2014). Zero-inflated count models have also been used to characterize the molecular basis of phenotypic variation in diseases, including next-generation sequencing of breast cancer data (Zhou et al., 2017).

The main challenge in applying mixture distributions is estimating the parameters for the component densities. This is usually done with the EM algorithm, and the best model is chosen by the lowest Akaike or Bayesian information criterion (AIC or BIC). Current packages that provide Gaussian mixture models include: **AdaptGauss**, which uses Pareto density estimation (Thrun et al., 2017); **DPP**, which uses a Dirichlet process prior (Avila et al., 2017); **bgmm**, which employs two partially supervised mixture modeling methods (Biecek and Szczurek, 2017); and **ClusterR**, **mclust**, and **mixture**, which conduct cluster analysis (Mouselimis, 2017; Fraley et al., 2017; Browne et al., 2015). Although Gaussian distributions are the most common, the mixture may contain any combination of component distributions. Packages that provide alternatives include: **AdMit**, which fits an adaptive mixture of Student-t distributions (Ardia, 2017); **bimixt**, which uses case-control data (Winerip et al., 2015); **bmixture**, which conducts Bayesian estimation for finite mixtures of

Gamma, Normal and t-distributions (Mohammadi, 2017); **CAMAN**, which provides tools for the analysis of finite semiparametric mixtures in univariate and bivariate data (Schlattmann et al., 2016); **flexmix**, which implements mixtures of standard linear models, generalized linear models and model-based clustering (Gruen and Leisch, 2017); **mixdist**, which applies to grouped or conditional data (MacDonald and with contributions from Juan Du, 2012); **mixtools** and **nspmix**, which analyze a variety of parametric and semiparametric models (Young et al., 2017; Wang, 2017); **MixtureInf**, which conducts model inference (Li et al., 2016); and **Rmixmod**, which provides an interface to the MIXMOD software and permits Gaussian or multinomial mixtures (Langrognet et al., 2016). With regards to count mixtures, the **BhGLM**, **hurdlr**, and **zic** packages model zero-inflated distributions with Bayesian methods (Yi, 2017; Balderama and Trippe, 2017; Jochmann, 2017).

Given component parameters, there are existing R packages which simulate mixture distributions. The **mixpack** package generates univariate random Gaussian mixtures (Comas-Cufí et al., 2017). The **distr** package produces univariate mixtures with components specified by name from **stats** distributions (Kohl, 2017; R Core Team, 2017). The **rebmix** package simulates univariate or multivariate random datasets for mixtures of conditionally independent Normal, Lognormal, Weibull, Gamma, Binomial, Poisson, Dirac, Uniform, or von Mises component densities. It also simulates multivariate random datasets for Gaussian mixtures with unrestricted variance-covariance matrices (Nagode, 2017).

Existing simulation packages are limited by: 1) the variety of available component distributions and 2) the inability to produce correlated data sets with multiple variable types. Clinical and genetic studies which involve variables with mixture distributions frequently incorporate influential covariates, such as gender, race, drug treatment, and age. These covariates are correlated with the mixture variables and maintaining this correlation structure is necessary when simulating data based on real data sets (*plasmodes*, as in Vaughan et al., 2009). The simulated data sets can then be used to accurately perform hypothesis testing and power calculations with the desired type-I or type-II error.

SimCorrMix is an important addition to existing R simulation packages because it is the first to include continuous mixture and zero-inflated count variables in correlated data sets. Therefore, the package can be used to simulate data sets that mimic real-world clinical or genetic data. **SimCorrMix** generates continuous (normal, non-normal, or mixture distributions), binary, ordinal, and count (regular or zero-inflated, Poisson or Negative Binomial) variables with a specified correlation matrix via the functions `corrvar` and `corrvar2`. The user may also generate one continuous mixture variable with the `contmixvar1` function. The methods extend those found in the **SimMultiCorrData** package (version $\geqslant$ 0.2.1, Fialkowski, 2017; Fialkowski and Tiwari, 2017). Standard normal variables with an imposed intermediate correlation matrix are transformed to generate the desired distributions. Continuous variables are simulated using either Fleishman (1978)'s third-order or Headrick (2002)'s fifth-order polynomial transformation method (the power method transformation, PMT). The fifth-order PMT accurately reproduces non-normal data up to the sixth moment, produces more random variables with valid PDF's, and generates data with a wider range of standardized kurtoses. Simulation occurs at the component-level for continuous mixture distributions. These components are transformed into the desired mixture variables using random multinomial variables based on the mixing probabilities. The target correlation matrix is specified in terms of correlations with components of continuous mixture variables. However, **SimCorrMix** provides functions to approximate expected correlations with continuous mixture variables given target correlations with the components. Binary and ordinal variables are simulated using a modification of **GenOrd**'s `ordsample` function (Barbiero and Ferrari, 2015b). Count variables are simulated using the inverse cumulative density function (CDF) method with distribution functions imported from **VGAM** (Yee, 2017).

Two simulation pathways (*correlation method 1* and *correlation method 2*) within **SimCorrMix** provide two different techniques for calculating intermediate correlations involving count variables. Each pathway is associated with functions to calculate feasible correlation boundaries and/or validate a target correlation matrix `rho`, calculate intermediate correlations (during simulation), and generate correlated variables. Correlation method 1 uses `validcorr`, `intercorr`, and `corrvar`. Correlation method 2 uses `validcorr2`, `intercorr2`, and `corrvar2`. The order of the variables in `rho` must be 1^{st} ordinal ($r \geqslant 2$ categories), 2^{nd} continuous non-mixture, 3^{rd} components of continuous mixture, 4^{th} regular Poisson, 5^{th} zero-inflated Poisson, 6^{th} regular Negative Binomial (NB), and 7^{th} zero-inflated NB. This ordering is integral for the simulation process. Each simulation pathway shows greater accuracy under different parameter ranges and Calculation of intermediate correlations for count variables details the differences in the methods. The optional error loop can improve the accuracy of the final correlation matrix in most situations.

The simulation functions do not contain parameter checks or variable summaries in order to decrease simulation time. All parameters should be checked first with `validpar` in order to prevent errors. The function `summary_var` generates summaries by variable type and calculates the final correlation matrix and maximum correlation error. The package also provides the functions `calc_mixmoments` to calculate the standardized cumulants of continuous mixture distributions, `plot_simpdf_theory` to plot simulated PDF's, and `plot_simtheory` to plot simulated data values. The plotting functions work for continuous or count variables and overlay target distributions, which are specified by name (39 distributions currently available) or PDF function `fx`. The `fx` input is useful when plotting continuous mixture variables since there are no distribution functions available in R. There are five vignettes in the package documentation to help the user understand the simulation and analysis methods. The stable version of the package is available via the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=SimCorrMix, and the development version may be found on GitHub at https://github.com/AFialkowski/SimCorrMix. The results given in this paper are reproducible (for R version $\geqslant$ 3.4.1, **SimCorrMix** version $\geqslant$ 0.1.0).

Overview of mixture distributions

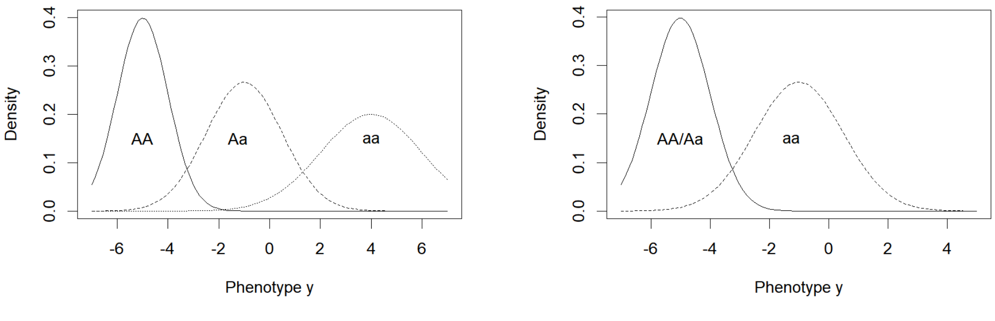
Mixture distributions describe continuous or discrete random variables that are drawn from more than one component distribution. For a random variable Y from a finite mixture distribution with k components, the probability density function (PDF) or probability mass function (PMF) is:

$$h_Y\left(y\right) = \sum_{i=1}^{k} \pi_i f_{Y_i}\left(y\right), \qquad \sum_{i=1}^{k} \pi_i = 1 \tag{N.2.1}$$

The π_i are mixing parameters which determine the weight of each component distribution $f_{Y_i}\left(y\right)$ in the overall probability distribution. As long as each component has a valid probability distribution, the overall distribution $h_Y\left(y\right)$ has a valid probability distribution. The main assumption is statistical independence between the process of randomly selecting the component distribution and the distributions themselves. Assume there is a random selection process that first generates the numbers 1, ..., k with probabilities π_1, ..., π_k. After selecting number i, where $1 \leqslant i \leqslant k$, a random variable y_i is drawn from component distribution $f_{Y_i}\left(y\right)$ (Davenport et al., 1988; Everitt, 1996).

Continuous mixture distributions

Continuous mixture distributions are used in genetic research to model the effect of underlying genetic factors (e.g., genotypes, alleles, or mutations at chromosomal loci) on continuous traits. Consider a single locus with two alleles A and a, producing three genotypes AA, Aa, and aa with population frequencies p_{AA}, p_{Aa}, and p_{aa}. Figure 1a shows a *codominant mixture* in which each genotype exhibits a different phenotype; Figure 1b shows a *dominant mixture* in which individuals with at least one A allele possess the same phenotype (Schork et al., 1996).



(a) Codominant mixture.

(b) Dominant mixture.

Figure 1: Examples of commingled distributions in genetics.

For a continuous phenotype y, the normal mixture density function describing a commingled distribution is given by:

$$\begin{aligned} &f\left(y|p_{AA},\ \mu_{AA},\ \sigma^2_{AA};\ p_{Aa},\ \mu_{Aa},\ \sigma^2_{Aa};\ p_{aa},\ \mu_{aa},\ \sigma^2_{aa}\right) = \\ &\quad p_{AA}\phi\left(y|\mu_{AA},\ \sigma^2_{AA}\right) + p_{Aa}\phi\left(y|\mu_{Aa},\ \sigma^2_{Aa}\right) + p_{aa}\phi\left(y|\mu_{aa},\ \sigma^2_{aa}\right), \end{aligned} \tag{N.3.1}$$

where $\phi\left(y|\mu,\ \sigma^2\right)$ is the normal density function with mean μ and variance σ^2. *Commingling analysis* may also study traits that are polygenic (result from the additive effects of several genes) or multifactorial (polygenic traits with environmental factors, see Elston et al., 2002). For example, mixture models explain the heterogeneity observed in gene-mapping studies of complex human diseases, including cancer, chronic fatigue syndrome, bipolar disorder, coronary artery disease, and diabetes (Fridley et al., 2010; Bahcall, 2015; Bhattacharjee et al., 2015; **?**). *Segregation analysis* extends commingling analysis to individuals within a pedigree. Mixed models evaluate whether a genetic locus is affecting a particular quantitative trait and incorporate additional influential factors. Finally, *linkage analysis* discovers the location of genetic loci using recombination rates, and the regression likelihood equation may be written as a mixture distribution (Schork et al., 1996).

Generation of continuous distributions in SimCorrMix

Continuous variables, including components of mixture variables, are created using either Fleishman (1978)'s third-order (`method = "Fleishman"`) or Headrick (2002)'s fifth-order (`method = "Polynomial"`) PMT applied to standard normal variables. The transformation is expressed as follows:

$$Y = p\left(Z\right) = c_0 + c_1 Z + c_2 Z^2 + c_3 Z^3 + c_4 Z^4 + c_5 Z^5,\ Z \sim N\left(0, 1\right), \qquad \text{(N.3.2)}$$

where $c_4 = c_5 = 0$ for Fleishman's method. The real constants are calculated by **SimMultiCorrData**'s `find_constants`, which solves the system of non-linear equations given in `poly` or `fleish`. The simulation functions `corrvar` and `corrvar2` contain checks to see if any distributions are repeated for non-mixture or components of mixture variables. If so, these are noted so the constants are only calculated once, decreasing simulation time. Mixture variables are generated from their components based on random multinomial variables described by their mixing probabilities (using **stat**'s `rmultinom`).

The fifth-order PMT allows additional control over the fifth and sixth moments of the generated distribution. In addition, the range of feasible standardized kurtosis (γ_2) values, given skew (γ_1) and standardized fifth (γ_3) and sixth (γ_4) cumulants, is larger than with the third-order PMT. For example, Fleishman's method can not be used to generate a non-normal distribution with a ratio of $\gamma_1^2/\gamma_2 > 9/14$. This eliminates the χ^2 family of distributions, which has a constant ratio of $\gamma_1^2/\gamma_2 = 2/3$ (Headrick and Kowalchuk, 2007). The fifth-order method also generates more distributions with valid PDFs. However, if the fifth and sixth cumulants do not exist, the Fleishman approximation should be used. This would be the case for t-distributions with degrees of freedom below 7.

For some sets of cumulants, it is either not possible to find power method constants (indicated by a `stop` error) or the calculated constants do not generate valid PDF's (indicated in the simulation function results). For the fifth-order PMT, adding a value to the sixth cumulant may provide solutions. This can be done for non-mixture variables in `Six` or components of mixture variables in `mix_Six`, and `find_constants` will use the smallest correction that yields a valid PDF. Another possible reason for function failure is that the standardized kurtosis for a distribution is below the lower boundary of values which can be generated using the third or fifth-order PMT. This boundary can be found with **SimMultiCorrData**'s `calc_lower_skurt` using skew (for `method = "Fleishman"`) and standardized fifth and sixth cumulants (for `method = "Polynomial"`).

Expected cumulants of continuous mixture variables

The PMT simulates continuous variables by matching standardized cumulants derived from central moments. Using standardized cumulants decreases the complexity involved in calculations when a distribution has large central moments. In view of this, let Y be a real-valued random variable with cumulative distribution function F. Define the central moments, μ_r, of Y as:

$$\mu_r = \mu_r\left(Y\right) = \mathbb{E}[y-\mu]^r = \int_{-\infty}^{+\infty} [y-\mu]^r dF\left(y\right). \qquad \text{(N.3.3)}$$

The standardized cumulants are found by dividing the first six cumulants κ_1 - κ_6 by $\sqrt{\kappa_2}^r = \left(\sigma^2\right)^{r/2} = \sigma^r$, where σ^2 is the variance of Y and r is the order of the cumulant (Kendall and Stuart, 1977):

$$0 = \frac{\kappa_1}{\sqrt{\kappa_2}^1} = \frac{\mu_1}{\sigma^1} \quad \text{(N.3.4)} \qquad \gamma_2 = \frac{\kappa_4}{\sqrt{\kappa_2}^4} = \frac{\mu_4}{\sigma^4} - 3 \quad \text{(N.3.7)}$$

$$1 = \frac{\kappa_2}{\sqrt{\kappa_2}^2} = \frac{\mu_2}{\sigma^2} \quad \text{(N.3.5)} \qquad \gamma_3 = \frac{\kappa_5}{\sqrt{\kappa_2}^5} = \frac{\mu_5}{\sigma^5} - 10\gamma_1 \quad \text{(N.3.8)}$$

$$\gamma_1 = \frac{\kappa_3}{\sqrt{\kappa_2}^3} = \frac{\mu_3}{\sigma^3} \quad \text{(N.3.6)} \qquad \gamma_4 = \frac{\kappa_6}{\sqrt{\kappa_2}^6} = \frac{\mu_6}{\sigma^6} - 15\gamma_2 - 10{\gamma_1}^2 - 15. \quad \text{(N.3.9)}$$

The values γ_1, γ_2, γ_3, and γ_4 correspond to skew, standardized kurtosis (so that the normal distribution has a value of 0, subsequently referred to as *skurtosis*), and standardized fifth and sixth cumulants. The corresponding sample values for the above can be obtained by replacing μ_r by $m_r = \sum_{j=1}^{n} \left(x_j - m_1\right)^r / n$ (Headrick, 2002).

The standardized cumulants for a continuous mixture variable can be derived in terms of the standardized cumulants of its component distributions. Suppose the goal is to simulate a continuous mixture variable Y with PDF $h_Y(y)$ that contains two component distributions Y_a and Y_b with mixing parameters π_a and π_b:

$$h_Y(y) = \pi_a f_{Y_a}(y) + \pi_b g_{Y_b}(y), \ y \in \boldsymbol{Y}, \ \pi_a \in (0,\ 1), \ \pi_b \in (0,\ 1), \ \pi_a + \pi_b = 1. \quad \text{(N.3.10)}$$

Here,

$$Y_a = \sigma_a Z'_a + \mu_a, \ Y_a \sim f_{Y_a}(y), \ y \in \boldsymbol{Y_a} \ \text{ and } \ Y_b = \sigma_b Z'_b + \mu_b, \ Y_b \sim g_{Y_b}(y), \ y \in \boldsymbol{Y_b} \quad \text{(N.3.11)}$$

so that Y_a and Y_b have expected values μ_a and μ_b and variances σ_a^2 and σ_b^2. Assume the variables Z'_a and Z'_b are generated with zero mean and unit variance using Headrick's fifth-order PMT given the specified values for skew $\left(\gamma'_{1_a},\ \gamma'_{1_b}\right)$, skurtosis $\left(\gamma'_{2_a},\ \gamma'_{2_b}\right)$, and standardized fifth $\left(\gamma'_{3_a},\ \gamma'_{3_b}\right)$ and sixth $\left(\gamma'_{4_a},\ \gamma'_{4_b}\right)$ cumulants:

$$\begin{aligned} Z'_a &= c_{0_a} + c_{1_a} Z_a + c_{2_a} Z_a^2 + c_{3_a} Z_a^3 + c_{4_a} Z_a^4 + c_{5_a} Z_a^5, \ Z_a \sim N(0,\ 1) \\ Z'_b &= c_{0_b} + c_{1_b} Z_b + c_{2_b} Z_b^2 + c_{3_b} Z_b^3 + c_{4_b} Z_b^4 + c_{5_b} Z_b^5, \ Z_b \sim N(0,\ 1). \end{aligned} \quad \text{(N.3.12)}$$

The constants c_{0_a}, ..., c_{5_a} and c_{0_b}, ..., c_{5_b} are the solutions to the system of equations given in **SimMultiCorrData**'s `poly` function and calculated by `find_constants`. Similar results hold for Fleishman's third-order PMT, where the constants c_{0_a}, ..., c_{3_a} and c_{0_b}, ..., c_{3_b} are the solutions to the system of equations given in `fleish` ($c_{4_a} = c_{5_a} = c_{4_b} = c_{5_b} = 0$).

The r^{th} expected value of Y can be expressed as:

$$\begin{aligned} \mathbb{E}\left[Y^r\right] &= \int y^r h_Y(y)\, dy = \pi_a \int y^r f_{Y_a}(y)\, dy + \pi_b \int y^r g_{Y_b}(y)\, dy \\ &= \pi_a\, \mathbb{E}\left[Y_a^r\right] + \pi_b\, \mathbb{E}\left[Y_b^r\right]. \end{aligned} \quad \text{(N.3.13)}$$

Equation N.3.13 can be used to derive expressions for the mean, variance, skew, skurtosis, and standardized fifth and sixth cumulants of Y in terms of the r^{th} expected values of Y_a and Y_b. See Derivation of expected cumulants of continuous mixture variables in the Appendix for the expressions and proofs.

Extension to more than two component distributions

If the desired mixture distribution Y contains more than two component distributions, the expected values of Y are again expressed as sums of the expected values of the component distributions, with weights equal to the associated mixing parameters. For example, assume Y contains k component distributions Y_1, ..., Y_k with mixing parameters given by π_1, ..., π_k, where $\sum_{i=1}^{k} \pi_i = 1$. The component distributions are described by the following parameters: means μ_1, ..., μ_k, variances σ_1^2, ..., σ_k^2, skews γ'_{1_1}, ..., γ'_{1_k}, skurtoses γ'_{2_1}, ..., γ'_{2_k}, fifth cumulants γ'_{3_1}, ..., γ'_{3_k}, and sixth cumulants γ'_{4_1}, ..., γ'_{4_k}. Then the r^{th} expected value of Y can be expressed as:

$$\mathbb{E}\left[Y^r\right] = \int y^r h_Y(y)\, dy = \sum_{i=1}^{k} \pi_i \int y^r f_{Y_i}(y)\, dy = \sum_{i=1}^{k} \pi_i E_{f_i}\left[Y_i^r\right]. \quad \text{(N.3.14)}$$

Therefore, a method similar to that above can be used to derive the system of equations defining the mean, variance, skew, skurtosis, and standardized fifth and sixth cumulants of Y. These equations are used within the function `calc_mixmoments` to determine the values for a mixture variable. The `summary_var` function executes `calc_mixmoments` to provide target distributions for simulated continuous mixture variables.

Example with Normal and Beta mixture variables

Let Y_1 be a mixture of Normal(-5, 2), Normal(1, 3), and Normal(7, 4) distributions with mixing parameters $0.36, 0.48$, and 0.16. This variable could represent a continuous trait with a codominant mixture distribution, as in Figure 1a, where $p_A = 0.6$ and $p_a = 0.4$. Let Y_2 be a mixture of Beta(13, 11) and Beta(13, 4) distributions with mixing parameters 0.3 and 0.7. Beta-mixture models are widely used in bioinformatics to represent correlation coefficients. These could arise from pathway analysis of a relevant gene to study if gene-expression levels are correlated with those of other genes. The correlations could also describe the expression levels of the same gene measured in different studies, as in meta-analyses of multiple gene-expression experiments. Since expression varies greatly across genes, the correlations may come from different probability distributions within one mixture distribution. Each component distribution represents groups of genes with similar behavior. Ji et al. (2005) proposed a Beta-mixture model for correlation coefficients. Laurila et al. (2011) extended this model to methylation microarray data in order to reduce dimensionality and detect fluctuations in methylation status between various samples and tissues. Other extensions include cluster analysis (Dai et al., 2009), single nucleotide polymorphism (SNP) analysis (Fu et al., 2011), pattern recognition and image processing (Bouguila et al., 2006; Ma and Leijon, 2011), and quantile normalization to correct probe design bias (Teschendorff et al., 2013). Since these methods assume independence among components, Dai and Charnigo (2015) developed a compound hierarchical correlated Beta-mixture model to permit correlations among components, applying it to cluster mouse transcription factor DNA binding data.

The standardized cumulants for the Normal and Beta mixtures using the fifth-order PMT are found as follows:

```
library("SimCorrMix")
B1 <- calc_theory("Beta", c(13, 11))
B2 <- calc_theory("Beta", c(13, 4))
mix_pis <- list(c(0.36, 0.48, 0.16), c(0.3, 0.7))
mix_mus <- list(c(-5, 1, 7), c(B1[1], B2[1]))
mix_sigmas <- list(c(sqrt(2), sqrt(3), sqrt(4)), c(B1[2], B2[2]))
mix_skews <- list(c(0, 0, 0), c(B1[3], B2[3]))
mix_skurts <- list(c(0, 0, 0), c(B1[4], B2[4]))
mix_fifths <- list(c(0, 0, 0), c(B1[5], B2[5]))
mix_sixths <- list(c(0, 0, 0), c(B1[6], B2[6]))
Nstcum <- calc_mixmoments(mix_pis[[1]], mix_mus[[1]], mix_sigmas[[1]],
  mix_skews[[1]], mix_skurts[[1]], mix_fifths[[1]], mix_sixths[[1]])
Nstcum
##       mean         sd       skew   kurtosis      fifth      sixth
## -0.2000000  4.4810713  0.3264729 -0.6238472 -1.0244454  1.4939902
Bstcum <- calc_mixmoments(mix_pis[[2]], mix_mus[[2]], mix_sigmas[[2]],
  mix_skews[[2]], mix_skurts[[2]], mix_fifths[[2]], mix_sixths[[2]])
Bstcum
##      mean         sd       skew   kurtosis      fifth      sixth
## 0.6977941  0.1429099 -0.4563146 -0.5409080  1.7219898  0.5584577
```

SimMultiCorrData's `calc_theory` was used first to obtain the standardized cumulants for each of the Beta distributions.

Calculation of intermediate correlations for continuous variables

The target correlation matrix `rho` in the simulation functions `corrvar` and `corrvar2` is specified in terms of the correlations with components of continuous mixture variables. This allows the user to set the correlation between components of the same mixture variable to any desired value. If this correlation is small (i.e., 0–0.2), the intermediate correlation matrix `Sigma` may need to be converted to the nearest positive-definite (PD) matrix. This is done within the simulation functions by specifying `use.nearPD = TRUE`, and Higham (2002)'s algorithm is executed with the **Matrix** package's `nearPD` function (Bates and Maechler, 2017). Otherwise, negative eigenvalues are replaced with 0.

The function `intercorr_cont` calculates the intermediate correlations for the standard normal variables used in Equation N.3.2. This is necessary because the transformation decreases the absolute value of the final correlations. The function uses Equation 7b derived by Headrick and Sawilowsky (1999, p. 28) for the third-order PMT and Equation 26 derived by Headrick (2002, p. 694) for the fifth-order PMT.

Approximate correlations for continuous mixture variables:

Even though the correlations for the continuous mixture variables are set at the component level, we can approximate the resulting correlations for the mixture variables. Assume Y_1 and Y_2 are two continuous mixture variables. Let Y_1 have k_1 components with mixing probabilities $\alpha_1, ..., \alpha_{k_1}$ and standard deviations $\sigma_{1_1}, ..., \sigma_{1_{k_1}}$. Let Y_2 have k_2 components with mixing probabilities $\beta_1, ..., \beta_{k_2}$ and standard deviations $\sigma_{2_1}, ..., \sigma_{2_{k_2}}$.

Correlation between continuous mixture variables Y_1 and Y_2

The correlation between the mixture variables Y_1 and Y_2 is given by:

$$\rho_{Y_1Y_2} = \frac{\mathbb{E}\left[Y_1Y_2\right] - \mathbb{E}\left[Y_1\right]\mathbb{E}\left[Y_2\right]}{\sigma_1\sigma_2}, \tag{N.3.15}$$

where σ_1^2 is the variance of Y_1 and σ_2^2 is the variance of Y_2. Equation N.3.15 requires the expected value of the product of Y_1 and Y_2. Since Y_1 and Y_2 may contain any number of components and these components may have any continuous distribution, there is no general way to determine this expected value. Therefore, it is approximated by expressing Y_1 and Y_2 as sums of their component variables:

$$\rho_{Y_1Y_2} = \frac{\mathbb{E}\left[\left(\sum_{i=1}^{k_1}\alpha_i Y_{1_i}\right)\left(\sum_{j=1}^{k_2}\beta_j Y_{2_j}\right)\right] - \mathbb{E}\left[\sum_{i=1}^{k_1}\alpha_i Y_{1_i}\right]\mathbb{E}\left[\sum_{j=1}^{k_2}\beta_j Y_{2_j}\right]}{\sigma_1\sigma_2}, \tag{N.3.16}$$

where

$$\begin{aligned}\mathbb{E}\left[\left(\sum_{i=1}^{k_1}\alpha_i Y_{1_i}\right)\left(\sum_{j=1}^{k_2}\beta_j Y_{2_j}\right)\right] &= \mathbb{E}\left[\alpha_1 Y_{1_1}\beta_1 Y_{2_1} + \alpha_1 Y_{1_1}\beta_2 Y_{2_2} + ... + \alpha_{k_1}Y_{1_{k_1}}\beta_{k_2}Y_{2_{k_2}}\right] \\ &= \alpha_1\beta_1\,\mathbb{E}\left[Y_{1_1}Y_{2_1}\right] + \alpha_1\beta_2\,\mathbb{E}\left[Y_{1_1}Y_{2_2}\right] + ... + \alpha_{k_1}\beta_{k_2}\,\mathbb{E}\left[Y_{1_{k_1}}Y_{2_{k_2}}\right].\end{aligned} \tag{N.3.17}$$

Using the general correlation equation, for $1 \leqslant i \leqslant k_1$ and $1 \leqslant j \leqslant k_2$:

$$\mathbb{E}\left[Y_{1_i}Y_{2_j}\right] = \sigma_{1_i}\sigma_{2_j}\rho_{Y_{1_i}Y_{2_j}} + \mathbb{E}\left[Y_{1_i}\right]\mathbb{E}\left[Y_{2_j}\right], \tag{N.3.18}$$

so that we can rewrite $\rho_{Y_1Y_2}$ as:

$$\begin{aligned}\rho_{Y_1Y_2} &= \frac{\alpha_1\beta_1\left(\sigma_{1_1}\sigma_{2_1}\rho_{Y_{1_1}Y_{2_1}} + \mathbb{E}\left[Y_{1_1}\right]\mathbb{E}\left[Y_{2_1}\right]\right)}{\sigma_1\sigma_2} \\ &\quad + ... + \frac{\alpha_{k_1}\beta_{k_2}\left(\sigma_{1_{k_1}}\sigma_{2_{k_2}}\rho_{Y_{1_{k_1}}Y_{2_{k_2}}} + \mathbb{E}\left[Y_{1_{k_1}}\right]\mathbb{E}\left[Y_{2_{k_2}}\right]\right)}{\sigma_1\sigma_2} \\ &\quad - \frac{\alpha_1\beta_1\,\mathbb{E}\left[Y_{1_1}\right]\mathbb{E}\left[Y_{2_1}\right] + ... + \alpha_{k_1}\beta_{k_2}\,\mathbb{E}\left[Y_{1_{k_1}}\right]\mathbb{E}\left[Y_{2_{k_2}}\right]}{\sigma_1\sigma_2} \\ &= \frac{\sum_{i=1}^{k_1}\alpha_i\sigma_{1_i}\sum_{j=1}^{k_2}\beta_j\sigma_{2_j}\rho_{Y_{1_i},Y_{2_j}}}{\sigma_1\sigma_2}.\end{aligned} \tag{N.3.19}$$

Extending the example from Extension to more than two component distributions, assume there are now three variables: Y_1 (the Normal mixture), Y_2 (the Beta mixture), and Y_3 (a zero-inflated Poisson variable with mean 5 and probability of a structural zero set at 0.1). Let the target correlations among the components of Y_1, the components of Y_2, and Y_3 be 0.4. The components of Y_1 have a weak correlation of 0.1 and the components of Y_2 are independent. The resulting correlation between Y_1 and Y_2 is approximated as:

```
rho <- matrix(0.4, 6, 6)
rho[1:3, 1:3] <- matrix(0.1, 3, 3)
rho[4:5, 4:5] <- matrix(0, 2, 2)
diag(rho) <- 1
```

```
rho_M1M2(mix_pis, mix_mus, mix_sigmas, rho[1:3, 4:5])
## [1] 0.103596
```

Note that `rho` has 6 columns because $k_1 = 3$, $k_2 = 2$, and $k_1 + k_2 + 1 = 6$.

Correlation between continuous mixture variable Y_1 and other random variable Y_3

Here Y_3 can be an ordinal, a continuous non-mixture, or a regular or zero-inflated Poisson or Negative Binomial variable. The correlation between the mixture variable Y_1 and Y_3 is given by:

$$\rho_{Y_1 Y_3} = \frac{\mathbb{E}[Y_1 Y_3] - \mathbb{E}[Y_1]\mathbb{E}[Y_3]}{\sigma_1 \sigma_3}, \tag{N.3.20}$$

where σ_3^2 is the variance of Y_3. Equation N.3.20 requires the expected value of the product of Y_1 and Y_3, which is again approximated by expressing Y_1 as a sum of its component variables:

$$\rho_{Y_1 Y_3} = \frac{\mathbb{E}\left[\left(\sum_{i=1}^{k_1}\alpha_i Y_{1_i}\right) Y_3\right] - \mathbb{E}\left[\sum_{i=1}^{k_1}\alpha_i Y_{1_i}\right]\mathbb{E}[Y_3]}{\sigma_1 \sigma_3}, \tag{N.3.21}$$

where

$$\begin{aligned} \mathbb{E}\left[\left(\sum_{i=1}^{k_1}\alpha_i Y_{1_i}\right) Y_3\right] &= \mathbb{E}\left[\alpha_1 Y_{1_1} Y_3 + \alpha_2 Y_{1_2} Y_3 + ... + \alpha_{k_1} Y_{1_{k_1}} Y_3\right] \\ &= \alpha_1 \mathbb{E}[Y_{1_1} Y_3] + \alpha_2 \mathbb{E}[Y_{1_2} Y_3] + ... + \alpha_{k_1} \mathbb{E}\left[Y_{1_{k_1}} Y_3\right]. \end{aligned} \tag{N.3.22}$$

Using the general correlation equation, for $1 \leqslant i \leqslant k_1$:

$$\mathbb{E}[Y_{1_i} Y_3] = \sigma_{1_i}\sigma_3\rho_{Y_{1_i}Y_3} + \mathbb{E}[Y_{1_i}]\mathbb{E}[Y_3], \tag{N.3.23}$$

so that we can rewrite $\rho_{Y_1 Y_3}$ as:

$$\begin{aligned} \rho_{Y_1 Y_3} = &\frac{\alpha_1\left(\sigma_{1_1}\sigma_3\rho_{Y_{1_1}Y_3} + \mathbb{E}[Y_{1_1}]\mathbb{E}[Y_3]\right) + ... + \alpha_{k_1}\left(\sigma_{1_{k_1}}\sigma_3\rho_{Y_{1_{k_1}}Y_3} + \mathbb{E}\left[Y_{1_{k_1}}\right]\mathbb{E}[Y_3]\right)}{\sigma_1\sigma_3} \\ &- \frac{\alpha_1 \mathbb{E}[Y_{1_1}]\mathbb{E}[Y_3] + ... + \alpha_{k_1}\mathbb{E}\left[Y_{1_{k_1}}\right]\mathbb{E}[Y_3]}{\sigma_1\sigma_3} \\ = &\frac{\sum_{i=1}^{k_1}\alpha_i\sigma_{Y_{1_i}}\rho_{Y_{1_i}Y_3}}{\sigma_1}. \end{aligned} \tag{N.3.24}$$

Continuing with the example, the correlations between Y_1 and Y_3 and between Y_2 and Y_3 are approximated as:

```
rho_M1Y(mix_pis[[1]], mix_mus[[1]], mix_sigmas[[1]], rho[1:3, 6])
## [1] 0.1482236
rho_M1Y(mix_pis[[2]], mix_mus[[2]], mix_sigmas[[2]], rho[4:5, 6])
## [1] 0.2795669
```

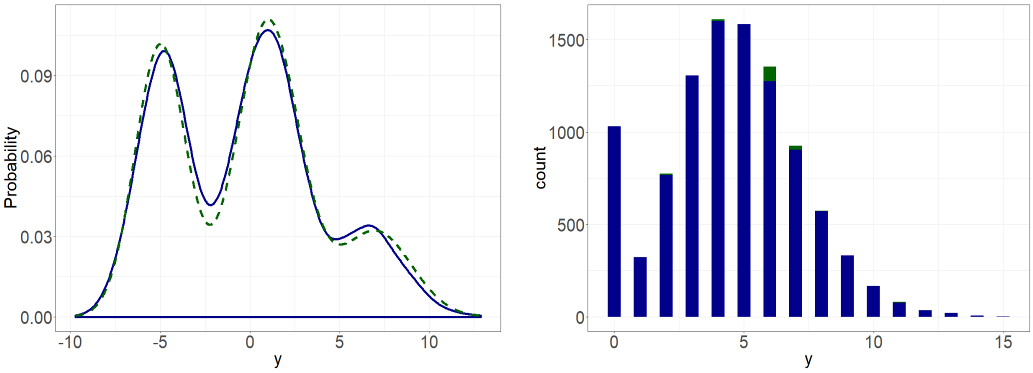
The accuracy of these approximations can be determined through simulation:

```
means <- c(Nstcum[1], Bstcum[1])
vars <- c(Nstcum[2]^2, Bstcum[2]^2)
seed <- 184
Sim1 <- corrvar(n = 100000, k_mix = 2, k_pois = 1, method = "Polynomial",
  means = means, vars = vars, mix_pis = mix_pis, mix_mus = mix_mus,
  mix_sigmas = mix_sigmas, mix_skews = mix_skews, mix_skurts = mix_skurts,
  mix_fifths = mix_fifths, mix_sixths = mix_sixths, lam = 5, p_zip = 0.1,
  rho = rho, seed = seed, use.nearPD = FALSE)
## Total Simulation time: 0.065 minutes
names(Sim1)
## [1] "constants"        "Y_cont"           "Y_comp"           "sixth_correction"
## [5] "valid.pdf"        "Y_mix"            "Y_pois"           "Sigma"
## [9] "Error_Time"       "Time"             "niter"
Sum1 <- summary_var(Y_comp = Sim1$Y_comp, Y_mix = Sim1$Y_mix,
  Y_pois = Sim1$Y_pois, means = means, vars = vars, mix_pis = mix_pis,
  mix_mus = mix_mus, mix_sigmas = mix_sigmas, mix_skews = mix_skews,
  mix_skurts = mix_skurts, mix_fifths = mix_fifths, mix_sixths = mix_sixths,
  lam = 5, p_zip = 0.1, rho = rho)
names(Sum1)
## [1] "cont_sum"   "target_sum" "mix_sum"    "target_mix" "rho_mix"    "pois_sum"
## [7] "rho_calc"   "maxerr"
Sum1$rho_mix
##           [,1]      [,2]      [,3]
## [1,] 1.0000000 0.1012219 0.1475749
## [2,] 0.1012219 1.0000000 0.2776299
## [3,] 0.1475749 0.2776299 1.0000000
```

The results show that Equation N.3.19 and Equation N.3.24 provided good approximations to the simulated correlations. Examples comparing the two simulation pathways also compares approximated expected correlations for continuous mixture variables with simulated correlations.

Figure 2 displays the PDF of the Normal mixture variable and the simulated values of the zero-inflated Poisson (ZIP) variable obtained using **SimCorrMix**'s graphing functions. These functions are written with **ggplot2** functions and the results are `ggplot` objects that can be saved or further modified (Wickham and Chang, 2016). As demonstrated below, the target distribution, specified by distribution name and parameters (39 distributions currently available by name) or PDF function `fx`, can be overlayed on the plot for continuous or count variables.

```
plot_simpdf_theory(sim_y = Sim1$Y_mix[, 1], title = "", sim_size = 2,
  target_size = 2, fx = function(x) mix_pis[[1]][1] *
    dnorm(x, mix_mus[[1]][1], mix_sigmas[[1]][1]) + mix_pis[[1]][2] *
    dnorm(x, mix_mus[[1]][2], mix_sigmas[[1]][2]) + mix_pis[[1]][3] *
    dnorm(x, mix_mus[[1]][3], mix_sigmas[[1]][3]), lower = -10, upper = 10,
  legend.position = "none", axis.text.size = 30, axis.title.size = 30)
plot_simtheory(sim_y = Sim1$Y_pois[, 1], title = "", cont_var = FALSE,
  binwidth = 0.5, Dist = "Poisson", params = c(5, 0.1),
  legend.position = "none", axis.text.size = 30, axis.title.size = 30)
```



(a) PDF of Normal mixture variable.

(b) Simulated values of ZIP variable.

Figure 2: Graphs of variables (simulated = blue, target = green).

The **Continuous Mixture Distributions** vignette explains how to compare simulated to theoretical distributions of continuous mixture variables, as demonstrated here for the Beta mixture variable Y_2 (adapted from Headrick and Kowalchuk, 2007):

1. Obtain the standardized cumulants for the target mixture variable Y_2^* and its components: these were found above using `calc_mixmoments` and `calc_theory`.
2. Obtain the PMT constants for the components of Y_2^*: these are returned in the simulation result `Sim1$constants`.
3. Determine whether these constants produce valid PDF's for the components of Y_2 (and therefore for Y_2): this is indicated for all continuous variables in the simulation result `Sim1$valid.pdf`.

```
## [1] "TRUE" "TRUE" "TRUE" "TRUE" "TRUE"
```

4. Select a critical value from the distribution of Y_2^*, i.e. y_2^* such that $\Pr\left[Y_2^* \geqslant y_2^*\right] = \alpha$, for the desired significance level α: Let $\alpha = 0.05$. Since there are no quantile functions for mixture distributions, determine where the cumulative probability equals $1 - \alpha = 0.95$.

```
beta_fx <- function(x) mix_pis[[2]][1] * dbeta(x, 13, 11) +
  mix_pis[[2]][2] * dbeta(x, 13, 4)
beta_cfx <- function(x, alpha, fx = beta_fx) {
  integrate(function(x, FUN = fx) FUN(x), -Inf, x, subdivisions = 1000,
    stop.on.error = FALSE)$value - (1 - alpha)
}
y2_star <- uniroot(beta_cfx, c(0, 1), tol = 0.001, alpha = 0.05)$root
y2_star
## [1] 0.8985136
```

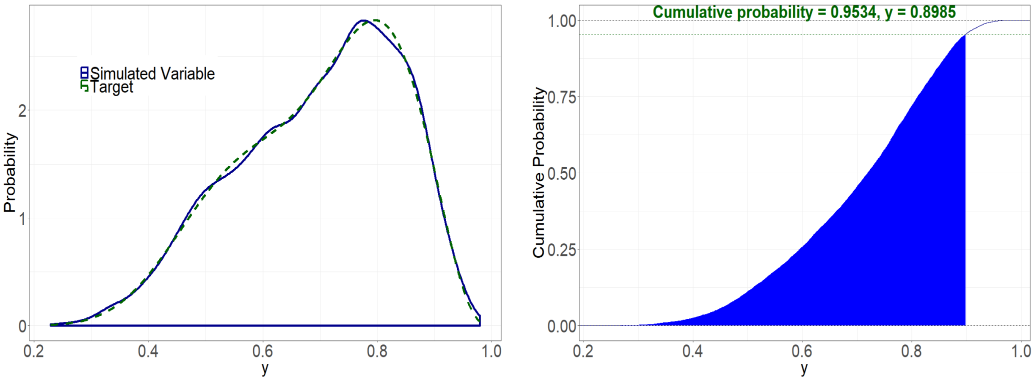
5. Calculate the cumulative probability for the simulated mixture variable Y_2 up to y_2^* and compare to $1 - \alpha$: The function `sim_cdf_prob` from **SimMultiCorrData** calculates cumulative probabilities.

```
sim_cdf_prob(sim_y = Sim1$Y_mix[, 2], delta = y2_star)$cumulative_prob
## [1] 0.9534
```

   This is approximately equal to the $1 - \alpha$ value of 0.95, indicating that the simulation provides a good approximation to the theoretical distribution.

6. Plot a graph of Y_2^* and Y_2: Figure 3 shows the PDF and empirical CDF obtained as follows (`plot_sim_cdf` is in **SimMultiCorrData**):

```
plot_simpdf_theory(sim_y = Sim1$Y_mix[, 2], title = "", sim_size = 2,
  target_size = 2, fx = beta_fx, lower = 0, upper = 1,
  legend.position = c(0.4, 0.85), legend.text.size = 30,
  axis.text.size = 30, axis.title.size = 30)
plot_sim_cdf(sim_y = Sim1$Y_mix[, 2], title = "", calc_cprob = TRUE,
  delta = y2_star, text.size = 30, axis.text.size = 30, axis.title.size = 30)
```



(a) PDF.

(b) CDF.

Figure 3: Graphs of the Beta mixture variable.

Count mixture distributions

SimCorrMix extends the methods in **SimMultiCorrData** for regular Poisson and Negative Binomial (NB) variables to zero-inflated Poisson and NB variables. All count variables are generated using the *inverse CDF method* with distribution functions imported from **VGAM**. The CDF of a standard normal variable has a uniform distribution. The appropriate quantile function F_Y^{-1} is applied to this uniform variable with the designated parameters to generate the count variable: $Y = F_y^{-1}\left(\Phi\left(Z\right)\right)$. The order within all parameters for count variables should be 1^{st} regular and 2^{nd} zero-inflated.

A *zero-inflated random variable* Y_{ZI} is a mixture of a degenerate distribution having the point mass at 0 and another distribution Y that contributes both zero and non-zero values. If the mixing probability is ϕ, then:

$$\Pr\left[Y_{ZI} = 0\right] = \phi + (1 - \phi)\Pr\left[Y = 0\right], \quad 0 < \phi < 1. \qquad \text{(N.4.1)}$$

Therefore, ϕ is the probability of a structural zero, and setting $\phi = 0$ reduces Y_{ZI} to the variable Y. In **SimCorrMix**, Y can have either a Poisson (Y_P) or a Negative Binomial (Y_{NB}) distribution.

Zero-inflated Poisson (ZIP) distribution

The model for $Y_{ZIP} \sim ZIP(\lambda,\ \phi),\ \lambda > 0,\ 0 < \phi < 1$ is:

$$\begin{aligned} \Pr[Y_{ZIP} = 0] &= \phi + (1-\phi)\exp(-\lambda) \\ \Pr[Y_{ZIP} = y] &= (1-\phi)\exp(-\lambda)\frac{\lambda^y}{y!},\quad y = 1, 2, ... \end{aligned} \tag{N.4.2}$$

The mean of Y_{ZIP} is $(1-\phi)\lambda$, and the variance is $\lambda + \lambda^2\phi/(1-\phi)$ (Lambert, 1992). The parameters λ (mean of the regular Poisson component) and ϕ are specified in **SimCorrMix** through the inputs `lam` and `p_zip`. Setting `p_zip = 0` (the default setting) generates a regular Poisson variable.

The *zero-deflated Poisson distribution* is obtained by setting $\phi \in (-1/(\exp(\lambda)-1),\ 0)$, so that the probability of a zero count is less than the nominal Poisson value. In this case, ϕ no longer represents a probability. When $\phi = -1/(\exp(\lambda)-1)$, the random variable has a *positive-Poisson distribution*. The probability of a zero response is 0, and the other probabilities are scaled to sum to 1.

Zero-inflated Negative Binomial (ZINB) distribution

A major limitation of the Poisson distribution is that the mean and variance are equal. In practice, population heterogeneity creates extra variability (*overdispersion*), e.g., if Y represents the number of events which occur in a given time interval and the length of the observation period varies across subjects. If the length of these periods are available for each subject, an offset term may be used. Otherwise, the length can be considered a latent variable and the mean of the Poisson distribution for each subject is a random variable. If these means are described by a Gamma distribution, then Y has a NB distribution, which has an extra parameter to account for overdispersion. However, an excessive number of zeros requires using a zero-inflated distribution. These extra (structural) zeros may arise from a subpopulation of subjects who are not at risk for the event during the study period. These subjects are still important to the analysis because they may possess different characteristics from the at-risk subjects (He et al., 2014).

The model for $Y_{ZINB} \sim ZINB(\eta,\ p,\ \phi),\ \eta > 0,\ 0 < p \leqslant 1,\ \ 0 < \phi < 1$ is:

$$\begin{aligned} \Pr[Y_{ZINB} = 0] &= \phi + (1-\phi)p^{\eta} \\ \Pr[Y_{ZINB} = y] &= (1-\phi)\frac{\Gamma(y+\eta)}{\Gamma(\eta)\,y!}p^{\eta}(1-p)^{\eta},\quad y = 1, 2, ... \end{aligned} \tag{N.4.3}$$

In this formulation, the Negative Binomial component Y_{NB} represents the number of failures that occur in a sequence of independent Bernoulli trials before a target number of successes (η) is reached. The probability of success in each trial is p. Y_{NB} may also be parameterized by the mean μ (of the regular NB component) and dispersion parameter η so that $p = \eta/(\eta+\mu)$ or $\mu = \eta(1-p)/p$. The mean of Y_{ZINB} is $(1-\phi)\mu$, and the variance is $(1-\phi)\mu(1+\mu(\phi+1/\eta))$ (Ismail and Zamani, 2013; Zhang et al., 2016). The parameters η, p, μ, and ϕ are specified through the inputs `size`, `prob`, `mu`, and `p_zinb`. Either `prob` or `mu` should be given for all NB and ZINB variables. Setting `p_zinb = 0` (the default setting) generates a regular NB variable.

The *zero-deflated NB distribution* may be obtained by setting $\phi \in (-p^{\eta}/(1-p^{\eta}),\ 0)$, so that the probability of a zero count is less than the nominal NB value. In this case, ϕ no longer represents a probability. The *positive-NB distribution* results when $\phi = -p^{\eta}/(1-p^{\eta})$. The probability of a zero response is 0, and the other probabilities are scaled to sum to 1.

Calculation of intermediate correlations for count variables

The two simulation pathways differ by the technique used for count variables. The intermediate correlations used in correlation method 1 are simulation based and accuracy increases with sample size and number of repetitions. The intermediate correlations used in correlation method 2 involve correction loops which make iterative adjustments until a maximum error has been reached (if possible). Correlation method 1 is described below:

1. Count variable pairs: Based on Yahav and Shmueli (2012)'s method, the intermediate correlation between the standard normal variables Z_1 and Z_2 is calculated using a logarithmic transformation of the target correlation. First, the upper and lower Fréchet-Hoeffding bounds

(mincor, maxcor) on $\rho_{Y_1Y_2}$ are simulated (see Calculation of correlation boundaries; Fréchet, 1957; Hoeffding, 1994). Then the intermediate correlation $\rho_{Z_1Z_2}$ is found as follows:

$$\rho_{Z_1Z_2} = \frac{1}{b}\log\left(\frac{\rho_{Y_1Y_2} - c}{a}\right), \tag{N.4.4}$$

where

$$a = -\frac{\text{maxcor} * \text{mincor}}{\text{maxcor} + \text{mincor}}, \quad b = \log\left(\frac{\text{maxcor} + a}{a}\right), \quad c = -a.$$

The functions `intercorr_pois`, `intercorr_nb`, and `intercorr_pois_nb` calculate these correlations.

2. Ordinal-count variable pairs: Extending Amatya and Demirtas (2015)'s method, the intermediate correlations are the ratio of the target correlations to correction factors. The correction factor is the product of the upper Fréchet-Hoeffding bound on the correlation between the count variable and the normal variable used to generate it and a simulated upper bound on the correlation between an ordinal variable and the normal variable used to generate it. This upper bound is Demirtas and Hedeker (2011)'s generate, sort, and correlate (GSC) upper bound (see Calculation of correlation boundaries). The functions `intercorr_cat_pois` and `intercorr_cat_nb` calculate these correlations.

3. Continuous-count variable pairs: Extending Amatya and Demirtas (2015)'s and Demirtas and Hedeker (2011)'s methods, the correlation correction factor is the product of the upper Fréchet-Hoeffding bound on the correlation between the count variable and the normal variable used to generate it and the power method correlation between the continuous variable and the normal variable used to generate it. This power method correlation is given by $\rho_{p(Z)Z} = c_1 + 3c_3 + 15c_5$, where $c_3 = 0$ for Fleishman's method (Headrick and Kowalchuk, 2007). The functions `intercorr_cont_pois` and `intercorr_cont_nb` calculate these correlations.

Fialkowski and Tiwari (2017) showed that this method is less accurate for positive correlations with small count variable means (i.e., less than 1) or high negative correlations with large count variable means.

In correlation method 2, count variables are treated as "ordinal" variables, based on the methods of Barbiero and Ferrari (Ferrari and Barbiero, 2012; Barbiero and Ferrari, 2015a). The Poisson or NB support is made finite by removing a small user-specified value (specified by `pois_eps` and `nb_eps`) from the total cumulative probability. This truncation factor may differ for each count variable, but the default value is 0.0001 (suggested by Barbiero and Ferrari, 2015a). For example, `pois_eps = 0.0001` means that the support values removed have a total probability of 0.0001 of occurring in the distribution of that variable. The effect is to remove improbable values, which may be of concern if the user wishes to replicate a distribution with outliers. The function `maxcount_support` creates these new supports and associated marginal distributions.

1. Count variable or ordinal-count variable pairs: The intermediate correlations are calculated using the correction loop of `ord_norm` (see Simulation of ordinal variables).

2. Continuous-count variable pairs: Extending Demirtas et al. (2012)'s method, the intermediate correlations are the ratio of the target correlations to correction factors. The correction factor is the product of the power method correlation between the continuous variable and the normal variable used to generate it and the point-polyserial correlation between the ordinalized count variable and the normal variable used to generate it (Olsson et al., 1982). The functions `intercorr_cont_pois2` and `intercorr_cont_nb2` calculate these correlations.

This method performs best under the same circumstances as ordinal variables, i.e., when there are few categories and the probability of any given category is not very small. This occurs when the count variable has a small mean. Therefore, method 2 performs well in situations when method 1 has poor accuracy. In contrast, large means for the count variables would result in longer computational times. Examples comparing the two simulation pathways compares the accuracy of correlation methods 1 and 2 under different scenarios.

Simulation of ordinal variables

Ordinal variables ($r \geqslant 2$ categories) are generated by discretizing standard normal variables at the quantiles determined from the cumulative probabilities specified in `marginal`. Each element of this list is a vector of length $r - 1$ (the r^{th} value is 1). If the support is not provided, the default is to use $\{1,\ 2,\ ...,\ r\}$ (Ferrari and Barbiero, 2012). The *tetrachoric* correlation is used for the intermediate correlation of binary pairs (Emrich and Piedmonte, 1991; Demirtas et al., 2012). The assumptions are that the binary variables arise from latent normal variables and the actual trait is continuous and not discrete. For Y_1 and Y_2, with success probabilities p_1 and p_2, the intermediate correlation $\rho_{Z_1 Z_2}$ is the solution to the following equation:

$$\Phi\left[z\left(p_1\right),\ z\left(p_2\right),\ \rho_{Z_1 Z_2}\right] = \rho_{Y1Y2}\sqrt{p_1\left(1-p_1\right)p_2\left(1-p_2\right)} + p_1 p_2, \tag{N.5.1}$$

where $z\left(p\right)$ indicates the p^{th} quantile of the standard normal distribution.

If at least one ordinal variable has more than 2 categories, `ord_norm` is called. Based on **SimMultiCorrData**'s `ordnorm` and **GenOrd**'s `ordcont` and `contord`, the algorithm to simulate `k_cat` ordinal random variables with target correlation matrix `rho0` is as follows:

1. Create the default `support` if necessary.
2. Use `norm_ord` to calculate the initial correlation of the ordinal variables (`rhoordold`) generated by discretizing `k_cat` random normal variates with correlation matrix set equal to `rho0`, using `marginal` and the corresponding normal quantiles. These correlations are calculated using means and variances found from multivariate normal probabilities determined by **mvtnorm**'s `pmvnorm` (Genz et al., 2017; Genz and Bretz, 2009).
3. Let `rho` be the intermediate normal correlation updated in each iteration, `rhoord` be the ordinal correlation calculated in each iteration, `rhoold` be the intermediate correlation from the previous iteration (initialized at `rhoordold`), `it` be the iteration number, and `maxit` and `epsilon` be the user-specified maximum number of iterations and pairwise correlation error. For each variable pair, execute the following:
   (a) If `rho0 = 0`, set `rho = 0`.
   (b) While the absolute error between `rhoord` and `rho0` is greater than `epsilon` and `it` is less than `maxit`:
      i. If `rho0 * (rho0/rhoord) <= -1`:
      `rho = rhoold * (1 + 0.1 * (1 -rhoold) * -sign(rho0 -rhoord))`.
      ii. If `rho0 * (rho0/rhoord) >= 1`:
      `rho = rhoold * (1 + 0.1 * (1 -rhoold) * sign(rho0 -rhoord))`.
      iii. Else, `rho = rhoold * (rho0/rhoord)`.
      iv. If `rho >1`, set `rho = 1`. If `rho <-1`, set `rho = -1`.
      v. Calculate `rhoord` using `norm_ord` and the 2 × 2 correlation matrix formed by `rho`.
      vi. Set `rhoold = rho` and increase `it` by 1.
   (c) Store the number of iterations in the matrix `niter`.
4. Return the final intermediate correlation matrix `SigmaC = rho` for the random normal variables. Discretize these to produce ordinal variables with the desired correlation matrix.

Calculation of correlation boundaries

For binary variable pairs, the correlation bounds are calculated as by Demirtas et al. (2012). The joint distribution is determined using the moments of a multivariate normal distribution (Emrich and Piedmonte, 1991). For Y_1 and Y_2, with success probabilities p_1 and p_2, the boundaries are approximated by:

$$\left\{\max\left(-\sqrt{\frac{p_1 p_2}{q_1 q_2}},\ -\sqrt{\frac{q_1 q_2}{p_1 p_2}}\right),\ \min\left(\sqrt{\frac{p_1 q_2}{q_1 p_2}},\ \sqrt{\frac{q_1 p_2}{p_1 q_2}}\right)\right\}, \tag{N.6.1}$$

where $q_1 = 1 - p_1$ and $q_2 = 1 - p_2$. If one of an ordinal variable pair has more than 2 categories, randomly generated variables with the given `marginal` distributions and `support` values are used in Demirtas and Hedeker (2011)'s generate, sort, and correlate (GSC) algorithm. A large number (default $100,000$) of independent random samples from the desired distributions are generated. The lower bound is the sample correlation of the two variables sorted in opposite directions (i.e., one increasing and one decreasing). The upper bound is the sample correlation of the two variables sorted in the same direction.

The GSC algorithm is also used for continuous, continuous-ordinal, ordinal-count, and continuous-count variable pairs. Since count variables are treated as "ordinal" in correlation method 2, the correlation bounds for count variable pairs is found with the GSC algorithm after creating finite supports with associated marginal distributions (with `maxcount_support`). The correlation bounds for count variable pairs in correlation method 1 are the Fréchet-Hoeffding bounds (Fréchet, 1957;

Hoeffding, 1994). For two random variables Y_1 and Y_2 with CDF's F_1 and F_2, the correlation bounds are approximated by:

$$\left\{\text{Cor}\left(F_1^{-1}(U), F_2^{-1}(1-U)\right),\ \text{Cor}\left(F_1^{-1}(U), F_2^{-1}(U)\right)\right\}, \tag{N.6.2}$$

where U is a Uniform(0, 1) random variable of default length $100,000$.

Example with multiple variable types

Consider the Normal and Beta mixture variables from Continuous mixture distributions. Additional variables are an ordinal variable with three equally-weighted categories (e.g., drug treatment), two zero-inflated Poisson variables with means 0.5 and 1 (for the regular Poisson components) and structural zero probabilities 0.1 and 0.2, and two zero-inflated NB variables with means 0.5 and 1 (for the regular NB components), success probabilities 0.8 and 0.6, and structural zero probabilities 0.1 and 0.2. The target pairwise correlation is set at -0.5. The components of the Normal mixture variable again have weak correlation of 0.1 and those for the Beta mixture variable are uncorrelated. The parameter inputs are first checked with `validpar`.

```
marginal <- list(c(1/3, 2/3))
support <- list(c(0, 1, 2))
lam <- c(0.5, 1)
p_zip <- c(0.1, 0.2)
mu <- c(0.5, 1)
prob <- c(0.8, 0.6)
size <- prob * mu/(1 - prob)
p_zinb <- c(0.1, 0.2)
rho <- matrix(-0.5, 10, 10)
rho[2:4, 2:4] <- matrix(0.1, 3, 3)
rho[5:6, 5:6] <- matrix(0, 2, 2)
diag(rho) <- 1
validpar(k_cat = 1, k_mix = 2, k_pois = 2, k_nb = 2, method = "Polynomial",
  means = means, vars = vars, mix_pis = mix_pis, mix_mus = mix_mus,
  mix_sigmas = mix_sigmas, mix_skews = mix_skews, mix_skurts = mix_skurts,
  mix_fifths = mix_fifths, mix_sixths = mix_sixths, marginal = marginal,
  support = support, lam = lam, p_zip = p_zip, size = size, mu = mu,
  p_zinb = p_zinb, rho = rho)
## Default of pois_eps = 0.0001 will be used for Poisson variables
##               if using correlation method 2.
## Default of nb_eps = 0.0001 will be used for NB variables
##               if using correlation method 2.
Target correlation matrix is not positive definite.
## [1] TRUE
valid1 <- validcorr(10000, k_cat = 1, k_mix = 2, k_pois = 2, k_nb = 2,
  method = "Polynomial", means = means, vars = vars, mix_pis = mix_pis,
  mix_mus = mix_mus, mix_sigmas = mix_sigmas, mix_skews = mix_skews,
  mix_skurts = mix_skurts, mix_fifths = mix_fifths, mix_sixths = mix_sixths,
  marginal = marginal, lam = lam, p_zip = p_zip, size = size, mu = mu,
  p_zinb = p_zinb, rho = rho, use.nearPD = FALSE, quiet = TRUE)
## Range error! Corr[ 7 , 9 ] must be between -0.388605 and 0.944974
## Range error! Corr[ 7 , 10 ] must be between -0.432762 and 0.925402
## Range error! Corr[ 8 , 9 ] must be between -0.481863 and 0.877668
## Range error! Corr[ 9 , 10 ] must be between -0.386399 and 0.937699
names(valid1)
## [1] "rho"              "L_rho"            "U_rho"            "constants"
## [5] "sixth_correction" "valid.pdf"        "valid.rho"
valid2 <- validcorr2(10000, k_cat = 1, k_mix = 2, k_pois = 2, k_nb = 2,
  method = "Polynomial", means = means, vars = vars, mix_pis = mix_pis,
  mix_mus = mix_mus, mix_sigmas = mix_sigmas, mix_skews = mix_skews,
  mix_skurts = mix_skurts, mix_fifths = mix_fifths, mix_sixths = mix_sixths,
  marginal = marginal, lam = lam, p_zip = p_zip, size = size, mu = mu,
  p_zinb = p_zinb, rho = rho, use.nearPD = FALSE, quiet = TRUE)
## Range error! Corr[ 7 , 9 ] must be between -0.385727 and 0.947462
## Range error! Corr[ 7 , 10 ] must be between -0.428145 and 0.921001
## Range error! Corr[ 8 , 9 ] must be between -0.477963 and 0.879439
## Range error! Corr[ 9 , 10 ] must be between -0.384557 and 0.939524
```

The `validpar` function indicates that all parameter inputs have the correct format and the default cumulative probability truncation value of 0.0001 will be used in correlation method 2 for `pois_eps` and `nb_eps`. Since `rho` is not PD, the intermediate correlation matrix `Sigma` will probably also be non-PD. The user has three choices: 1) convert `rho` to the nearest PD matrix before simulation, 2) set `use.nearPD = TRUE` (default) in the simulation functions to convert `Sigma` to the nearest PD matrix during simulation, or 3) set `use.nearPD = FALSE` in the simulation functions to replace negative eigenvalues with 0. Using `use.nearPD = TRUE` in `validcorr` or `validcorr2` converts `rho` to the nearest PD matrix before checking if all pairwise correlations fall within the feasible boundaries, whereas `use.nearPD = FALSE` checks the initial matrix `rho`. Setting `quiet = TRUE` keeps the non-PD message from being reprinted.

Range violations occur with the count variables. The lower and upper correlation bounds are given in the list components `L_rho` and `U_rho`. Note that these are *pairwise* correlation bounds. Although `valid.rho` will return `TRUE` if all elements of `rho` are within these bounds, this does not guarantee that the overall target correlation matrix `rho` can be obtained in simulation.

Overall workflow for generation of correlated data

The vignette **Overall Workflow for Generation of Correlated Data** provides a detailed step-by-step guideline for correlated data simulation with examples for `corrvar` and `corrvar2`. These steps are briefly reviewed here.

1. Obtain the distributional parameters for the desired variables.
   (a) Continuous variables: For non-mixture and components of mixture variables, these are skew, skurtosis, plus standardized fifth and sixth cumulants (for `method = "Polynomial"`) and sixth cumulant corrections (if desired). The inputs are `skews`, `skurts`, `fifths`, `sixths`, and `Six` for non-mixture variables; `mix_skews`, `mix_skurts`, `mix_fifths`, `mix_sixths`, and `mix_Six` for components of mixture variables. If the goal is to simulate a theoretical distribution, **SimMultiCorrData**'s `calc_theory` will return these values given a distribution's name and parameters (39 distributions currently available by name) or PDF function `fx`. If the goal is to mimic a real data set, **SimMultiCorrData**'s `calc_moments` uses the method of moments or `calc_fisherk` uses Fisher (1929)'s k-statistics given a vector of data. For mixture variables, the mixing parameters (`mix_pis`), component means (`mix_mus`), and component standard deviations (`mix_sigmas`) are also required. The means and variances of non-mixture and mixture variables are specified in `means` and `vars` and these can be found using `calc_mixmoments` for mixture variables.
   (b) Ordinal variables: The cumulative marginal probabilities in `marginal` and support values in `support` as described in Simulation of ordinal variables.
   (c) Poisson variables: The mean values in `lam` and probabilities of structural zeros in `p_zip` (default of 0 to yield regular Poisson distributions). The mean refers to the mean of the Poisson component of the distribution. For correlation method 2, also cumulative probability truncation values in `pois_eps`.
   (d) NB variables: The target number of successes in `size`, probabilities of structural zeros in `p_zinb` (default of 0 to yield regular NB distributions), plus means in `mu` or success probabilities in `prob`. The mean refers to the mean of the NB component of the distribution. For correlation method 2, also cumulative probability truncation values in `nb_eps`.
2. Check that all parameter inputs have the correct format using `validpar`. Incorrect parameter specification is the most likely cause of function failure.
3. If continuous variables are desired, verify that the skurtoses are greater than the lower skurtoses bounds using **SimMultiCorrData**'s `calc_lower_skurt`. The function permits a skurtosis correction vector to aid in discovering a lower bound associated with PMT constants that yield a valid PDF. Since this step can take considerable time, the user may wish to do this at the end if any of the variables have invalid PDF's. The sixth cumulant value should be the actual sixth cumulant used in simulation, i.e., the distribution's sixth cumulant plus any necessary correction (if `method = "Polynomial"`).
4. Check if the target correlation matrix `rho` falls within the feasible correlation boundaries. The variables in `rho` must be ordered correctly (see Introduction).
5. Generate the variables using either `corrvar` or `corrvar2`, with or without the error loop.
6. Summarize the results numerically with `summary_var` or graphically with `plot_simpdf_theory`, `plot_simtheory`, or any of the plotting functions in **SimMultiCorrData**.

Examples comparing the two simulation pathways

Correlation methods 1 and 2 were compared to demonstrate situations when each has greater simulation accuracy. In scenario A, the ordinal (O1), Normal mixture (Nmix with components N1, N2, and N3), Beta mixture (Bmix with components B1 and B2), two zero-inflated Poisson (P1 and P2), and two zero-inflated NB (NB1 and NB2) variables from the Calculation of correlation boundaries example were simulated. All count variables in this case had small means (less than 1). In scenario B, the two Poisson variables were replaced with two zero-inflated NB (NB3 and NB4) variables with means 50 and 100 (for the regular NB components), success probabilities 0.4 and 0.2, and structural zero probabilities 0.1 and 0.2. This yielded two count variables with small means and two with large means. The simulations were done with $\texttt{n} = 10,000$ sample size and $\texttt{r} = 1,000$ repetitions using three different positive correlations as given in Table 1 (chosen based on the upper correlation bounds). The correlation among N1, N2, N3 was set at 0.1; the correlation between B1 and B2 was set at 0. The default total cumulative probability truncation value of 0.0001 was used for each count variable in `corrvar2`.

In scenarios A and B, the simulated correlations among the count variables were compared to the target values using boxplots generated with **ggplot2**'s `geom_boxplot`. In scenario A, the simulated correlations with the continuous mixture variables were compared to the expected correlations approximated by `rho_M1M2` and `rho_M1Y`, with O1 as the non-mixture variable. Simulation times included simulation of the variables only with `corrvar` or `corrvar2`. Medians and interquartile ranges (IQR) were computed for the summary tables. Variable summaries are given for Nmix, Bmix, and NB1–NB4 in scenario B. This example was run on R version 3.4.1 with **SimCorrMix** version 0.1.0 using CentOS. The complete code is in the supplementary file for this article.

Results

Table 1 gives the three different correlations and total simulation times (1,000 repetitions) for correlation method 1 using `corrvar` (Time M_1) and correlation method 2 using `corrvar2` (Time M_2). The strong correlation was different between NB variables with small means (NB1, NB2) and NB variables with large means (NB3, NB4) because the upper bounds were lower for these variable pairs.

Scenario			A: Poisson and NB		B: NB	
Correlation Type	ρ	ρ^*	Time M_1	Time M_2	Time M_1	Time M_2
Strong	0.7	0.6	2.55	2.03	2.00	9.30
Moderate	0.5	0.5	1.65	0.92	1.98	8.01
Weak	0.3	0.3	1.39	0.90	1.95	5.78

Table 1: Six comparisons and total simulation times for method 1 (M_1) and method 2 (M_2) in hours. Correlation ρ^* applied to the NB1–NB3, NB1–NB4, NB2–NB3, and NB2–NB4 variable pairs.

The strong correlations required the most time for each correlation method. Although method 2 was faster when all count variables had small means (scenario A), it was notably slower when two of the count variables had large means (scenario B). The reason is that method 2 treats all count variables as "ordinal," which requires creating finite supports and associated marginal distributions, as described in Calculation of intermediate correlations for count variables. When a count variable has a large mean, there are several support values with very small probabilities, making simulation more difficult.

Scenario A: Ordinal, Normal and Beta mixtures, Poisson, and NB variables

Figure 4 contains boxplots of the simulated correlations for the continuous mixture variables. Method 1 is in red; method 2 is in green. The middle line is the median (50^{th} percentile); the lower and upper hinges correspond to the first and third quartiles (the 25^{th} and 75^{th} percentiles). The upper whisker extends from the hinge to the largest value up to 1.5 * IQR from the hinge. The lower whisker extends from the hinge to the smallest value at most 1.5 * IQR from the hinge. Data beyond the end of the whiskers are considered "outliers." The black horizontal lines show the approximate expected values obtained with the functions `rho_M1M2` and `rho_M1Y` (also given in Table 2).

Correlation Type	ρ	$\rho_{\text{Nmix,Bmix}}$	$\rho_{\text{Nmix,O1}}$	$\rho_{\text{Bmix,O1}}$
Strong	0.7	0.1813	0.2594	0.4892
Moderate	0.5	0.1295	0.1853	0.3495
Weak	0.3	0.0777	0.1112	0.2097

Table 2: Approximate expected correlations with the continuous mixture variables.

Notice in Table 2 that the expected correlations are much smaller than the pairwise correlations, demonstrating an important consideration when setting the correlations for mixture components. Even though the strong correlation between the components of Nmix and the components of Bmix was set at 0.7, the expected correlation between Nmix and Bmix was only 0.1813. Combining continuous components into one continuous mixture variable always decreases the absolute correlation between the mixture variable and other variables.

Figure 4 shows that, as expected, the results with correlation methods 1 and 2 were similar, since the methods differ according to count variable correlations. The simulated correlations were farthest from the approximate expected values with the strong correlation and closest for the weak correlation. In the simulations with strong or moderate correlations, the intermediate correlation matrix `Sigma` was not PD due to the weak correlation (0.1) between N1, N2, and N3 and independence (zero correlation) of B1 and B2. During simulation, after `Sigma` is calculated with `intercorr` or `intercorr2`, eigenvalue decomposition is done on `Sigma`. The square roots of the eigenvalues form a diagonal matrix. The product of the eigenvectors, diagonal matrix, and transposed standard normal variables produces normal variables with the desired intermediate correlations. If `Sigma` is not PD and `use.nearPD` is set to `FALSE` in the simulation functions, negative eigenvalues are replaced with 0 before forming the diagonal matrix of eigenvalue square roots. If `use.nearPD` is set to `TRUE` (default), `Sigma` is replaced with the nearest PD matrix using (Higham, 2002)'s algorithm and **Matrix**'s `nearPD` function. Either method increases correlation errors because the resulting intermediate correlations are different from those found in `Sigma`. As the maximum absolute correlation in the target matrix `rho` increases, these differences increase. In this example, the `Sigma` matrix had two negative eigenvalues in the strong correlation simulations and one negative eigenvalue in the moderate correlation simulations. This is why the correlation errors were largest for the strong correlation setting.

Figure 5 shows boxplots of the simulated correlations for the count variables. The horizontal lines show the target values. These correlations were also affected by the adjusted eigenvalues and the errors for the strong correlations were again the largest. Correlation method 2 performed better in each case except when generating $\rho_{\text{P1,NB1}}$ in the strong correlation case. Barbiero and Ferrari (2015a)'s method of treating count variables as "ordinal" is expected to exhibit better accuracy than Yahav and Shmueli (2012)'s equation when the count variables have small means (less than 1). Tables 6–8 in the Appendix provide median (IQR) correlation errors for all variables and each correlation type.

Scenario B: Ordinal, Normal and Beta mixtures, and NB variables

Tables 3 and 4 describe the target and simulated distributions for Nmix, Bmix, and NB1–NB4 in the weak correlation case. In all instances, the simulated distributions are close to the target

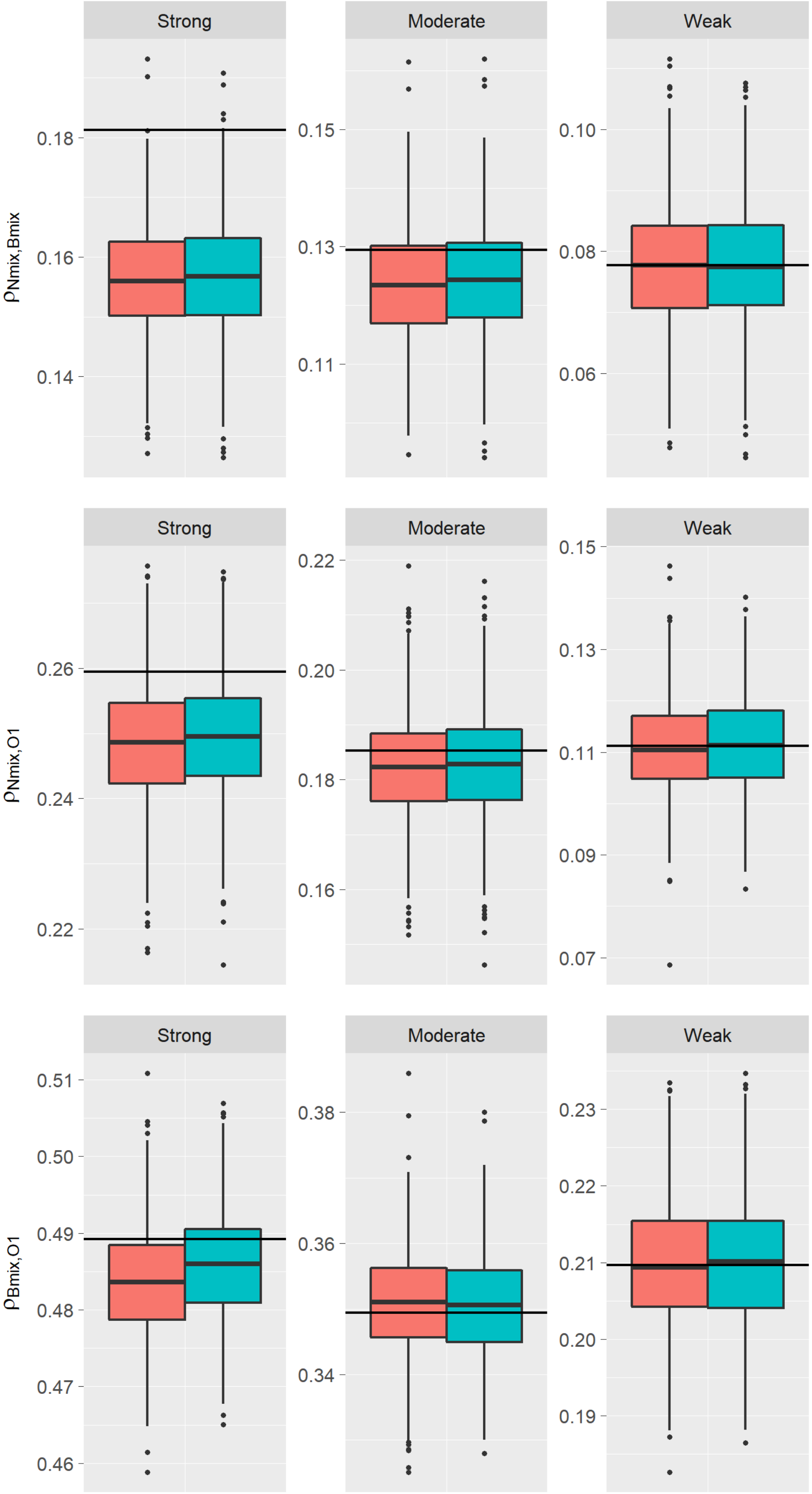


Figure 4: Boxplots of simulated correlations for continuous mixture variables (scenario A). Method 1 is in red; method 2 is in green. The horizontal lines show the approximate expected values.

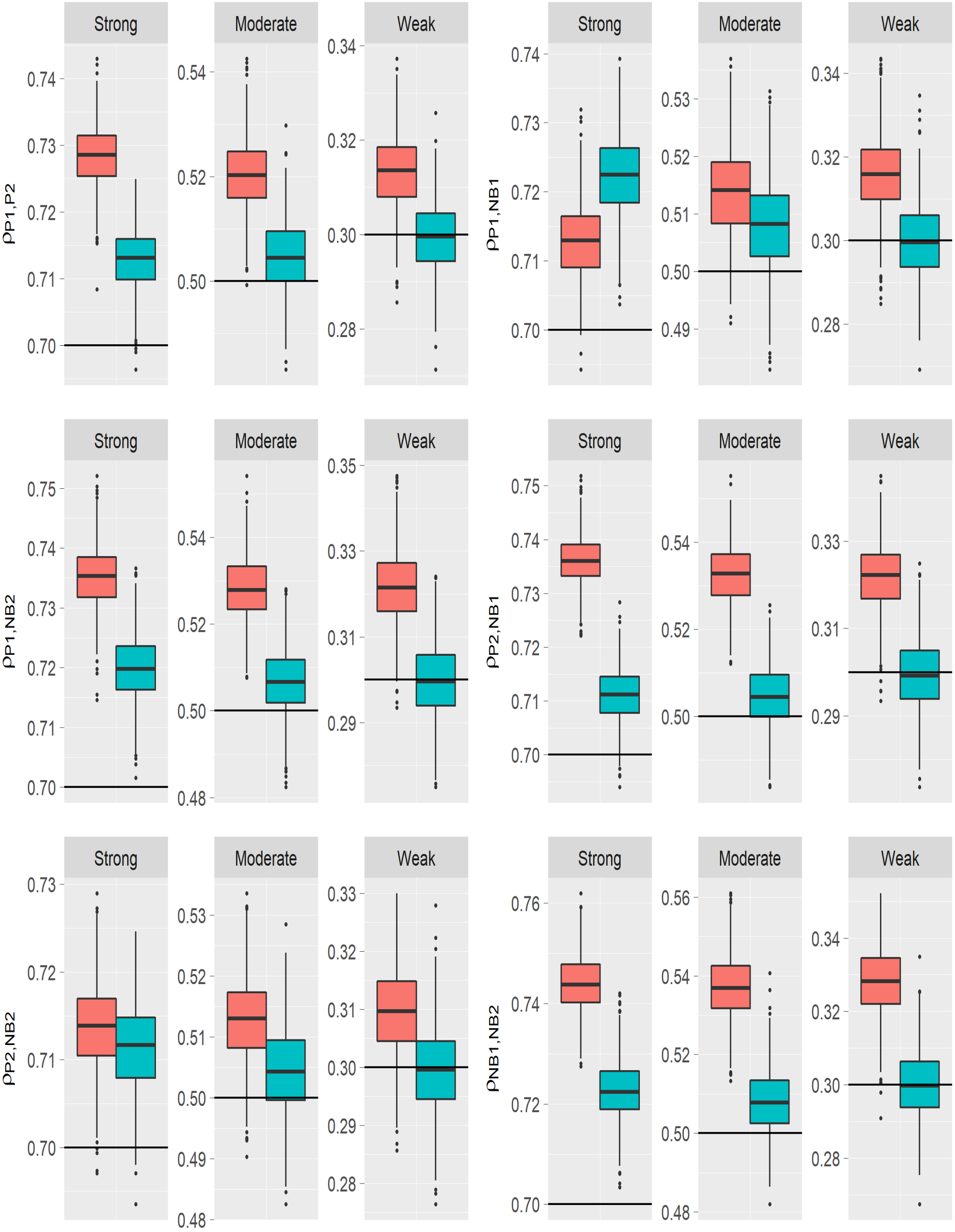


Figure 5: Boxplots of simulated correlations for P1, P2, NB1, and NB2 (scenario A). Method 1 is in red; method 2 is in green. The horizontal lines show the target values.

distributions.

	Nmix		Bmix	
Mean	-0.20	-0.20 (-0.20, -0.20)	0.70	0.70 (0.70, 0.70)
SD	4.48	4.48 (4.48, 4.48)	0.14	0.14 (0.14, 0.14)
Skew	0.33	0.33 (0.32, 0.33)	-0.46	-0.46 (-0.47, -0.45)
Skurtosis	-0.62	-0.62 (-0.64, -0.61)	-0.54	-0.54 (-0.56, -0.52)
Fifth	-1.02	-1.03 (-1.07, -0.98)	1.72	1.73 (1.68, 1.77)
Sixth	1.49	1.50 (1.36, 1.62)	0.56	0.54 (0.37, 0.72)

Table 3: Target and median (IQR) simulated distributions of continuous mixture variables.

	$\mathrm{P}[Y=0]$	$\mathbb{E}(\mathrm{P}[Y=0])$	Mean	$\mathbb{E}$[Mean]
NB1	0.68 (0.67, 0.68)	0.68	0.45 (0.45, 0.45)	0.45
NB2	0.57 (0.57, 0.57)	0.57	0.80 (0.80, 0.80)	0.80
NB3	0.10 (0.10, 0.10)	0.10	45.00 (44.96, 45.03)	45.00
NB4	0.20 (0.20, 0.20)	0.20	80.00 (79.90, 80.10)	80.00
	Var	$\mathbb{E}$[Var]	Median	Max
NB1	0.58 (0.58, 0.59)	0.58	0 (0, 0)	7 (6, 7)
NB2	1.49 (1.48, 1.51)	1.49	0 (0, 0)	11 (10, 12)
NB3	337.76 (335.43, 339.67)	337.50	48 (48, 48)	101 (98, 105)
NB4	2000.09 (1990.21, 2010.18)	2000.00	92 (91, 92)	204 (199, 212)

Table 4: Target and median (IQR) simulated distributions of zero-inflated NB variables.

Figure 6 shows boxplots of the simulated correlations for the count variables. The horizontal lines show the target values. Method 1 performed better for all strong correlation cases except between the two NB variables with small means (NB1 and NB2). Although method 2 had smaller errors overall, it did require considerably longer simulation times. Therefore, the user should consider using correlation method 1 when the data set contains count variables with large means. Tables 9–11 in the Appendix provide median (IQR) correlation errors for all variables and each correlation type.

Summary

The package **SimCorrMix** generates correlated continuous (normal, non-normal, and mixture), ordinal ($r \geqslant 2$ categories), and count (regular or zero-inflated, Poisson or Negative Binomial) variables. It is a significant contribution to existing R simulation packages because it is the first to include continuous and count mixture variables in correlated data sets. Since **SimCorrMix** simulates variables which mimic real-world data sets and provides great flexibility, the package has a wide range of applications in clinical trial and genetic studies. The simulated data sets could be used to compare statistical methods, conduct hypothesis tests, perform bootstrapping, or calculate power. The two simulation pathways, excecuted by the functions `corrvar` and `corrvar2`, permit the user to accurately reproduce desired correlation matrices for different parameter ranges. Correlation method 1 should be used when the target distributions include count variables with large means, and correlation method 2 is preferable in opposite situations. The package also provides helper functions to calculate standardized cumulants of continuous mixture variables, approximate expected correlations with continuous mixture variables, validate parameter inputs, determine feasible correlation boundaries, and summarize simulation results numerically and graphically. Future extensions of the package include adding more variable types (e.g., zero-inflated Binomial, Gaussian, and Gamma).

Supplementary Material

The article's supplementary file contains replication code for the examples in the paper and Examples comparing the two simulation pathways.

Acknowledgments

This research serves as part of Allison Fialkowski's dissertation, which was made possible by grant $T32HL079888$ from the National Heart, Lung, and Blood Institute of the National Institute of Health, USA and Dr. Hemant K. Tiwari's William "Student" Sealy Gosset Professorship Endowment. I would like to thank my dissertation mentor, Hemant K. Tiwari, PhD; and committee members T. Mark Beasley, PhD; Charles R. Katholi, PhD; Nita A. Limdi, PhD; M. Ryan Irvin, PhD; and Nengjun Yi, PhD.

Bibliography

A. Amatya and H. Demirtas. Simultaneous generation of multivariate mixed data with Poisson and normal marginals. *Journal of Statistical Computation and Simulation*, 85(15):3129–3139, 2015. URL `https://doi.org/10.1080/00949655.2014.953534`. [p261]

D. Ardia. *AdMit: Adaptive Mixture of Student-t Distributions*, 2017. URL `https://CRAN.R-project.org/package=AdMit`. R package version 2.1.3. [p250]

L. M. Avila, M. R. May, and J. Ross-Ibarra. *DPP: Inference of Parameters of Normal Distributions from a Mixture of Normals*, 2017. URL `https://CRAN.R-project.org/package=DPP`. R package version 0.1.1. [p250]

A. A. Baghban, A. Pourhoseingholi, F. Zayeri, A. A. Jafari, and S. M. Alavian. Application of zero-inflated Poisson mixed models in prognostic factors of Hepatitis C. *BioMed Research International*, 2013. URL `https://doi.org/10.1155/2013/403151`. [p250]

O. G. Bahcall. Complex traits: Genetic discovery, heritability and prediction. *Nature Reviews Genetics*, 16(257), 2015. URL `https://doi.org/10.1038/nrg3947`. [p253]

E. Balderama and T. Trippe. *Hurdlr: Zero-Inflated and Hurdle Modelling Using Bayesian Inference*, 2017. URL `https://CRAN.R-project.org/package=hurdlr`. R package version 0.1. [p251]

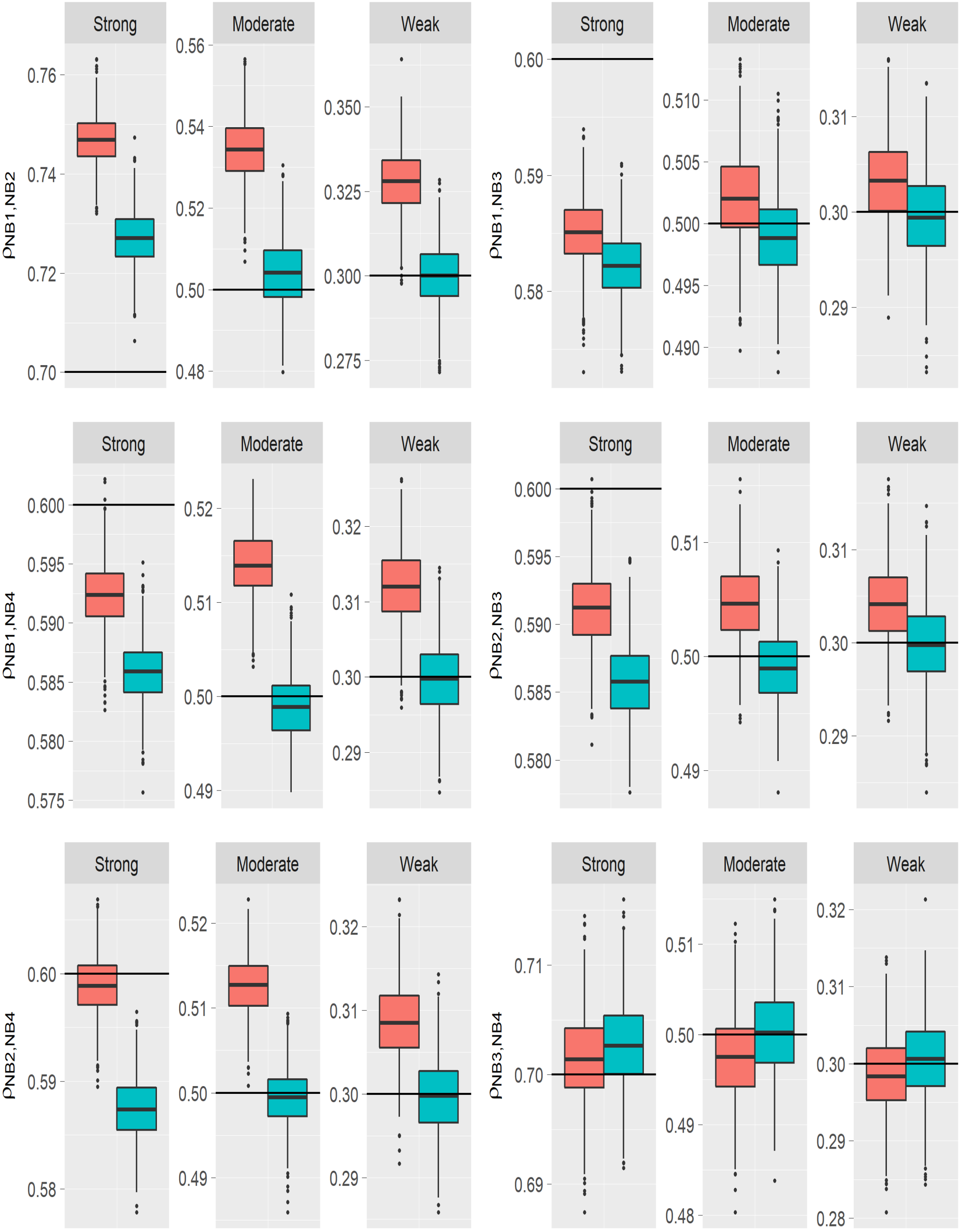


Figure 6: Boxplots of simulated correlations for NB1, NB2, NB3, and NB4 (scenario B). Method 1 is in red; method 2 is in green. The horizontal lines show the target values.

A. Barbiero and P. A. Ferrari. Simulation of correlated Poisson variables. *Applied Stochastic Models in Business and Industry*, 31:669–680, 2015a. URL https://doi.org/10.1002/asmb.2072. [p261, 266]

A. Barbiero and P. A. Ferrari. *GenOrd: Simulation of Discrete Random Variables with Given Correlation Matrix and Marginal Distributions*, 2015b. URL https://CRAN.R-project.org/package=GenOrd. R package version 1.4.0. [p251]

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 1.2-12. [p255]

M. Bhattacharjee, M. S. Rajeevan, and M. J. Sillanpää. Prediction of complex human diseases from pathway-focused candidate markers by joint estimation of marker effects: Case of chronic fatigue syndrome. *Human Genomics*, 9(1):8, 2015. URL https://doi.org/10.1186/s40246-015-0030-6. [p253]

P. Biecek and E. Szczurek. *Bgmm: Gaussian Mixture Modeling Algorithms and the Belief-Based Mixture Modeling*, 2017. URL https://CRAN.R-project.org/package=bgmm. R package version 1.8.3. [p250]

N. Bouguila, D. Ziou, and E. Monga. Practical Bayesian estimation of a finite beta mixture through Gibbs sampling and its applications. *Statistics and Computing*, 16:215–225, 2006. URL https://doi.org/10.1007/s11222-006-8451-7. [p255]

R. P. Browne, A. ElSherbiny, and P. D. McNicholas. *Mixture: Mixture Models for Clustering and Classification*, 2015. URL https://CRAN.R-project.org/package=mixture. R package version 1.4. [p250]

M. Comas-Cufí, J. A. Martín-Fernández, and G. Mateu-Figueras. *Mixpack: Tools to Work with Mixture Components*, 2017. URL https://CRAN.R-project.org/package=mixpack. R package version 0.3.6. [p251]

H. Dai and R. Charnigo. Compound hierarchical correlated beta mixture with an application to cluster mouse transcription factor DNA binding data. *Biostatistics (Oxford, England)*, 16(4): 641–654, 2015. URL http://doi.org/10.1093/biostatistics/kxv016. [p255]

X. Dai, T. Erkkilä, O. Yli-Harja, and H. Lähdesmäki. A joint finite mixture model for clustering genes from independent Gaussian and beta distributed data. *BMC Bioinformatics*, 10(1):165, 2009. URL https://doi.org/10.1186/1471-2105-10-165. [p255]

J. Davenport, J. Bezder, and R. Hathaway. Parameter estimation for finite mixture distributions. *Computers & Mathematics with Applications*, 15(10):819–828, 1988. [p252]

H. Demirtas and D. Hedeker. A practical way for computing approximate lower and upper correlation bounds. *The American Statistician*, 65(2):104–109, 2011. URL https://doi.org/10.1198/tast.2011.10090. [p261, 262]

H. Demirtas, D. Hedeker, and R. J. Mermelstein. Simulation of massive public health data by power polynomials. *Statistics in Medicine*, 31(27):3337–3346, 2012. URL https://doi.org/10.1002/sim.5362. [p261, 262]

R. C. Elston, J. M. Olson, and L. Palmer. *Biostatistical Genetics and Genetic Epidemiology*. John Wiley & Sons, Hoboken, New Jersey, 2002. [p253]

L. J. Emrich and M. R. Piedmonte. A method for generating high-dimensional multivariate binary variates. *The American Statistician*, 45:302–304, 1991. URL https://doi.org/10.1080/00031305.1991.10475828. [p261, 262]

B. S. Everitt. An introduction to finite mixture distributions. *Statistical Methods in Medical Research*, 5(2):107–127, 1996. URL https://doi.org/10.1177/096228029600500202. [p250, 252]

P. A. Ferrari and A. Barbiero. Simulating ordinal data. *Multivariate Behavioral Research*, 47(4): 566–589, 2012. URL https://doi.org/10.1080/00273171.2012.692630. [p261]

A. C. Fialkowski. *SimMultiCorrData: Simulation of Correlated Data with Multiple Variable Types*, 2017. URL https://CRAN.R-project.org/package=SimMultiCorrData. R package version 0.2.1. [p251]

A. C. Fialkowski and H. K. Tiwari. SimMultiCorrData: An R package for simulation of correlated non-normal or normal, binary, ordinal, poisson, and negative binomial variables. *Manuscript submitted for publication*, 2017. [p251, 261]

R. A. Fisher. Moments and product moments of sampling distributions. *Proceedings of the London Mathematical Society Series 2*, 30:199–238, 1929. [p264]

A. I. Fleishman. A method for simulating non-normal distributions. *Psychometrika*, 43:521–532, 1978. URL https://doi.org/10.1007/BF02293811. [p251, 253]

C. Fraley, A. E. Raftery, and L. Scrucca. *Mclust: Gaussian Mixture Modelling for Model-Based Clustering, Classification, and Density Estimation*, 2017. URL https://CRAN.R-project.org/package=mclust. R package version 5.4. [p250]

B. L. Fridley, D. Serie, G. Jenkins, K. White, W. Bamlet, J. D. Potter, and E. L. Goode. Bayesian mixture models for the incorporation of prior knowledge to inform genetic association studies. *Genetic Epidemiology*, 34(5):418–426, 2010. URL https://doi.org/10.1002/gepi.20494. [p253]

M. Fréchet. Les tableaux de corrélation et les programmes linéaires. *Revue de L'Institut International de Statistique / Review of the International Statistical Institute*, 25(1/3):23–40, 1957. URL https://doi.org/10.2307/1401672. [p261, 262]

R. Fu, D. K. Dey, and K. E. Holsinger. A beta-mixture model for assessing genetic population structure. *Biometrics*, 67(3):1073–1082, 2011. URL http://www.jstor.org/stable/41242556. [p255]

A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*, volume 195 of *Lecture Notes in Statistics*. Springer-Verlag, Heidelberg, 2009. URL https://doi.org/10.1007/978-3-642-01689-9. [p262]

A. Genz, F. Bretz, T. Miwa, X. Mi, and T. Hothorn. *Mvtnorm: Multivariate Normal and t Distributions*, 2017. URL https://CRAN.R-project.org/package=mvtnorm. R package version 1.0-6. [p262]

B. Gruen and F. Leisch. *Flexmix: Flexible Mixture Modeling*, 2017. URL https://CRAN.R-project.org/package=flexmix. R package version 2.3-14. [p251]

D. B. Hall. Zero-inflated Poisson and binomial regression with random effects: A case study. *Biometrics*, 56(4):1030–1039, 2000. URL https://doi.org/10.1111/j.0006-341X.2000.01030.x. [p250]

H. He, W. Tang, W. Wang, and P. Crits-Christoph. Structural zeroes and zero-inflated models. *Shanghai Archives of Psychiatry*, 26(4):236–242, 2014. URL https://doi.org/10.3969/j.issn.1002-0829.2014.04.008. [p260]

T. C. Headrick. Fast fifth-order polynomial transforms for generating univariate and multivariate non-normal distributions. *Computational Statistics & Data Analysis*, 40(4):685–711, 2002. URL https://doi.org/10.1016/S0167-9473(02)00072-5. [p251, 253, 254, 256]

T. C. Headrick and R. K. Kowalchuk. The power method transformation: Its probability density function, distribution function, and its further use for fitting data. *Journal of Statistical Computation and Simulation*, 77:229–249, 2007. URL https://doi.org/10.1080/10629360600605065. [p253, 258, 261]

T. C. Headrick and S. S. Sawilowsky. Simulating correlated non-normal distributions: Extending the Fleishman power method. *Psychometrika*, 64:25–35, 1999. URL https://doi.org/10.1007/BF02294317. [p256]

N. Higham. Computing the nearest correlation matrix - a problem from finance. *IMA Journal of Numerical Analysis*, 22(3):329–343, 2002. URL https://doi.org/10.1093/imanum/22.3.329. [p255, 266]

W. Hoeffding. Scale-invariant correlation theory. In N. I. Fisher and P. K. Sen, editors, *The Collected Works of Wassily Hoeffding*, Springer Series in Statistics (Perspectives in Statistics), pages 57–107. Springer-Verlag, New York, 1994. URL https://doi.org/10.1007/978-1-4612-0865-5_4. [p261, 263]

N. Ismail and H. Zamani. Estimation of claim count data using negative binomial, generalized Poisson, zero-inflated negative binomial and zero-inflated generalized Poisson regression models. *Casualty Actuarial Society E-Forum*, 41(20):1–28, 2013. [p250, 260]

Y. Ji, C. Wu, P. Liu, J. Wang, and K. R. Coombes. Applications of beta-mixture models in bioinformatics. *Bioinformatics*, 21(9):2118–2122, 2005. URL http://dx.doi.org/10.1093/bioinformatics/bti318. [p255]

M. Jochmann. *Zic: Bayesian Inference for Zero-Inflated Count Models*, 2017. URL https://CRAN.R-project.org/package=zic. R package version 0.9.1. [p251]

M. Kendall and A. Stuart. *The Advanced Theory of Statistics*. Macmillan, New York, 4th edition, 1977. [p253]

M. Kohl. *Distr: Object Oriented Implementation of Distributions*, 2017. URL https://CRAN.R-project.org/package=distr. R package version 2.6.2. [p251]

D. Lambert. Zero-inflated Poisson regression, with an application to defects in manufacturing. *Technometrics*, 34(1):1–14, 1992. [p250, 260]

F. Langrognet, R. Lebret, C. Poli, and S. Iovleff. *Rmixmod: Supervised, Unsupervised, Semi-Supervised Classification with MIXture MODelling (Interface of MIXMOD Software)*, 2016. URL https://CRAN.R-project.org/package=Rmixmod. R package version 2.1.1. [p251]

M. G. Larson and G. E. Dinse. A mixture model for the regression analysis of competing risks data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 34(3):201–211, 1985. URL http://www.jstor.org/stable/2347464. [p250]

B. Lau, S. R. Cole, and S. J. Gange. Competing risk regression models for epidemiologic data. *American Journal of Epidemiology*, 170(2):244–256, 2009. URL http://dx.doi.org/10.1093/aje/kwp107. [p250]

B. Lau, S. R. Cole, and S. J. Gange. Parametric mixture models to evaluate and summarize hazard ratios in the presence of competing risks with time-dependent hazards and delayed entry. *Statistics in Medicine*, 30(6):654–665, 2011. URL http://dx.doi.org/10.1002/sim.4123. [p250]

K. Laurila, B. Oster, C. L. Andersen, P. Lamy, T. Orntoft, O. Yli-Harja, and C. Wiuf. A beta-mixture model for dimensionality reduction, sample classification and analysis. *BMC Bioinformatics*, 12 (1):215, 2011. URL https://doi.org/10.1186/1471-2105-12-215. [p255]

R. R. J. Lewine. Sex differences in schizophrenia: Timing or subtypes? *Psychological Bulletin*, 90: 432–444, 1981. [p250]

S. Li, J. Chen, and P. Li. *MixtureInf: Inference for Finite Mixture Models*, 2016. URL https://CRAN.R-project.org/package=MixtureInf. R package version 1.1. [p251]

Z. Ma and A. Leijon. Bayesian estimation of beta mixture models with variational inference. *IEEE Trans Pattern Anal Mach Intell*, 33(11):2160–2173, 2011. URL https://doi.org/10.1109/TPAMI.2011.63. [p255]

P. MacDonald and with contributions from Juan Du. *Mixdist: Finite Mixture Distribution Models*, 2012. URL https://CRAN.R-project.org/package=mixdist. R package version 0.5-4. [p251]

G. J. McLachlan. Cluster analysis and related techniques in medical research. *Statistical Methods in Medical Research*, 1(1):27–48, 1992. URL https://doi.org/10.1177/096228029200100103. [p250]

A. Mohammadi. *Bmixture: Bayesian Estimation for Finite Mixture of Distributions*, 2017. URL https://CRAN.R-project.org/package=bmixture. R package version 0.5. [p251]

L. Mouselimis. *ClusterR: Gaussian Mixture Models, K-Means, Mini-Batch-Kmeans and K-Medoids Clustering*, 2017. URL https://CRAN.R-project.org/package=ClusterR. R package version 1.0.9. [p250]

M. Nagode. *Rebmix: Finite Mixture Modeling, Clustering & Classification*, 2017. URL https://CRAN.R-project.org/package=rebmix. R package version 2.9.3. [p251]

S. R. Newcomer, J. F. Steiner, and E. A. Bayliss. Identifying subgroups of complex patients with cluster analysis. *The American Journal of Managed Care*, 17(8):e324–32, 2011. [p250]

U. Olsson, F. Drasgow, and N. J. Dorans. The polyserial correlation coefficient. *Psychometrika*, 47 (3):337–347, 1982. URL https://doi.org/10.1007/BF02294164. [p261]

L. Pamulaparty, C. V. G. Rao, and M. S. Rao. Cluster analysis of medical research data using R. *Global Journal of Computer Science and Technology*, 16(1):1–6, 2016. [p250]

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/. [p251]

P. Schlattmann, J. Hoehne, and M. Verba. *CAMAN: Finite Mixture Models and Meta-Analysis Tools - Based on C.A.MAN*, 2016. URL https://CRAN.R-project.org/package=CAMAN. R package version 0.74. [p251]

N. J. Schork, D. B. Allison, and B. Thiel. Mixture distributions in human genetics research. *Statistical Methods in Medical Research*, 5:155–178, 1996. URL https://doi.org/10.1177/096228029600500204. [p252, 253]

P. C. Sham, C. J. MacLean, and K. S. Kendler. A typological model of schizophrenia based on age at onset, sex and familial morbidity. *Acta Psychiatrica Scandinavica*, 89(2):135–141, 1994. URL http://dx.doi.org/10.1111/j.1600-0447.1994.tb01501.x. [p250]

D. L. Solomon. Using RNA-seq data to detect differentially expressed genes. In S. Datta and D. Nettleton, editors, *Statistical Analysis of Next Generation Sequencing Data*, chapter 2, pages 25–49. Springer-Verlag, 2014. [p250]

C. Soneson and M. Delorenzi. A comparison of methods for differential expression analysis of RNA-seq data. *BMC Bioinformatics*, 14:91, 2013. URL https://doi.org/10.1186/1471-2105-14-91. [p250]

A. E. Teschendorff, F. Marabita, M. Lechner, T. Bartlett, J. Tegner, D. Gomez-Cabrero, and S. Beck. A beta-mixture quantile normalization method for correcting probe design bias in Illumina Infinium 450 k DNA methylation data. *Bioinformatics*, 29(2):189–196, 2013. URL https://doi.org/10.1093/bioinformatics/bts680. [p255]

M. Thrun, O. Hansen-Goos, R. Griese, C. Lippmann, F. Lerch, J. Lotsch, and A. Ultsch. *AdaptGauss: Gaussian Mixture Models (GMM)*, 2017. URL https://CRAN.R-project.org/package=AdaptGauss. R package version 1.3.3. [p250]

L. K. Vaughan, J. Divers, M. Padilla, D. T. Redden, H. K. Tiwari, D. Pomp, and D. B. Allison. The use of plasmodes as a supplement to simulations: A simple example evaluating individual admixture estimation methodologies. *Computational Statistics & Data Analysis*, 53(5):1755–1766, 2009. URL https://doi.org/10.1016/j.csda.2008.02.032. [p251]

Y. Wang. *Nspmix: Nonparametric and Semiparametric Mixture Estimation*, 2017. URL https://CRAN.R-project.org/package=nspmix. R package version 1.4-0. [p251]

J. Welham, G. Mclachlan, G. Davies, and J. McGrath. Heterogeneity in schizophrenia; mixture modelling of age-at-first-admission, gender and diagnosis. *Acta Psychiatrica Scandinavica*, 101(4):312–317, 2000. URL http://dx.doi.org/10.1034/j.1600-0447.2000.101004312.x. [p250]

H. Wickham and W. Chang. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2016. URL https://CRAN.R-project.org/package=ggplot2. R package version 2.2.1. [p258]

M. Winerip, G. Wallstrom, and J. LaBaer. *Bimixt: Estimates Mixture Models for Case-Control Data*, 2015. URL https://CRAN.R-project.org/package=bimixt. R package version 1.0. [p250]

I. Yahav and G. Shmueli. On generating multivariate Poisson data in management science applications. *Applied Stochastic Models in Business and Industry*, 28(1):91–102, 2012. URL https://doi.org/10.1002/asmb.901. [p260, 266]

T. W. Yee. *VGAM: Vector Generalized Linear and Additive Models*, 2017. URL https://CRAN.R-project.org/package=VGAM. R package version 1.0-4. [p251]

N. Yi. *BhGLM: Bayesian Hierarchical GLMs and Survival Models, with Application to Genetics and Epidemiology*, 2017. URL http://www.ssg.uab.edu/bhglm/. R package version 1.1.0. [p251]

D. Young, T. Benaglia, D. Chauveau, and D. Hunter. *Mixtools: Tools for Analyzing Finite Mixture Models*, 2017. URL https://CRAN.R-project.org/package=mixtools. R package version 1.1.0. [p251]

X. Zhang, H. Mallick, and N. Yi. Zero-inflated negative binomial regression for differential abundance testing in microbiome studies. *Journal of Bioinformatics and Genomics*, 2(2):1–9, 2016. URL https://doi.org/10.18454/jbg.2016.2.2.1. [p250, 260]

Y. Zhou, X. Wan, B. Zhang, and T. Tong. Classifying next-generation sequencing data using a zero-inflated poisson model. *Bioinformatics*, page btx768, 2017. URL https://doi.org/10.1093/bioinformatics/btx768. [p250]

Allison Fialkowski
Department of Biostatistics
School of Public Health
University of Alabama at Birmingham
RPHB 327
1720 2nd Ave S
Birmingham, AL 35294-0022
allijazz@uab.edu

Hemant Tiwari
Department of Biostatistics
School of Public Health
University of Alabama at Birmingham
RPHB 420C
1720 2nd Ave S
Birmingham, AL 35294-0022
htiwari@uab.edu

Appendix

Derivation of expected cumulants of continuous mixture variables

Suppose the goal is to simulate a continuous mixture variable Y with PDF $h_Y(y)$ that contains two component distributions Y_a and Y_b with mixing parameters π_a and π_b:

$$h_Y(y) = \pi_a f_{Y_a}(y) + \pi_b g_{Y_b}(y), \ y \in \boldsymbol{Y}, \ \pi_a \in (0, \ 1), \ \pi_b \in (0, \ 1), \ \pi_a + \pi_b = 1. \tag{N.11.1}$$

Here,

$$Y_a = \sigma_a Z'_a + \mu_a, \ Y_a \sim f_{Y_a}(y), \ y \in \boldsymbol{Y_a} \ \text{ and } \ Y_b = \sigma_b Z'_b + \mu_b, \ Y_b \sim g_{Y_b}(y), \ y \in \boldsymbol{Y_b} \tag{N.11.2}$$

so that Y_a and Y_b have expected values μ_a and μ_b and variances σ_a^2 and σ_b^2. Assume the variables Z'_a and Z'_b are generated with zero mean and unit variance using Headrick's fifth-order PMT given the specified values for skew $\left(\gamma'_{1_a}, \ \gamma'_{1_b}\right)$, skurtosis $\left(\gamma'_{2_a}, \ \gamma'_{2_b}\right)$, and standardized fifth $\left(\gamma'_{3_a}, \ \gamma'_{3_b}\right)$ and sixth $\left(\gamma'_{4_a}, \ \gamma'_{4_b}\right)$ cumulants. The r^{th} expected value of Y can be expressed as:

$$\begin{aligned}\mathbb{E}\left[Y^r\right] &= \int y^r h_Y(y)\,dy = \pi_a \int y^r f_{Y_a}(y)\,dy + \pi_b \int y^r g_{Y_b}(y)\,dy \\ &= \pi_a\,\mathbb{E}\left[Y_a^r\right] + \pi_b\,\mathbb{E}\left[Y_b^r\right].\end{aligned} \tag{N.11.3}$$

Equation N.11.3 can be used to derive expressions for the mean, variance, skew, skurtosis, and standardized fifth and sixth cumulants of Y in terms of the r^{th} expected values of Y_a and Y_b.

1. Mean: Using $r = 1$ in Equation N.11.3 yields μ:

$$\begin{aligned}\mathbb{E}[Y] &= \pi_a\,\mathbb{E}[Y_a] + \pi_b\,\mathbb{E}[Y_b] = \pi_a\,\mathbb{E}\left[\sigma_a Z'_a + \mu_a\right] + \pi_b\,\mathbb{E}\left[\sigma_b Z'_b + \mu_b\right] \\ &= \pi_a\left(\sigma_a\,\mathbb{E}\left[Z'_a\right] + \mu_a\right) + \pi_b\left(\sigma_b\,\mathbb{E}\left[Z'_b\right] + \mu_b\right).\end{aligned} \tag{N.11.4}$$

   Since $\mathbb{E}\left[Z'_a\right] = \mathbb{E}\left[Z'_b\right] = 0$, this becomes:

$$\mathbb{E}[Y] = \pi_a\mu_a + \pi_b\mu_b. \tag{N.11.5}$$

2. Variance: The variance of Y can be expressed by the relation $\text{Var}[Y] = \mathbb{E}\left[Y^2\right] - (\mathbb{E}[Y])^2$.

Using $r = 2$ in Equation N.11.3 yields μ_2:

$$\begin{aligned}\mathbb{E}\left[Y^2\right] &= \pi_a\,\mathbb{E}\left[Y_a^2\right] + \pi_b\,\mathbb{E}\left[Y_b^2\right] = \pi_a\,\mathbb{E}\left[\left(\sigma_a Z_a' + \mu_a\right)^2\right] + \pi_b\,\mathbb{E}\left[\left(\sigma_b Z_b' + \mu_b\right)^2\right] \\ &= \pi_a\,\mathbb{E}\left[\sigma_a^2 {Z_a'}^2 + 2\mu_a\sigma_a Z_a' + \mu_a^2\right] + \pi_b\,\mathbb{E}\left[\sigma_b^2 {Z_b'}^2 + 2\mu_b\sigma_b Z_b' + \mu_b^2\right] \\ &= \pi_a\left(\sigma_a^2\,\mathbb{E}\left[{Z_a'}^2\right] + 2\mu_a\sigma_a\,\mathbb{E}\left[Z_a'\right] + \mu_a^2\right) \\ &\quad + \pi_b\left(\sigma_b^2\,\mathbb{E}\left[{Z_b'}^2\right] + 2\mu_b\sigma_b\,\mathbb{E}\left[Z_b'\right] + \mu_b^2\right). \end{aligned} \tag{N.11.6}$$

Applying the variance relation to Z_a' and Z_b' gives:

$$\begin{aligned}\mathbb{E}\left[{Z_a'}^2\right] &= \mathrm{Var}\left[Z_a'\right] + \left(\mathbb{E}\left[Z_a'\right]\right)^2 \\ \mathbb{E}\left[{Z_b'}^2\right] &= \mathrm{Var}\left[Z_b'\right] + \left(\mathbb{E}\left[Z_b'\right]\right)^2. \end{aligned} \tag{N.11.7}$$

Since $\mathbb{E}\left[Z_a'\right] = \mathbb{E}\left[Z_b'\right] = 0$ and $\mathrm{Var}\left[Z_a'\right] = \mathrm{Var}\left[Z_b'\right] = 1$, $\mathbb{E}\left[{Z_a'}^2\right]$ and $\mathbb{E}\left[{Z_b'}^2\right]$ both equal 1. Therefore, Equation N.11.6 simplifies to:

$$\mathbb{E}\left[Y^2\right] = \pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right), \tag{N.11.8}$$

and the variance of Y is given by:

$$\mathrm{Var}\left[Y\right] = \pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2. \tag{N.11.9}$$

3. Skew: Using $r = 3$ in Equation N.11.3 yields μ_3:

$$\begin{aligned}\mathbb{E}\left[Y^3\right] &= \pi_a\,\mathbb{E}\left[Y_a^3\right] + \pi_b\,\mathbb{E}\left[Y_b^3\right] = \pi_a\,\mathbb{E}\left[\left(\sigma_a Z_a' + \mu_a\right)^3\right] + \pi_b\,\mathbb{E}\left[\left(\sigma_b Z_b' + \mu_b\right)^3\right] \\ &= \pi_a\,\mathbb{E}\left[\sigma_a^3 {Z_a'}^3 + 3\sigma_a^2\mu_a {Z_a'}^2 + 3\sigma_a\mu_a^2 Z_a' + \mu_a^3\right] \\ &\quad + \pi_b\,\mathbb{E}\left[\sigma_b^3 {Z_b'}^3 + 3\sigma_b^2\mu_b {Z_b'}^2 + 3\sigma_b\mu_b^2 Z_b' + \mu_b^3\right] \\ &= \pi_a\left(\sigma_a^3\,\mathbb{E}\left[{Z_a'}^3\right] + 3\sigma_a^2\mu_a\,\mathbb{E}\left[{Z_a'}^2\right] + 3\sigma_a\mu_a^2\,\mathbb{E}\left[Z_a'\right] + \mu_a^3\right) \\ &\quad + \pi_b\left(\sigma_b^3\,\mathbb{E}\left[{Z_b'}^3\right] + 3\sigma_b^2\mu_b\,\mathbb{E}\left[{Z_b'}^2\right] + 3\sigma_b\mu_b^2\,\mathbb{E}\left[Z_b'\right] + \mu_b^3\right). \end{aligned} \tag{N.11.10}$$

Then $\mathbb{E}\left[{Z_a'}^3\right] = \mu_{3_a}'$ and $\mathbb{E}\left[{Z_b'}^3\right] = \mu_{3_b}'$ are given by:

$$\begin{aligned}\mathbb{E}\left[{Z_a'}^3\right] &= \left(\mathrm{Var}\left[Z_a'\right]\right)^{3/2}\gamma_{1_a}' = \gamma_{1_a}' \\ \mathbb{E}\left[{Z_b'}^3\right] &= \left(\mathrm{Var}\left[Z_b'\right]\right)^{3/2}\gamma_{1_b}' = \gamma_{1_b}'. \end{aligned} \tag{N.11.11}$$

Combining these with $\mathbb{E}\left[Z_a'\right] = \mathbb{E}\left[Z_b'\right] = 0$ and $\mathbb{E}\left[{Z_a'}^2\right] = \mathbb{E}\left[{Z_b'}^2\right] = 1$, Equation N.11.10 simplifies to:

$$\mathbb{E}\left[Y^3\right] = \pi_a\left(\sigma_a^3\gamma_{1_a}' + 3\sigma_a^2\mu_a + \mu_a^3\right) + \pi_b\left(\sigma_b^3\gamma_{1_b}' + 3\sigma_b^2\mu_b + \mu_b^3\right). \tag{N.11.12}$$

From Equation N.3.6, the skew of Y is given by:

$$\gamma_1 = \frac{\pi_a\left(\sigma_a^3\gamma_{1_a}' + 3\sigma_a^2\mu_a + \mu_a^3\right) + \pi_b\left(\sigma_b^3\gamma_{1_b}' + 3\sigma_b^2\mu_b + \mu_b^3\right)}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^{3/2}}. \tag{N.11.13}$$

4. Skurtosis: Using $r = 4$ in Equation N.11.3 yields μ_4:

$$\begin{aligned}
\mathbb{E}\left[Y^4\right] &= \pi_a\,\mathbb{E}\left[Y_a^4\right] + \pi_b\,\mathbb{E}\left[Y_b^4\right] = \pi_a\,\mathbb{E}\left[\left(\sigma_a Z'_a + \mu_a\right)^4\right] + \pi_b\,\mathbb{E}\left[\left(\sigma_b Z'_b + \mu_b\right)^4\right] \\
&= \pi_a\,\mathbb{E}\left[\sigma_a^4 {Z'_a}^4 + 4\sigma_a^3\mu_a {Z'_a}^3 + 6\sigma_a^2\mu_a^2 {Z'_a}^2 + 4\sigma_a\mu_a^3 Z'_a + \mu_a^4\right] \\
&\quad + \pi_b\,\mathbb{E}\left[\sigma_b^4 {Z'_b}^4 + 4\sigma_b^3\mu_b {Z'_b}^3 + 6\sigma_b^2\mu_b^2 {Z'_b}^2 + 4\sigma_b\mu_b^3 Z'_b + \mu_b^4\right] \\
&= \pi_a\Big(\sigma_a^4\,\mathbb{E}\left[{Z'_a}^4\right] + 4\sigma_a^3\mu_a\,\mathbb{E}\left[{Z'_a}^3\right] + 6\sigma_a^2\mu_a^2\,\mathbb{E}\left[{Z'_a}^2\right] + 4\sigma_a\mu_a^3\,\mathbb{E}\left[Z'_a\right] + \mu_a^4\Big) \\
&\quad + \pi_b\Big(\sigma_b^4\,\mathbb{E}\left[{Z'_b}^4\right] + 4\sigma_b^3\mu_b\,\mathbb{E}\left[{Z'_b}^3\right] + 6\sigma_b^2\mu_b^2\,\mathbb{E}\left[{Z'_b}^2\right] + 4\sigma_b\mu_b^3\,\mathbb{E}\left[Z'_b\right] + \mu_b^4\Big)
\end{aligned} \tag{N.11.14}$$

Then $\mathbb{E}\left[{Z'_a}^4\right] = \mu'_{4_a}$ and $\mathbb{E}\left[{Z'_b}^4\right] = \mu'_{4_b}$ are given by:

$$\begin{aligned}
\mathbb{E}\left[{Z'_a}^4\right] &= \left(\mathrm{Var}\left[Z'_a\right]\right)^2\left(\gamma'_{2_a} + 3\right) = \gamma'_{2_a} + 3 \\
\mathbb{E}\left[{Z'_b}^4\right] &= \left(\mathrm{Var}\left[Z'_b\right]\right)^2\left(\gamma'_{2_b} + 3\right) = \gamma'_{2_b} + 3.
\end{aligned} \tag{N.11.15}$$

Since $\mathbb{E}\left[Z'_a\right] = \mathbb{E}\left[Z'_b\right] = 0$ and $\mathbb{E}\left[{Z'_a}^2\right] = \mathbb{E}\left[{Z'_b}^2\right] = 1$, Equation N.11.14 simplifies to:

$$\begin{aligned}
\mathbb{E}\left[Y^4\right] = \pi_a &\left[\sigma_a^4\left(\gamma'_{2_a} + 3\right) + 4\sigma_a^3\mu_a\gamma'_{1_a} + 6\sigma_a^2\mu_a^2 + \mu_a^4\right] \\
&+ \pi_b\left[\sigma_b^4\left(\gamma'_{2_b} + 3\right) + 4\sigma_b^3\mu_b\gamma'_{1_b} + 6\sigma_b^2\mu_b^2 + \mu_b^4\right].
\end{aligned} \tag{N.11.16}$$

From Equation N.3.7, the skurtosis of Y is given by:

$$\begin{aligned}
\gamma_2 = &\frac{\pi_a\left[\sigma_a^4\left(\gamma'_{2_a} + 3\right) + 4\sigma_a^3\mu_a\gamma'_{1_a} + 6\sigma_a^2\mu_a^2 + \mu_a^4\right]}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^2} \\
&+ \frac{\pi_b\left[\sigma_b^4\left(\gamma'_{2_b} + 3\right) + 4\sigma_b^3\mu_b\gamma'_{1_b} + 6\sigma_b^2\mu_b^2 + \mu_b^4\right]}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^2}.
\end{aligned} \tag{N.11.17}$$

5. Standardized fifth cumulant: Using $r = 5$ in Equation N.11.3 yields μ_5:

$$\begin{aligned}
\mathbb{E}\left[Y^5\right] &= \pi_a\,\mathbb{E}\left[Y_a^5\right] + \pi_b\,\mathbb{E}\left[Y_b^5\right] = \pi_a\,\mathbb{E}\left[\left(\sigma_a Z'_a + \mu_a\right)^5\right] + \pi_b\,\mathbb{E}\left[\left(\sigma_b Z'_b + \mu_b\right)^5\right] \\
&= \pi_a\,\mathbb{E}\left[\sigma_a^5 {Z'_a}^5 + 5\sigma_a^4\mu_a {Z'_a}^4 + 10\sigma_a^3\mu_a^2 {Z'_a}^3 + 10\sigma_a^2\mu_a^3 {Z'_a}^2 + 5\sigma_a\mu_a^4 Z'_a + \mu_a^5\right] \\
&\quad + \pi_b\,\mathbb{E}\left[\sigma_b^5 {Z'_b}^5 + 5\sigma_b^4\mu_b {Z'_b}^4 + 10\sigma_b^3\mu_b^2 {Z'_b}^3 + 10\sigma_b^2\mu_b^3 {Z'_b}^2 + 5\sigma_b\mu_b^4 Z'_b + \mu_b^5\right] \\
&= \pi_a\Big(\sigma_a^5\,\mathbb{E}\left[{Z'_a}^5\right] + 5\sigma_a^4\mu_a\,\mathbb{E}\left[{Z'_a}^4\right] + 10\sigma_a^3\mu_a^2\,\mathbb{E}\left[{Z'_a}^3\right] \\
&\qquad + 10\sigma_a^2\mu_a^3\,\mathbb{E}\left[{Z'_a}^2\right] + 5\sigma_a\mu_a^4\,\mathbb{E}\left[Z'_a\right] + \mu_a^5\Big) \\
&\quad + \pi_b\Big(\sigma_b^5\,\mathbb{E}\left[{Z'_b}^5\right] + 5\sigma_b^4\mu_b\,\mathbb{E}\left[{Z'_b}^4\right] + 10\sigma_b^3\mu_b^2\,\mathbb{E}\left[{Z'_b}^3\right] \\
&\qquad + 10\sigma_b^2\mu_b^3\,\mathbb{E}\left[{Z'_b}^2\right] + 5\sigma_b\mu_b^4\,\mathbb{E}\left[Z'_b\right] + \mu_b^5\Big).
\end{aligned} \tag{N.11.18}$$

Then $\mathbb{E}\left[{Z'_a}^5\right] = \mu'_{5_a}$ and $\mathbb{E}\left[{Z'_b}^5\right] = \mu'_{5_b}$ are given by:

$$\begin{aligned}
\mathbb{E}\left[{Z'_a}^5\right] &= \left(\mathrm{Var}\left[Z'_a\right]\right)^{5/2}\left(\gamma'_{3_a} + 10\gamma'_{1_a}\right) = \gamma'_{3_a} + 10\gamma'_{1_a} \\
\mathbb{E}\left[{Z'_b}^5\right] &= \left(\mathrm{Var}\left[Z'_b\right]\right)^{5/2}\left(\gamma'_{3_b} + 10\gamma'_{1_b}\right) = \gamma'_{3_b} + 10\gamma'_{1_b}.
\end{aligned} \tag{N.11.19}$$

Since $\mathbb{E}\left[Z'_a\right] = \mathbb{E}\left[Z'_b\right] = 0$ and $\mathbb{E}\left[{Z'_a}^2\right] = \mathbb{E}\left[{Z'_b}^2\right] = 1$, Equation N.11.18 simplifies to:

$$\begin{aligned}\mathbb{E}\left[Y^5\right] = \pi_a &\left[\sigma_a^5\left(\gamma'_{3_a} + 10\gamma'_{1_a}\right) + 5\sigma_a^4\mu_a\left(\gamma'_{2_a} + 3\right) + 10\sigma_a^3\mu_a^2\gamma'_{1_a} + 10\sigma_a^2\mu_a^3 + \mu_a^5\right] \\ &+ \pi_b\left[\sigma_b^5\left(\gamma'_{3_b} + 10\gamma'_{1_b}\right) + 5\sigma_b^4\mu_b\left(\gamma'_{2_b} + 3\right) + 10\sigma_b^3\mu_b^2\gamma'_{1_b} + 10\sigma_b^2\mu_b^3 + \mu_b^5\right].\end{aligned} \tag{N.11.20}$$

From Equation N.3.8, the standardized fifth cumulant of Y is given by:

$$\begin{aligned}\gamma_3 = &\frac{\pi_a\left[\sigma_a^5\left(\gamma'_{3_a} + 10\gamma'_{1_a}\right) + 5\sigma_a^4\mu_a\left(\gamma'_{2_a} + 3\right) + 10\sigma_a^3\mu_a^2\gamma'_{1_a} + 10\sigma_a^2\mu_a^3 + \mu_a^5\right]}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^{5/2}} \\ &+ \frac{\pi_b\left[\sigma_b^5\left(\gamma'_{3_b} + 10\gamma'_{1_b}\right) + 5\sigma_b^4\mu_b\left(\gamma'_{2_b} + 3\right) + 10\sigma_b^3\mu_b^2\gamma'_{1_b} + 10\sigma_b^2\mu_b^3 + \mu_b^5\right]}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^{5/2}} - 10\gamma_1.\end{aligned} \tag{N.11.21}$$

6. Standardized sixth cumulant: Using $r = 6$ in Equation N.11.3 yields μ_6:

$$\begin{aligned}\mathbb{E}\left[Y^6\right] &= \pi_a\,\mathbb{E}\left[Y_a^6\right] + \pi_b\,\mathbb{E}\left[Y_b^6\right] = \pi_a\,\mathbb{E}\left[\left(\sigma_a Z'_a + \mu_a\right)^6\right] + \pi_b\,\mathbb{E}\left[\left(\sigma_b Z'_b + \mu_b\right)^6\right] \\ &= \pi_a\,\mathbb{E}\left[\sigma_a^6{Z'_a}^6 + 6\sigma_a^5\mu_a{Z'_a}^5 + 15\sigma_a^4\mu_a^2{Z'_a}^4 + 20\sigma_a^3\mu_a^3{Z'_a}^3\right. \\ &\qquad\qquad \left. + 15\sigma_a^2\mu_a^4{Z'_a}^2 + 6\sigma_a\mu_a^5 Z'_a + \mu_a^6\right] \\ &\qquad + \pi_b\,\mathbb{E}\left[\sigma_b^6{Z'_b}^6 + 6\sigma_b^5\mu_b{Z'_b}^5 + 15\sigma_b^4\mu_b^2{Z'_b}^4 + 20\sigma_b^3\mu_b^3{Z'_b}^3\right. \\ &\qquad\qquad \left. + 15\sigma_b^2\mu_b^4{Z'_b}^2 + 6\sigma_b\mu_b^5 Z'_b + \mu_b^6\right]\end{aligned}$$

$$\begin{aligned}&= \pi_a\Big(\sigma_a^6\,\mathbb{E}\left[{Z'_a}^6\right] + 6\sigma_a^5\mu_a\,\mathbb{E}\left[{Z'_a}^5\right] + 15\sigma_a^4\mu_a^2\,\mathbb{E}\left[{Z'_a}^4\right] + 20\sigma_a^3\mu_a^3\,\mathbb{E}\left[{Z'_a}^3\right] \\ &\qquad + 15\sigma_a^2\mu_a^4\,\mathbb{E}\left[{Z'_a}^2\right] + 6\sigma_a\mu_a^5\,\mathbb{E}\left[Z'_a\right] + \mu_a^6\Big) \\ &\quad + \pi_b\Big(\sigma_b^6\,\mathbb{E}\left[{Z'_b}^6\right] + 6\sigma_b^5\mu_b\,\mathbb{E}\left[{Z'_b}^5\right] + 15\sigma_b^4\mu_b^2\,\mathbb{E}\left[{Z'_b}^4\right] + 20\sigma_b^3\mu_b^3\,\mathbb{E}\left[{Z'_b}^3\right] \\ &\qquad + 15\sigma_b^2\mu_b^4\,\mathbb{E}\left[{Z'_b}^2\right] + 6\sigma_b\mu_b^5\,\mathbb{E}\left[Z'_b\right] + \mu_b^6\Big).\end{aligned} \tag{N.11.22}$$

Then $\mathbb{E}\left[{Z'_a}^6\right] = \mu'_{6_a}$ and $\mathbb{E}\left[{Z'_b}^6\right] = \mu'_{6_b}$ are given by:

$$\begin{aligned}\mathbb{E}\left[{Z'_a}^6\right] &= \left(\mathrm{Var}\left[Z'_a\right]\right)^3\left(\gamma'_{4_a} + 15\gamma'_{2_a} + 10{\gamma'_{1_a}}^2 + 15\right) = \gamma'_{4_a} + 15\gamma'_{2_a} + 10{\gamma'_{1_a}}^2 + 15 \\ \mathbb{E}\left[{Z'_b}^6\right] &= \left(\mathrm{Var}\left[Z'_b\right]\right)^3\left(\gamma'_{4_b} + 15\gamma'_{2_b} + 10{\gamma'_{1_b}}^2 + 15\right) = \gamma'_{4_b} + 15\gamma'_{2_b} + 10{\gamma'_{1_b}}^2 + 15.\end{aligned} \tag{N.11.23}$$

Since $\mathbb{E}\left[Z'_a\right] = \mathbb{E}\left[Z'_b\right] = 0$ and $\mathbb{E}\left[{Z'_a}^2\right] = \mathbb{E}\left[{Z'_b}^2\right] = 1$, Equation N.11.22 simplifies to:

$$\begin{aligned}\mathbb{E}\left[Y^6\right] = \pi_a\Big[&\sigma_a^6\left(\gamma'_{4_a} + 15\gamma'_{2_a} + 10{\gamma'_{1_a}}^2 + 15\right) + 6\sigma_a^5\mu_a\left(\gamma'_{3_a} + 10\gamma'_{1_a}\right) \\ &+ 15\sigma_a^4\mu_a^2\left(\gamma'_{2_a} + 3\right) + 20\sigma_a^3\mu_a^3\gamma'_{1_a} + 15\sigma_a^2\mu_a^4 + \mu_a^6\Big] \\ + \pi_b\Big[&\sigma_b^6\left(\gamma'_{4_b} + 15\gamma'_{2_b} + 10{\gamma'_{1_b}}^2 + 15\right) + 6\sigma_b^5\mu_b\left(\gamma'_{3_b} + 10\gamma'_{1_b}\right) \\ &+ 15\sigma_b^4\mu_b^2\left(\gamma'_{2_b} + 3\right) + 20\sigma_b^3\mu_b^3\gamma'_{1_b} + 15\sigma_b^2\mu_b^4 + \mu_b^6\Big].\end{aligned} \tag{N.11.24}$$

From Equation N.3.9, the standardized sixth cumulant of Y is given by:

$$\begin{aligned}\gamma_4 = {} & \frac{\pi_a\Big[\sigma_a^6\left({\gamma'_{4_a}} + 15{\gamma'_{2_a}} + 10{\gamma'_{1_a}}^2 + 15\right) + 6\sigma_a^5\mu_a\left({\gamma'_{3_a}} + 10{\gamma'_{1_a}}\right)}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^3} \\ & \qquad \frac{+15\sigma_a^4\mu_a^2\left({\gamma'_{2_a}} + 3\right) + 20\sigma_a^3\mu_a^3{\gamma'_{1_a}} + 15\sigma_a^2\mu_a^4 + \mu_a^6\Big]}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^3} \\ & + \frac{\pi_b\Big[\sigma_b^6\left({\gamma'_{4_b}} + 15{\gamma'_{2_b}} + 10{\gamma'_{1_b}}^2 + 15\right) + 6\sigma_b^5\mu_b\left({\gamma'_{3_b}} + 10{\gamma'_{1_b}}\right)}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^3} \\ & \qquad \frac{+15\sigma_b^4\mu_b^2\left({\gamma'_{2_b}} + 3\right) + 20\sigma_b^3\mu_b^3{\gamma'_{1_b}} + 15\sigma_b^2\mu_b^4 + \mu_b^6\Big]}{\left(\pi_a\left(\sigma_a^2 + \mu_a^2\right) + \pi_b\left(\sigma_b^2 + \mu_b^2\right) - \left[\pi_a\mu_a + \pi_b\mu_b\right]^2\right)^3} \\ & - 15\gamma_2 - 10{\gamma_1}^2 - 15.\end{aligned} \tag{N.11.25}$$

Results from examples comparing correlation methods 1 and 2

	Scenario	
Correlation Type	A: Poisson and NB	B: NB
Strong	6	9
Moderate	7	10
Weak	8	11

Table 5: Table numbers for matrices of correlation errors.

N2	-0.078 (-0.08, -0.076)	0.153 (0.153, 0.153)	0	0.153 (0.152, 0.153)	-0.164 (-0.164, -0.164)
N3	-0.078 (-0.081, -0.076)	0.153 (0.153, 0.153)	0.153 (0.153, 0.153)	0	-0.164 (-0.164, -0.164)
B1	-0.034 (-0.036, -0.031)	-0.164 (-0.164, -0.164)	-0.164 (-0.164, -0.164)	-0.164 (-0.164, -0.164)	0
B2	-0.035 (-0.037, -0.033)	-0.165 (-0.166, -0.165)	-0.166 (-0.166, -0.165)	-0.166 (-0.166, -0.165)	0.155 (0.154, 0.156)
P1	-0.033 (-0.036, -0.03)	-0.157 (-0.16, -0.153)	-0.156 (-0.159, -0.153)	-0.156 (-0.159, -0.153)	-0.123 (-0.125, -0.12)
P2	-0.018 (-0.02, -0.015)	-0.133 (-0.135, -0.13)	-0.133 (-0.135, -0.13)	-0.133 (-0.135, -0.13)	-0.097 (-0.1, -0.095)
NB1	-0.05 (-0.053, -0.047)	-0.168 (-0.172, -0.165)	-0.168 (-0.171, -0.165)	-0.168 (-0.171, -0.165)	-0.137 (-0.14, -0.134)
NB2	-0.028 (-0.031, -0.025)	-0.156 (-0.16, -0.153)	-0.156 (-0.159, -0.153)	-0.156 (-0.159, -0.153)	-0.125 (-0.128, -0.122)

	B2	P1	P2	NB1	NB2
O1	-0.038 (-0.04, -0.035)	-0.023 (-0.025, -0.02)	0.003 (0, 0.005)	-0.053 (-0.056, -0.05)	-0.043 (-0.046, -0.04)
N1	-0.166 (-0.167, -0.165)	-0.156 (-0.159, -0.153)	-0.128 (-0.131, -0.126)	-0.166 (-0.169, -0.163)	-0.153 (-0.156, -0.15)
N2	-0.166 (-0.167, -0.165)	-0.156 (-0.158, -0.153)	-0.128 (-0.131, -0.126)	-0.166 (-0.17, -0.163)	-0.153 (-0.156, -0.15)
N3	-0.166 (-0.167, -0.165)	-0.156 (-0.159, -0.153)	-0.128 (-0.131, -0.126)	-0.166 (-0.17, -0.163)	-0.153 (-0.156, -0.15)
B1	0.154 (0.153, 0.155)	-0.123 (-0.126, -0.12)	-0.093 (-0.096, -0.091)	-0.135 (-0.138, -0.132)	-0.121 (-0.124, -0.118)
B2	0	-0.156 (-0.159, -0.154)	-0.121 (-0.123, -0.118)	-0.174 (-0.177, -0.171)	-0.157 (-0.16, -0.155)
P1	-0.156 (-0.159, -0.153)	0	0.029 (0.025, 0.031)	0.013 (0.009, 0.016)	0.035 (0.032, 0.038)
P2	-0.134 (-0.136, -0.132)	0.013 (0.01, 0.016)	0	0.036 (0.033, 0.039)	0.014 (0.01, 0.017)

N2	-0.019 (-0.022, -0.016)	0.051 (0.051, 0.051)	0	0.049 (0.049, 0.05)	-0.035 (-0.035, -0.035)
N3	-0.019 (-0.022, -0.017)	0.051 (0.051, 0.051)	0.051 (0.051, 0.051)	0	-0.035 (-0.035, -0.035)
B1	-0.001 (-0.003, 0.002)	-0.034 (-0.034, -0.034)	-0.034 (-0.034, -0.033)	-0.034 (-0.034, -0.033)	0
B2	-0.001 (-0.003, 0.002)	-0.034 (-0.035, -0.033)	-0.034 (-0.035, -0.033)	-0.034 (-0.035, -0.033)	0.016 (0.015, 0.017)
P1	-0.002 (-0.005, 0.001)	-0.041 (-0.045, -0.038)	-0.041 (-0.044, -0.038)	-0.041 (-0.044, -0.038)	-0.009 (-0.012, -0.006)
P2	-0.001 (-0.005, 0.002)	-0.034 (-0.037, -0.032)	-0.034 (-0.037, -0.032)	-0.034 (-0.037, -0.032)	-0.006 (-0.009, -0.003)
NB1	-0.004 (-0.007, 0)	-0.044 (-0.048, -0.041)	-0.045 (-0.048, -0.041)	-0.044 (-0.048, -0.041)	-0.011 (-0.014, -0.008)
NB2	-0.003 (-0.006, 0)	-0.041 (-0.044, -0.037)	-0.041 (-0.044, -0.037)	-0.041 (-0.044, -0.038)	-0.01 (-0.012, -0.007)

	B2	P1	P2	NB1	NB2
O1	0.001 (-0.002, 0.003)	0 (-0.004, 0.004)	0.005 (0.001, 0.008)	-0.009 (-0.013, -0.006)	-0.007 (-0.011, -0.004)
N1	-0.035 (-0.036, -0.034)	-0.038 (-0.041, -0.035)	-0.031 (-0.034, -0.028)	-0.04 (-0.043, -0.037)	-0.037 (-0.04, -0.034)
N2	-0.035 (-0.036, -0.034)	-0.038 (-0.041, -0.035)	-0.031 (-0.034, -0.029)	-0.04 (-0.044, -0.037)	-0.037 (-0.04, -0.033)
N3	-0.035 (-0.036, -0.034)	-0.038 (-0.041, -0.035)	-0.031 (-0.034, -0.028)	-0.041 (-0.044, -0.037)	-0.037 (-0.04, -0.034)
B1	0.013 (0.012, 0.014)	-0.005 (-0.008, -0.002)	-0.003 (-0.006, 0)	-0.006 (-0.01, -0.003)	-0.005 (-0.008, -0.002)
B2	0	-0.027 (-0.029, -0.024)	-0.019 (-0.022, -0.017)	-0.033 (-0.035, -0.03)	-0.029 (-0.031, -0.027)
P1	-0.03 (-0.033, -0.028)	0	0.02 (0.016, 0.025)	0.014 (0.008, 0.019)	0.028 (0.023, 0.033)
P2	-0.022 (-0.024, -0.019)	0.004 (0, 0.008)	0	0.033 (0.029, 0.037)	0.013 (0.008, 0.017)

N2	0 (-0.003, 0.003)	0 (0, 0)	0	0 (0, 0)	0 (0, 0)
N3	0 (-0.003, 0.003)	0 (0, 0)	0 (0, 0)	0	0 (0, 0)
B1	0 (-0.003, 0.003)	0 (0, 0)	0 (0, 0)	0 (0, 0)	0
B2	0 (-0.003, 0.004)	0 (-0.001, 0.001)	0 (-0.001, 0.001)	0 (-0.001, 0.001)	0 (-0.001, 0.001)
P1	0 (-0.004, 0.004)	0 (-0.004, 0.004)	0 (-0.004, 0.003)	0 (-0.004, 0.004)	-0.001 (-0.004, 0.002)
P2	0 (-0.004, 0.004)	0 (-0.003, 0.003)	0 (-0.003, 0.003)	0 (-0.003, 0.003)	0 (-0.003, 0.003)
NB1	-0.001 (-0.005, 0.003)	0 (-0.004, 0.004)	0 (-0.004, 0.004)	0 (-0.004, 0.004)	-0.001 (-0.005, 0.002)
NB2	-0.001 (-0.005, 0.003)	0 (-0.004, 0.003)	0 (-0.003, 0.003)	0 (-0.004, 0.003)	-0.002 (-0.005, 0.002)

	B2	P1	P2	NB1	NB2
O1	0 (-0.003, 0.003)	0 (-0.004, 0.004)	0 (-0.004, 0.004)	-0.002 (-0.006, 0.002)	-0.002 (-0.005, 0.002)
N1	0 (-0.001, 0.001)	0 (-0.004, 0.004)	0 (-0.003, 0.003)	0 (-0.004, 0.004)	0 (-0.004, 0.004)
N2	0 (-0.001, 0.001)	0 (-0.003, 0.004)	0 (-0.003, 0.003)	0 (-0.004, 0.004)	0 (-0.003, 0.004)
N3	0 (-0.001, 0.001)	0 (-0.004, 0.004)	0 (-0.003, 0.003)	0 (-0.004, 0.004)	0 (-0.004, 0.003)
B1	0 (-0.001, 0.001)	-0.001 (-0.004, 0.003)	-0.001 (-0.004, 0.002)	-0.001 (-0.005, 0.002)	-0.001 (-0.005, 0.002)
B2	0	-0.009 (-0.012, -0.006)	-0.006 (-0.009, -0.003)	-0.011 (-0.014, -0.008)	-0.01 (-0.013, -0.006)
P1	-0.009 (-0.012, -0.006)	0	0.014 (0.008, 0.018)	0.016 (0.01, 0.022)	0.021 (0.016, 0.027)
P2	-0.006 (-0.009, -0.004)	0 (-0.006, 0.004)	0	0.022 (0.017, 0.027)	0.01 (0.004, 0.015)

N2	-0.094 (-0.096, -0.092)	0.141 (0.141, 0.141)	0	0.141 (0.141, 0.141)	-0.172 (-0.172, -0.171)
N3	-0.094 (-0.096, -0.092)	0.141 (0.141, 0.141)	0.141 (0.141, 0.141)	0	-0.172 (-0.172, -0.172)
B1	-0.048 (-0.05, -0.046)	-0.172 (-0.172, -0.171)	-0.172 (-0.172, -0.171)	-0.172 (-0.172, -0.171)	0
B2	-0.049 (-0.051, -0.047)	-0.173 (-0.174, -0.172)	-0.173 (-0.174, -0.172)	-0.173 (-0.174, -0.172)	0.14 (0.139, 0.141)
NB1	-0.056 (-0.059, -0.053)	-0.161 (-0.164, -0.158)	-0.16 (-0.164, -0.157)	-0.161 (-0.164, -0.158)	-0.129 (-0.132, -0.126)
NB2	-0.033 (-0.036, -0.03)	-0.151 (-0.154, -0.148)	-0.151 (-0.154, -0.148)	-0.151 (-0.154, -0.148)	-0.118 (-0.121, -0.115)
NB3	-0.001 (-0.003, 0)	-0.094 (-0.096, -0.092)	-0.094 (-0.096, -0.092)	-0.094 (-0.096, -0.092)	-0.052 (-0.054, -0.051)
NB4	0.001 (-0.002, 0.003)	-0.099 (-0.101, -0.098)	-0.1 (-0.101, -0.097)	-0.1 (-0.102, -0.098)	-0.056 (-0.057, -0.054)

	B2	NB1	NB2	NB3	NB4
O1	-0.05 (-0.052, -0.048)	-0.05 (-0.053, -0.047)	-0.04 (-0.043, -0.037)	-0.011 (-0.013, -0.009)	0.022 (0.02, 0.024)
N1	-0.173 (-0.174, -0.172)	-0.158 (-0.161, -0.155)	-0.148 (-0.151, -0.145)	-0.094 (-0.096, -0.092)	-0.095 (-0.097, -0.093)
N2	-0.173 (-0.174, -0.172)	-0.158 (-0.161, -0.155)	-0.148 (-0.151, -0.145)	-0.094 (-0.096, -0.092)	-0.095 (-0.097, -0.093)
N3	-0.173 (-0.174, -0.172)	-0.158 (-0.162, -0.155)	-0.148 (-0.151, -0.145)	-0.094 (-0.096, -0.092)	-0.095 (-0.097, -0.093)
B1	0.14 (0.138, 0.141)	-0.126 (-0.129, -0.124)	-0.115 (-0.118, -0.112)	-0.052 (-0.054, -0.05)	-0.051 (-0.053, -0.05)
B2	0	-0.166 (-0.169, -0.163)	-0.151 (-0.154, -0.149)	-0.042 (-0.044, -0.039)	-0.045 (-0.047, -0.042)
NB1	-0.168 (-0.171, -0.165)	0	0.047 (0.043, 0.05)	-0.015 (-0.017, -0.013)	-0.008 (-0.009, -0.006)
NB2	-0.155 (-0.157, -0.153)	0.027 (0.023, 0.031)	0	-0.009 (-0.011, -0.007)	-0.001 (-0.002, 0.001)

N2	-0.02 (-0.022, -0.017)	0.039 (0.039, 0.039)	0	0.038 (0.038, 0.038)	-0.043 (-0.043, -0.042)
N3	-0.019 (-0.022, -0.017)	0.039 (0.039, 0.039)	0.039 (0.039, 0.039)	0	-0.043 (-0.043, -0.042)
B1	0.001 (-0.001, 0.004)	-0.042 (-0.042, -0.042)	-0.042 (-0.042, -0.042)	-0.042 (-0.042, -0.042)	0
B2	0.002 (-0.001, 0.004)	-0.042 (-0.043, -0.041)	-0.042 (-0.043, -0.041)	-0.042 (-0.043, -0.041)	0.017 (0.016, 0.018)
NB1	0 (-0.004, 0.003)	-0.029 (-0.033, -0.025)	-0.029 (-0.033, -0.026)	-0.029 (-0.032, -0.026)	-0.001 (-0.004, 0.003)
NB2	0 (-0.003, 0.003)	-0.028 (-0.031, -0.024)	-0.027 (-0.03, -0.024)	-0.027 (-0.03, -0.025)	-0.001 (-0.004, 0.002)
NB3	0 (-0.003, 0.003)	-0.015 (-0.017, -0.013)	-0.015 (-0.017, -0.013)	-0.015 (-0.017, -0.013)	-0.001 (-0.003, 0.001)
NB4	0 (-0.003, 0.003)	-0.016 (-0.018, -0.014)	-0.016 (-0.018, -0.014)	-0.016 (-0.018, -0.014)	0 (-0.002, 0.002)

	B2	NB1	NB2	NB3	NB4
O1	0.002 (-0.001, 0.005)	-0.008 (-0.011, -0.004)	-0.006 (-0.009, -0.003)	-0.002 (-0.005, 0.001)	0.008 (0.004, 0.011)
N1	-0.043 (-0.044, -0.042)	-0.027 (-0.031, -0.024)	-0.026 (-0.029, -0.022)	-0.015 (-0.017, -0.013)	-0.015 (-0.017, -0.012)
N2	-0.043 (-0.044, -0.042)	-0.027 (-0.031, -0.024)	-0.025 (-0.029, -0.022)	-0.015 (-0.017, -0.013)	-0.014 (-0.017, -0.012)
N3	-0.043 (-0.044, -0.042)	-0.027 (-0.031, -0.024)	-0.026 (-0.029, -0.023)	-0.015 (-0.017, -0.013)	-0.015 (-0.017, -0.012)
B1	0.018 (0.017, 0.019)	0 (-0.004, 0.003)	-0.001 (-0.004, 0.002)	-0.001 (-0.003, 0.001)	-0.001 (-0.003, 0.002)
B2	0	-0.028 (-0.031, -0.024)	-0.025 (-0.027, -0.022)	0.005 (0.003, 0.008)	0.003 (0.001, 0.006)
NB1	-0.027 (-0.031, -0.025)	0	0.034 (0.029, 0.04)	0.002 (0, 0.005)	0.014 (0.012, 0.017)
NB2	-0.025 (-0.027, -0.022)	0.004 (-0.002, 0.01)	0	0.005 (0.003, 0.007)	0.013 (0.01, 0.015)

N2	0 (-0.003, 0.003)	0 (0, 0)	0	0 (0, 0)	0 (0, 0)
N3	-0.001 (-0.003, 0.003)	0 (0, 0)	0 (0, 0)	0	0 (0, 0)
B1	0 (-0.003, 0.003)	0 (0, 0)	0 (0, 0)	0 (0, 0)	0
B2	0 (-0.003, 0.003)	0 (-0.001, 0.001)	0 (-0.001, 0.001)	0 (-0.001, 0.001)	0 (-0.001, 0.001)
NB1	-0.001 (-0.005, 0.003)	0 (-0.004, 0.004)	0 (-0.004, 0.004)	0 (-0.004, 0.004)	-0.002 (-0.005, 0.002)
NB2	-0.001 (-0.005, 0.003)	0 (-0.004, 0.003)	0 (-0.003, 0.003)	0 (-0.004, 0.003)	-0.002 (-0.005, 0.002)
NB3	0 (-0.004, 0.003)	0 (-0.002, 0.002)	0 (-0.002, 0.002)	0 (-0.002, 0.002)	0 (-0.002, 0.003)
NB4	0.001 (-0.003, 0.004)	0 (-0.002, 0.002)	0 (-0.002, 0.003)	0 (-0.002, 0.002)	0.001 (-0.002, 0.003)

	B2	NB1	NB2	NB3	NB4
O1	0 (-0.003, 0.003)	-0.002 (-0.006, 0.002)	-0.002 (-0.005, 0.003)	0 (-0.004, 0.003)	0.001 (-0.002, 0.005)
N1	0 (-0.001, 0.001)	0 (-0.003, 0.004)	0 (-0.003, 0.003)	0 (-0.002, 0.002)	0 (-0.002, 0.002)
N2	0 (-0.001, 0.001)	0 (-0.004, 0.004)	0 (-0.004, 0.003)	0 (-0.002, 0.002)	0 (-0.002, 0.002)
N3	0 (-0.001, 0.001)	0 (-0.004, 0.004)	0 (-0.004, 0.003)	0 (-0.002, 0.002)	0 (-0.002, 0.002)
B1	0 (-0.001, 0.001)	-0.002 (-0.005, 0.002)	-0.002 (-0.005, 0.002)	0 (-0.002, 0.002)	0 (-0.002, 0.003)
B2	0	-0.011 (-0.014, -0.007)	-0.01 (-0.012, -0.007)	0.002 (0, 0.005)	0.002 (-0.001, 0.005)
NB1	-0.011 (-0.014, -0.007)	0	0.028 (0.021, 0.034)	0.003 (0, 0.006)	0.012 (0.009, 0.015)

shadow: R Package for Geometric Shadow Calculations in an Urban Environment

by Michael Dorman, Evyatar Erell, Adi Vulkan, Itai Kloog

Abstract This paper introduces the **shadow** package for R. The package provides functions for shadow-related calculations in the urban environment, namely shadow height, shadow footprint and Sky View Factor (SVF) calculations, as well as a wrapper function to estimate solar radiation while taking shadow effects into account. All functions operate on a layer of polygons with a height attribute, also known as "extruded polygons" or 2.5D vector data. Such data are associated with accuracy limitations in representing urban environments. However, unlike 3D models, polygonal layers of building outlines along with their height are abundantly available and their processing does not require specialized closed-source 3D software. The present package thus brings spatio-temporal shadow, SVF and solar radiation calculation capabilities to the open-source spatial analysis workflow in R. Package functionality is demonstrated using small reproducible examples for each function. Wider potential use cases include urban environment applications such as evaluation of micro-climatic influence for urban planning, studying urban climatic comfort and estimating photovoltaic energy production potential.

Introduction

Spatial analysis of the urban environment (Biljecki et al., 2015) frequently requires estimating whether a given point is shaded or not, given a representation of spatial obstacles (e.g. buildings) and a time-stamp with its associated solar position. For example, we may be interested in -

- Calculating the amount of time a given roof or facade is shaded, to determine the utility of installing photovoltaic cells for **electricity production** (e.g. Redweik et al., 2013).
- Calculating shadow footprint on vegetated areas, to determine the expected influence of a tall new building on the surrounding **microclimate** (e.g. Bourbia and Boucheriba, 2010).

Such calculations are usually carried out using GIS-based models (Freitas et al., 2015), in either **vector-based 3D** or **raster-based 2.5D** settings. Both approaches have their advantages and limitations, as discussed in the following paragraphs.

Shadow calculations on vector-based 3D models of the urban environment are mostly restricted to proprietary closed-source software such as **ArcGIS** (ESRI, 2017) or **SketchUp** (Google, 2017), though recently some open-source models such as **SURFSUN3D** have been developed (Liang et al., 2015). One of the drawbacks of using closed-source software in this context is the difficulty of adjusting the software for specific needs and uncommon scenarios. This problem is especially acute in research settings, where flexibility and extensibility are essential for exploring new computational approaches. The other difficulty with using 3D software in urban spatial analysis concerns interoperability of file formats. Since ordinary vector spatial data formats, such as the *ESRI Shapefile*, cannot represent three-dimensional surfaces, 3D software is associated with specialized file formats. The latter cannot be readily imported to a general-purpose geocomputational environment such as R or Python (Van Rossum and Drake, 2011), thus fragmenting the analysis workflow. Moreover, most 3D software, such as those mentioned above, are design-oriented, thus providing advanced visualization capabilities but limited quantitative tools (calculating areas, angles, coordinates, etc.). Finally, true-3D databases of large urban areas are difficult to obtain, while vector-based 2.5D databases (building outline and height, see below) are almost universal. The advantages of true-3D software are "wasted" when the input data are 2.5D, while the disadvantages, such as lack of quantitative procedures and data interoperability difficulties, still remain.

Raster-based 2.5D solutions, operating on a Digital Elevation Model (DEM) raster, are much simpler and have thus been more widely implemented in various software for several decades (Kumar et al., 1997; Ratti and Richens, 2004). For example, raster-based shadow calculations are available in open-source software such as the **r.sun** command (Hofierka and Suri, 2002) in **GRASS GIS** (GRASS Development Team, 2017), the **UMEP** plugin (Lindberg et al., 2018) for **QGIS** (QGIS Development Team, 2017) and package **insol** (Corripio, 2014) in R. In the proprietary **ArcGIS** software, raster-based shadow calculations are provided through the **Solar Analyst** extension (Fu and Rich, 1999). Thanks to this variety of tools, raster-based shadow modelling can be easily incorporated within a general spatial analysis workflow. However, raster-based models are more suitable for large-scale analysis of natural terrain, rather than fine-scale urban environments, for the following reasons -

- A raster representing surface elevation, known as a DEM, at sufficiently high resolution for the urban context, may not be available and is expensive to produce, e.g. using airborne Light Detection And Ranging (LiDAR) surveys (e.g. Redweik et al., 2013). Much more commonly, municipalities and other sources such as OpenStreetMap (Haklay and Weber, 2008) offer 2.5D vector-based data on cities, i.e. polygonal layers of building outlines associated with height attributes.
- Rasters are composed of pixels, which have no natural association to specific urban elements, such as an individual building, thus making it more difficult to associate analysis results with the corresponding urban elements.
- Vertical surfaces, such as building facades, are rare in natural terrain yet very common in urban environments. Raster-based representation of facades is problematic since the latter correspond to (vertical) discontinuities in the 2.5D digital elevation model, requiring unintuitive workarounds (Redweik et al., 2013).

It should be noted that more specialized approaches have been recently developed to address some of the above-mentioned difficulties, but they are usually not available as software packages (e.g. Redweik et al., 2013; Hofierka and Zlocha, 2012).

The **shadow** package (Dorman, 2019) aims at addressing these limitations by introducing a simple 2.5D vector-based algorithm for calculating shadows, Sky View Factor (SVF) and solar radiation estimates in the urban environment. The algorithms operate on a polygonal layer extruded to 2.5D, also known as *Levels-of-Detail (LoD) 1* in the terminology of the CityGML standard (Gröger and Plümer, 2012). On the one hand, the advantages of individual urban element representation (over raster-based approach) and input data availability (over both raster-based and full 3D approaches) are maintained. On the other hand, the drawbacks of closed-source software and difficult interoperability (as opposed to full 3D environment) are avoided.

As demonstrated below, functions in the **shadow** package operate on a vector layer of obstacle outlines (e.g. buildings) along with their heights, passed as a `"SpatialPolygonsDataFrame"` object defined in package **sp** (Bivand et al., 2013; Pebesma and Bivand, 2005). The latter makes incorporating shadow calculations in *Spatial* analysis workflow in R straightforward. Functions to calculate shadow height, shadow ground footprint, Sky View Factor (SVF) and solar radiation are implemented in the package.

Theory

Shadow height

All functions currently included in package **shadow** are based on trigonometric relations in the triangle defined by the sun's rays, the ground - or a plane parallel to the ground - and an obstacle.

For example, **shadow height** at any given ground point can be calculated based on (1) sun elevation, (2) the height of the building(s) that stand in the way of sun rays and (3) the distance(s) between the queried point and the building(s) along the sun rays projection on the ground. Figure 1 depicts a scenario where shadow is being cast by building A onto the facade of building B, given the solar position defined by its elevation angle α_{elev} and azimuth angle α_{az}. Once the intersection point is identified (marked with x in Figure 1), shadow height (h_{shadow}) at the queried point (*viewer*) can be calculated based on (1) sun elevation (α_{elev}), (2) the height of building A (h_{build}) and (3) the distance ($dist_1$) between the *viewer* and intersection point x (Equation P.2.1).

$$h_{shadow} = h_{build} - dist_1 \cdot tan(\alpha_{elev}) \tag{P.2.1}$$

The latter approach can be extended to the general case of shadow height calculation at any ground location and given any configuration of obstacles. For example, if there is more than one obstacle potentially casting shadow on the queried location, we can calculate h_{shadow} for each obstacle and then take the maximum value.

Logical shadow flag

Once the shadow height is determined, we may evaluate whether any given 3D point is in shadow or not. This is done simply by comparing the Z-coordinate (i.e. height) of the queried point with the calculated shadow height at the same X-Y (i.e. ground) location.

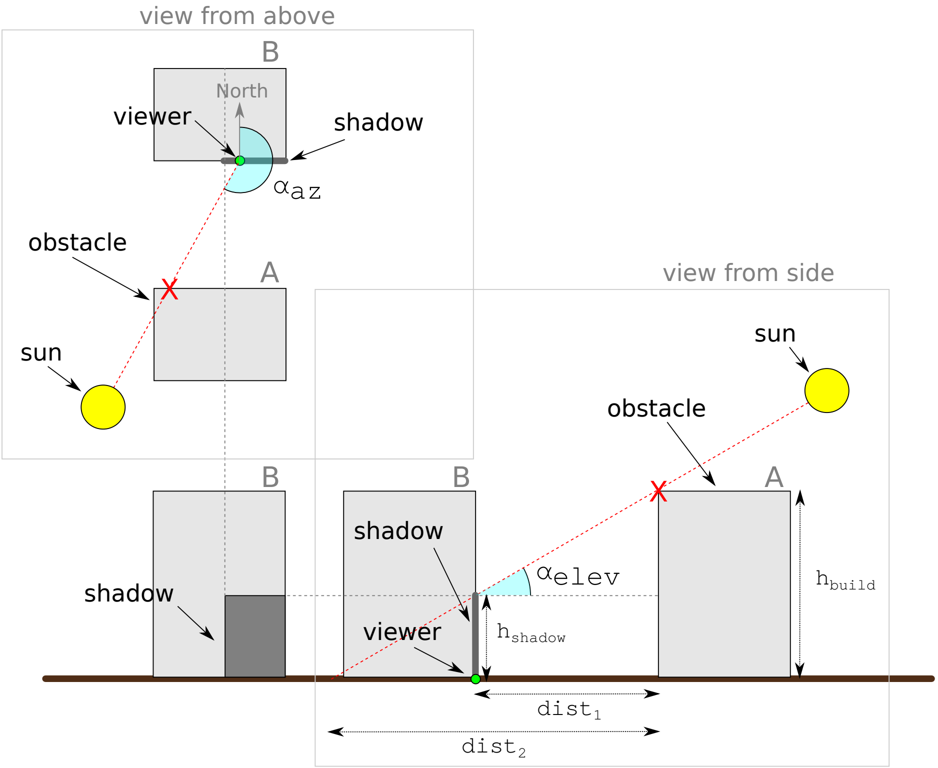


Figure 1: Shadow height calculation

Shadow footprint

Instead of calculating shadow height at a pre-specified point (e.g. the *viewer* in Figure 1), we can set h_{shadow} to zero and calculate the distance ($dist_2$) where the shadow intersects ground level (Equation P.2.2).

$$dist_2 = \frac{h_{build}}{tan(\alpha_{elev})} \tag{P.2.2}$$

Shifting the obstacle outline by the resulting distance ($dist_2$) in a direction opposite to sun azimuth (α_{az}) yields a **shadow footprint** outline (Weisthal, 2014). Shadow footprints are useful to calculate the exact ground area that is shaded at specific time. For example, Figure 2 shows the shadow footprints produced by a single building at different times of a given day.

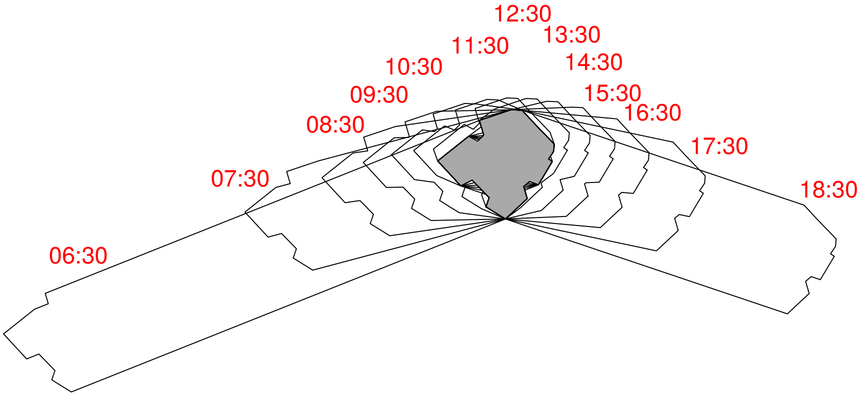


Figure 2: Shadow footprints cast by a building on a horizontal ground surface at hourly intervals on 2004-06-24. The building, indicated by the gray shaded area, is located at 31.97°N 34.78°E, and is 21.38 meters tall

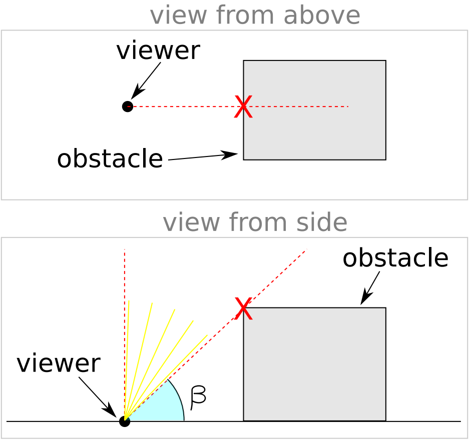


Figure 3: Sky View factor calculation

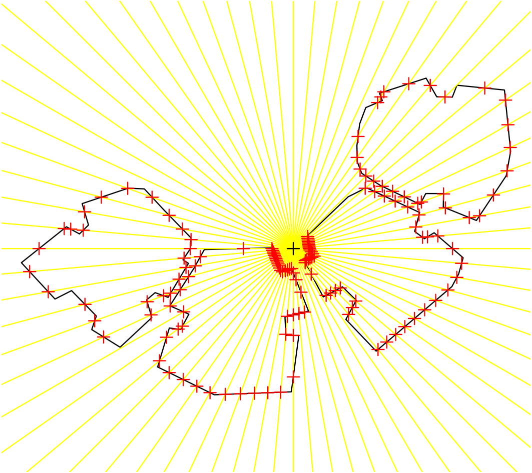


Figure 4: Angular cross sections for calculating the Sky View Factor (SVF)

Sky View Factor (SVF)

The **Sky View Factor** (Beckers, 2013; Erell et al., 2011; Grimmond et al., 2001) is the extent of sky observed from a point as a proportion of the entire sky hemisphere. The SVF can be calculated based on the maximal angles (β) formed in triangles defined by the queried location and the obstacles (Figure 3), evaluated in multiple circular cross-sections surrounding the queried location. Once the maximal angle β_i is determined for a given angular section i, SVF_i for that particular section is defined (Gál and Unger, 2014) in Equation P.2.3.

$$SVF_i = 1 - sin^2(\beta_i) \tag{P.2.3}$$

For example, in case ($\beta_i = 45^\circ$), as depicted in Figure 3, SVF_i is equal to -

$$SVF_i = 1 - sin^2(45^\circ) = 0.5$$

Averaging SVF_i values for all $i = 1, 2, ..., n$ circular cross-sections gives the final SVF estimate for the queried location (Equation P.2.4).

$$SVF = \frac{\sum_{i=1}^{n} SVF_i}{n} \tag{P.2.4}$$

The number of evaluated cross sections depends on the chosen angular resolution. For example, an angular resolution of 5° means the number of cross sections is $n = 360^\circ / 5^\circ = 72$ (Figure 4).

Solar radiation

Components

Frequently, evaluating whether a given location is shaded, and when, is just a first step towards evaluating the amount of solar radiation for a given period of time. The annual insolation at a given point is naturally affected by the degree of shading throughout the year, but shading is not the only factor.

The three components of the solar radiation are the **direct**, **diffuse** and **reflected** radiation -

- **Direct** radiation refers to solar radiation traveling on a straight line from the sun to the surface of the earth. Direct radiation can be estimated by taking into account: (1) shading, (2) surface orientation relatively to the sun, and (3) meteorological measurements of direct radiation on a horizontal plane or on a plane normal to the beam of sunlight.
- **Diffuse** radiation refers to solar radiation reaching the Earth's surface after having been scattered from the direct solar beam by molecules or particulates in the atmosphere. Diffuse radiation can be estimated by taking into account: (1) SVF, and (2) meteorological measurements of diffuse radiation at an exposed location.
- **Reflected** radiation refers to the sunlight that has been reflected off non-atmospheric obstacles such as ground surface cover or buildings. Most urban surfaces have a low albedo: asphalt reflects only 5-10 percent of incident solar radiation, brick and masonry 20-30 percent, and vegetation about 20 percent. Because a dense urban neighborhood will typically experience multiple reflections, an iterative process is required for a complete analysis. Calculating reflected radiation requires taking into account reflective properties of the various surfaces, their geometrical arrangement (Givoni, 1998) and their view factors from the receiving surface, which is beyond the scope of the **shadow** package.

The diffuse radiation component is the dominant one on overcast days, when most radiation is scattered, while the direct radiation component is dominant under clear sky conditions when direct radiation reaches the earth's surface.

Direct Normal Irradiance

Equation P.2.5 specifies the Coefficient of Direct Normal Irradiance for a vertical **facade** surface, as function of solar position given by the difference between facade azimuth and sun azimuth angles, and sun elevation angle, at time t.

$$\theta_{facade,t} = cos(\alpha_{az,t} - \alpha'_{az}) \cdot cos(\alpha_{elev,t}) \tag{P.2.5}$$

In Equation P.2.5, $\theta_{facade,t}$ is the Coefficient of Direct Normal Irradiance on a facade at time t, $\alpha_{az,t}$ is the sun azimuth angle at time t (see Figure 1), α'_{az} is the facade azimuth angle, i.e. the direction where the facade is facing, and $\alpha_{elev,t}$ is sun elevation angle at time t (see Figure 1). Note that all of latter variables, with the exception of facade azimuth angle α'_{az}, are specific for the time interval t due to the variation in solar position.

Horizontal **roof** surfaces, unlike facades, are not tilted towards any particular azimuth[1]. Equation P.2.5 thus simplifies to Equation P.2.6 when referring to a roof, rather than a facade, surface.

$$\theta_{roof,t} = cos(90^\circ - \alpha_{elev,t}) \tag{P.2.6}$$

Figure 5 demonstrates the relation given in Equations P.2.5 and P.2.6 for the entire relevant range of solar positions relative to facade or roof orientation. Again, note that for roof surfaces, the $\theta_{roof,t}$ coefficient is only dependent on sun elevation angle $\alpha_{elev,t}$ (Equation P.2.6) as illustrated on the **right** panel of Figure 5. (The code for producing Figure 5 can be found in the help page of function `coefDirect` from **shadow**).

For example, the left panel in Figure 5 shows that maximal proportion of incoming solar radiation (i.e. $\theta_{facade,t} = 1$) on a facade surface is attained when facade azimuth is equal to sun azimuth and sun elevation is 0 ($\alpha_{elev,t} = 0^\circ$, i.e. facade directly facing the sun). Similarly, the right panel shows that maximal proportion of solar radiation on a roof surface (i.e. $\theta_{roof,t} = 1$) is attained when the sun is at the zenith ($\alpha_{elev,t} = 90^\circ$, i.e. sun directly above the roof).

Once the Coefficient of Direct Normal Irradiance $\theta_{facade,t}$ or $\theta_{roof,t}$ is determined, the Direct Normal Irradiance meteorological measurement $rad_{direct,t}$ referring to the same time interval t,

[1] It should be noted that roof surfaces may be pitched rather than horizontal; however 2.5D models, which **shadow** supports, can only represent horizontal roofs

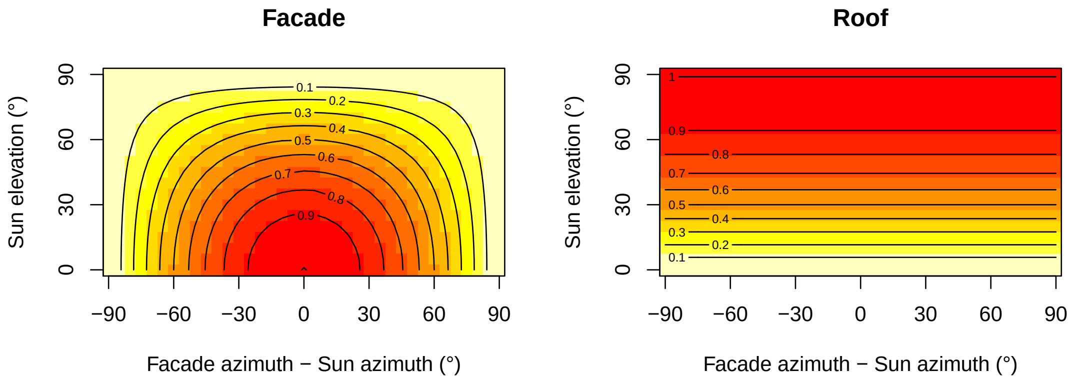


Figure 5: Coefficient of Direct Normal Irradiance, as function of solar position, expressed as the difference between facade and sun azimuths (X-axis) and sun elevation (Y-axis). The *left* panel refers to a facade, the *right* panel refers to a roof. Note that a horizontal roof has no azimuth, thus the X-axis is irrelevant for the *right* panel and only shown for uniformity

usually on an hourly time step, is multiplied by the coefficient at a point on the building surface to give the local irradiation at that point (Equation P.2.7). The result $rad'_{direct,t}$ is the corrected Direct Irradiance the surface receives given its orientation relative to the solar position.

$$rad'_{direct,t} = \theta_t \cdot rad_{direct,t} \tag{P.2.7}$$

Both $rad_{direct,t}$ and $rad'_{direct,t}$, as well as $rad_{diffuse,t}$, $rad'_{diffuse,t}$ (Equation P.2.8) and rad'_{total} (Equation P.2.9) (see below), are given for each time interval t in units of power per unit area, such as kWh/m^2.

Diffuse Horizontal Irradiance

Moving on to discussing the second component in the radiation balance, the diffuse irradiance. Diffuse irradiance is given by the meteorological measurement of Diffuse Horizontal Irradiance $rad_{diffuse,t}$, which needs to be corrected for the specific proportion of viewed sky given surrounding obstacles expressed by SVF. Assuming isotropic contribution (Freitas et al., 2015), $rad'_{diffuse,t}$ is the corrected diffuse irradiance the surface receives (Equation P.2.8). Note that SVF is unrelated to solar position; it is a function of the given configuration of the queried location and surrounding obstacles, and is thus invariable for all time intervals t.

$$rad'_{diffuse,t} = SVF \cdot rad_{diffuse,t} \tag{P.2.8}$$

Total irradiance

Finally, the direct and diffuse radiation estimates are summed for all time intervals t to obtain the total (e.g. annual) insolation for the given surface rad'_{total} (Equation P.2.9). The sum refers to n intervals $t = 1, 2, ..., n$, commonly $n = 24 \times 365 = 8,760$ when referring to an annual radiation estimate using an hourly time step.

$$rad'_{total} = \sum_{t=1}^{n} rad'_{direct,t} + \sum_{t=1}^{n} rad'_{diffuse,t} \tag{P.2.9}$$

Package structure

The **shadow** package contains four “low-level” functions, one “high-level” function, and several “helper functions”.

The “low-level” functions calculate distinct aspects of shading, and the SVF -

- `shadowHeight` - Calculates shadow height
- `inShadow` - Determines a logical shadow flag (in shadow or not)

Function	**Location**	**Obstacles**	**Sun Pos.**	**Output**
`shadowHeight`	Points (2D) / Raster	Polygons	Matrix	Numeric matrix / Raster
`inShadow`	Points (2D/3D) / Raster	Polygons	Matrix	Logical matrix / Raster
`shadowFootprint`	-	Polygons	Matrix	Polygons
`SVF`	Points (2D/3D) / Raster	Polygons	-	Numeric vector / Raster

Table 1: Inputs and outputs for main functions in package `shadow`

- `shadowFootprint` - Calculates shadow footprint
- `SVF` - Calculates the SVF

Table 1 gives a summary of the (main) input and output object types for each of the "low-level" functions. The following list clarifies the exact object classes referenced in the table -

- The queried locations **points** (e.g. the *viewer* point in Figure 1) can be specified in several ways. Points (`"SpatialPoints*"`) can be either 2D, specifying ground locations, or 3D[2] - specifying any location on the ground or above ground. Alternatively, a **raster** (`"Raster*"`) can be used to specify a regular grid of ground locations. Note that the shadow height calculation only makes sense for ground locations, as height above ground is what the function calculates, so it is not applicable for 3D points
- The obstacle **polygons** are specified as a `"SpatialPolygonsDataFrame"` object having a **height** attribute ("extrusion" height) given in the same units as the layer Coordinate Reference System (CRS), usually meters. Geographic coordinates (long/lat) are not allowed because these units are meaningless for specifying height
- Solar position **matrix** is given as a `"matrix"` object, where the first column specifies **sun azimuth** angle and the second column specifies **sun elevation** angle. Both angles should be given in decimal degrees, where -
  - **sun azimuth** (e.g. α_{az} in Figure 1) is measured clockwise relative to North, i.e North $= 0^\circ$, East $= 90^\circ$, South $= 180^\circ$, West $= 270^\circ$
  - **sun elevation** (e.g. α_{elev} in Figure 1) is measured relatively to a horizontal surface, i.e. sun on the horizon $= 0^\circ$, sun at its zenith $= 90^\circ$
- The **output** of `shadowHeight` and `inShadow` is a numeric or logical `"matrix"`, respectively, where rows represent locations and columns represent solar positions. The output of `shadowFootprint` is a polygonal layer of footprints. The output of `SVF` is a numeric vector where values correspond to locations. All functions that can accept a raster of ground locations return a corresponding raster of computed values

The "high-level" function `radiation` is a wrapper around `inShadow` and `SVF` for calculating direct and diffuse solar radiation on the obstacle surface area (i.e. building roofs and facades). In addition to the geometric layers and solar positions, this function also requires meteorological measurements of direct and diffuse radiation at an unobstructed weather station. The **shadow** package provides a sample Typical Meteorological Year (TMY) dataset `tmy` to illustrate the usage of the `radiation` function (see below). Similar TMY datasets were generated for many areas (e.g. Pusat et al., 2015) and are generally available from meteorological agencies, or from databases for building energy simulation such as EnergyPlus (**?**).

Finally, the **shadow** package provides several "helper functions" which are used internally by "low-level" and "high-level" functions, but can also be used independently -

[2] The third dimension of 3D points has to be specified using three-dimensional *coordinates*, rather than a "height" attribute in a 2D point layer (see Examples section)

- `classifyAz` - Determines the azimuth where the perpendicular of a line segment is facing; used internally to classify facade azimuth
- `coefDirect` - Calculates the Coefficient of Direct Normal Irradiance reduction (Equations P.2.5 and P.2.6)
- `plotGrid` - Makes an interactive plot of 3D spatial points. This is a wrapper around `scatterplot3js` from package **threejs** (Lewis, 2017)
- `ray` - Creates a spatial line between two given points
- `shiftAz` - Shifts spatial features by azimuth and distance
- `surfaceGrid` - Creates a 3D point layer with a grid which covers the facades and roofs of obstacles
- `toSeg` - Splits polygons or lines to segments

The following section provides a manual for using these functions through a simple example with four buildings.

Examples

In this section we demonstrate the main functionality of **shadow**, namely calculating -

- Shadow height (function `shadowHeight`)
- Logical shadow flag (function `inShadow`)
- Shadow footprint (function `shadowFootprint`)
- Sky View Factor (function `SVF`)
- Solar radiation (function `radiation`)

Before going into the examples, we load the **shadow** package. Package **sp** is loaded automatically along with **shadow**. Packages **raster** (Hijmans, 2017) and **rgeos** (Bivand and Rundel, 2017) are used throughout the following code examples for preparing the inputs and presenting the results, so they are loaded as well.

```
> library(shadow)
> library(raster)
> library(rgeos)
```

In the examples, we will use a small real-life dataset representing four buildings in Rishon-Le-Zion, Israel (Figure 6), provided with package **shadow** and named `build`.

The following code section also creates a hypothetical circular green park located 20 meters to the north and 8 meters to the west from the buildings layer centroid (hereby named `park`).

```
> location = gCentroid(build)
> park_location = shift(location, y = 20, x = -8)
> park = gBuffer(park_location, width = 12)
```

The following expressions visualize the `build` and `park` layers as shown in Figure 6. Note that the `build` layer has an attribute named `BLDG_HT` specifying the height of each building (in meters), as shown using text labels on top of each building outline.

```
> plot(build, col = "lightgrey")
> text(gCentroid(build, byid = TRUE), build$BLDG_HT)
> plot(park, col = "lightgreen", add = TRUE)
```

Shadow height

The `shadowHeight` function calculates shadow height(s) at the specified point location(s), given a layer of obstacles and solar position(s). The `shadowHeight` function, as well as other functions that require a solar position argument such as `inShadow`, `shadowFootprint` and `radiation` (see below), alternatively accept a time argument instead of the solar position. In case a time (`time`) argument is passed instead of solar position (`solar_pos`), the function internally calculates solar position using the lon/lat of the `location` layer centroid and the specified time, using function `solarpos` from package **maptools** (Bivand and Lewin-Koh, 2017).

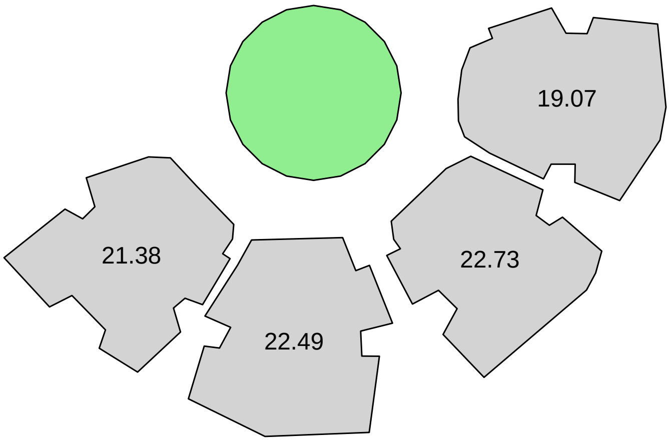


Figure 6: Sample data: a buildings layer and a green park layer. Text labels express building height in meters.

In the following example, we would like to calculate shadow height at the centroid of the buildings layer (`build`) on 2004-12-24 at 13:30:00. First we create the queried points layer (`location`), in this case consisting of a single point: the `build` layer centroid. This is our layer of locations where we would like to calculate shadow height.

```
> location = gCentroid(build)
```

Next we need to specify the solar position, i.e. sun elevation and azimuth, at the particular time and location (31.967°N 34.777°E), or let the function calculate it automatically based on the time. Using the former option, we can figure out solar position using function `solarpos` from package maptools. To do that, we first define a `"POSIXct"` object specifying the time we are interested in -

```
> time = as.POSIXct(
+   x = "2004-12-24 13:30:00",
+   tz = "Asia/Jerusalem"
+ )
```

Second, we find the longitude and latitude of the point by reprojecting it to a geographic CRS[3].

```
> location_geo = spTransform(
+   x = location,
+   CRSobj = "+proj=longlat +datum=WGS84"
+ )
```

Finally, we use the `solarpos` function to find solar position, given longitude, latitude and time -

```
> library(maptools)
> solar_pos = solarpos(
+   crds = location_geo,
+   dateTime = time
+ )
```

We now know the sun azimuth (208.7°) and elevation (28.8°) -

```
> solar_pos

#>          [,1]     [,2]
#> [1,] 208.7333 28.79944
```

Given the solar position along with the layer of obstacles `build`, shadow height in `location` can be calculated using the `shadowHeight` function, as follows -

[3] Note that calculating solar position is the only example where lon/lat coordinates are needed when working with **shadow**. All other spatial inputs are required to be passed in a projected CRS, due to the fact that obstacles height is meaningless to specify in lon/lat degree units

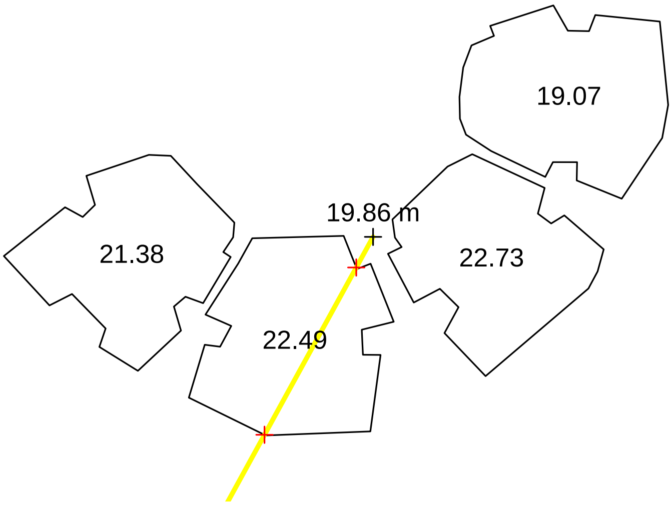


Figure 7: Shadow height (m) at a single point (indicated by black + symbol)

```
> h = shadowHeight(
+   location = location,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos
+ )
```

The resulting object contains the shadow height value of 19.86 meters -

```
> h

#>           [,1]
#> [1,] 19.86451
```

The second (shorter) approach is letting the function calculate solar position for us, in which case we can pass just the spatial layers and the `time`, without needing to calculate solar position ourselves -

```
> shadowHeight(
+   location = location,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   time = time
+ )

#>           [,1]
#> [1,] 19.86451
```

The results of both approaches are identical. The first approach, where solar position is manually defined, takes more work and thus may appear unnecessary. However, it is useful for situations when we want to use specific solar positions from an external data source, or to evaluate arbitrary solar positions that cannot be observed in the queried location in real life.

Either way, the resulting object `h` is a `"matrix"`, though in this case it only has a single row and a single column. The `shadowHeight` function accepts location layers with more than one point, in which case the resulting `"matrix"` will have additional rows. It also accepts more than one solar position or time value (see below), in which case the resulting `"matrix"` will have additional columns. It is thus possible to obtain a matrix of shadow height values for a set of locations in a set of times.

Figure 7 illustrates how the shadow height calculation was carried out. First, a line of sight is drawn between the point of interest and the sun direction based on sun azimuth (shown as a yellow line). Next, potential intersections are detected (marked with + symbols). Finally, shadow height induced by each intersection is calculated based on the distance towards intersection, sun elevation and intersected building height (see Figure 1). The final result is the maximum of the per-intersection heights.

The procedure can be readily expanded to calculate a continuous surface of shadow heights, as the `shadowHeight` function also accepts `"Raster*"` objects (package **raster**). The raster serves as

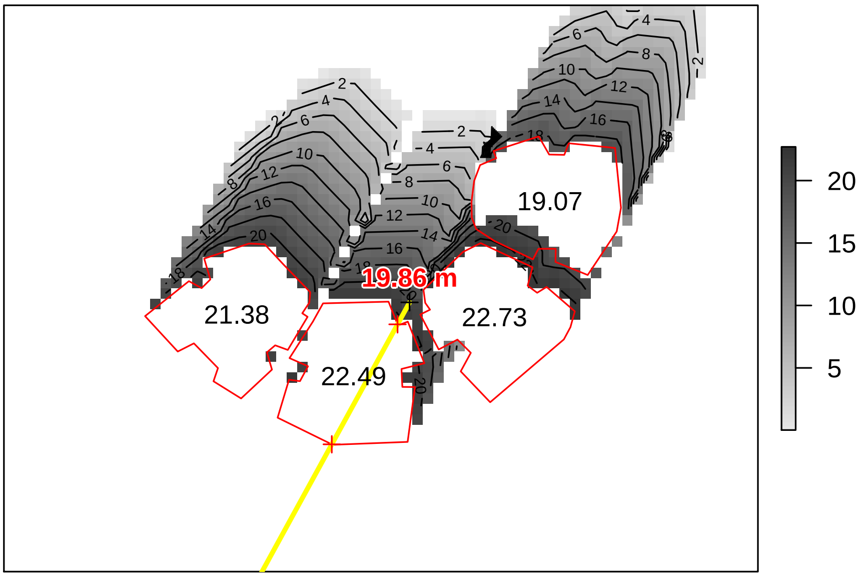


Figure 8: Shadow height (m) surface, and an individual shadow height value (indicated by black + symbol at the center of the image)

a template, defining the grid where shadow height values will be calculated. For example, in the following code section we create such a template raster covering the examined area plus a 50-meter buffer on all sides, with a spatial resolution of 2 meters -

```
> ext = as(extent(build) + 50, "SpatialPolygons")
> r = raster(ext, res = 2)
> proj4string(r) = proj4string(build)
```

Now we can calculate a shadow height raster by simply replacing the `location` argument with the raster `r` -

```
> height_surface = shadowHeight(
+   location = r,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos,
+   parallel = 5
+ )
```

The result (`height_surface`), in this case, is not a matrix - it is a shadow height surface (a `"RasterLayer"` object) of the same spatial dimensions as the input template `r`. Note that unshaded pixels get an `NA` shadow height value, thus plotted in white (Figure 8). Also note the partial shadow on the roof of the north-eastern building (top-right) caused by the neighboring building to the south-west.

The additional `parallel=5` argument splits the calculation of raster cells among 5 processor cores, thus making it faster. A different number can be specified, depending the number of available cores. Behind the scenes, parallel processing relies on the **parallel** package (R Core Team, 2018).

Shadow (logical)

Function `shadowHeight`, introduced in the previous section, calculates *shadow height* for a given ground location. In practice, the metric of interest is very often *whether* a given 3D location is in shade or not. Such a logical flag can be determined by comparing the Z-coordinate (i.e. the height) of the queried point with the calculated shadow height at the same X-Y location. The `inShadow` function is a wrapper around `shadowHeight` for doing that.

The `inShadow` function gives the logical shadow/non-shadow classification for a set of 3D points. The function basically calculates shadow height for a given unique ground location (X-Y), then compares it with the elevation (Z) of all points in that location. The points which are positioned "above" the shadow are considered non-shaded (receiving the value of `FALSE`), while the points which are positioned "below" the shadow are considered shaded (receiving the value of `TRUE`).

The 3D points we are interested in when doing urban analysis are usually located on the surface of elements such as buildings. The surfaceGrid helper function can be used to automatically generate a **grid** of such surface points. The inputs for this function include the obstacle layer for which to generate a surface grid and the required grid resolution. The returned object is a 3D point layer.

For example, the following expression calculates a 3D point layer named grid covering the build surface at a resolution of 2 meters -

```
> grid = surfaceGrid(
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   res = 2
+ )
```

The resulting grid points are associated with all attributes of the original obstacles each surface point corresponds to, as well as six new attributes -

- obs_id - Unique consecutive ID for each feature in obstacles
- type - Either "facade" or "roof"
- seg_id - Unique consecutive ID for each facade segment (only for “facade” points)
- xy_id - Unique consecutive ID for each ground location (only for “facade” points)
- facade_az - The azimuth of the corresponding facade, in decimal degrees (only for “facade” points)

In this case, the resulting 3D point grid has 2,693 features, starting with "roof" points -

```
> head(grid)

#>   build_id BLDG_HT obs_id type seg_id xy_id facade_az
#> 1      722   22.49      3 roof     NA    NA        NA
#> 2      722   22.49      3 roof     NA    NA        NA
#> 3      722   22.49      3 roof     NA    NA        NA
#> 4      722   22.49      3 roof     NA    NA        NA
#> 5      722   22.49      3 roof     NA    NA        NA
#> 6      722   22.49      3 roof     NA    NA        NA
```

Then going through the "facade" points -

```
> tail(grid)

#>       build_id BLDG_HT obs_id   type seg_id xy_id facade_az
#> 19610      831   19.07      4 facade     74    44  100.2650
#> 19710      831   19.07      4 facade     75    45  123.6695
#> 19810      831   19.07      4 facade     75    46  123.6695
#> 19910      831   19.07      4 facade     75    47  123.6695
#> 20010      831   19.07      4 facade     75    48  123.6695
#> 20110      831   19.07      4 facade     75    49  123.6695
```

Printing the coordinates confirms that, indeed, grid is a 3D point layer having three-dimensional coordinates where the third dimension h represents height above ground -

```
> head(coordinates(grid))

#>         x1      x2    h
#> 1 667882.9 3538086 22.5
#> 2 667884.9 3538086 22.5
#> 3 667886.9 3538086 22.5
#> 4 667888.9 3538086 22.5
#> 5 667890.9 3538086 22.5
#> 6 667892.9 3538086 22.5
```

Once the 3D grid is available, we can evaluate whether each point is in shadow or not, at the specified solar position(s), using the inShadow wrapper function -

```
> s = inShadow(
+   location = grid,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos
+ )
```

The resulting object `s` is a `"logical"` matrix with rows corresponding to the grid features and columns corresponding to the solar positions. In this particular case a single solar position was evaluated, thus the matrix has just one column -

```
> dim(s)
```

```
#> [1] 2693    1
```

The `scatter3D` function from package **plot3D** (Soetaert, 2017) is useful for visualizing the result. In the following code section, we use two separate `scatter3D` function calls to plot the grid with both variably colored filled circles (yellow or grey) and constantly colored (black) outlines.

```
> library(plot3D)
> scatter3D(
+   x = coordinates(grid)[, 1],
+   y = coordinates(grid)[, 2],
+   z = coordinates(grid)[, 3],
+   theta = 55,
+   colvar = s[, 1],
+   col = c("yellow", "grey"),
+   pch = 16,
+   scale = FALSE,
+   colkey = FALSE,
+   cex = 1.1
+ )
> scatter3D(
+   x = coordinates(grid)[, 1],
+   y = coordinates(grid)[, 2],
+   z = coordinates(grid)[, 3],
+   theta = 55,
+   col = "black",
+   pch = 1,
+   lwd = 0.1,
+   scale = FALSE,
+   colkey = FALSE,
+   cex = 1.1,
+   add = TRUE
+ )
```

The output is shown in Figure 9. It shows the 3D `grid` points, along with the `inShadow` classification encoded as point color: grey for shaded surfaces, yellow for sun-exposed surfaces.

Shadow footprint

The `shadowFootprint` function calculates the geometry of shadow projection on the ground. The resulting footprint layer can be used for various applications. For example, a shadow footprint layer can be used to calculate the proportion of shaded surface in a defined area, or to examine which obstacles are responsible for shading a given urban element.

In the following example, the `shadowFootprint` function is used to determine the extent of shading on the hypothetical green park (Figure 6) at different times of day. First, let us consider a single time instance of 2004-06-24 09:30:00. At this particular time and geographical location, the solar position is at an azimuth of 88.8° and at an elevation of 46.7° -

```
> time2 = as.POSIXct(
+   x = "2004-06-24 09:30:00",
+   tz = "Asia/Jerusalem"
+ )
> solar_pos2 = solarpos(
+   crds = location_geo,
+   dateTime = time2
+ )
> solar_pos2
```

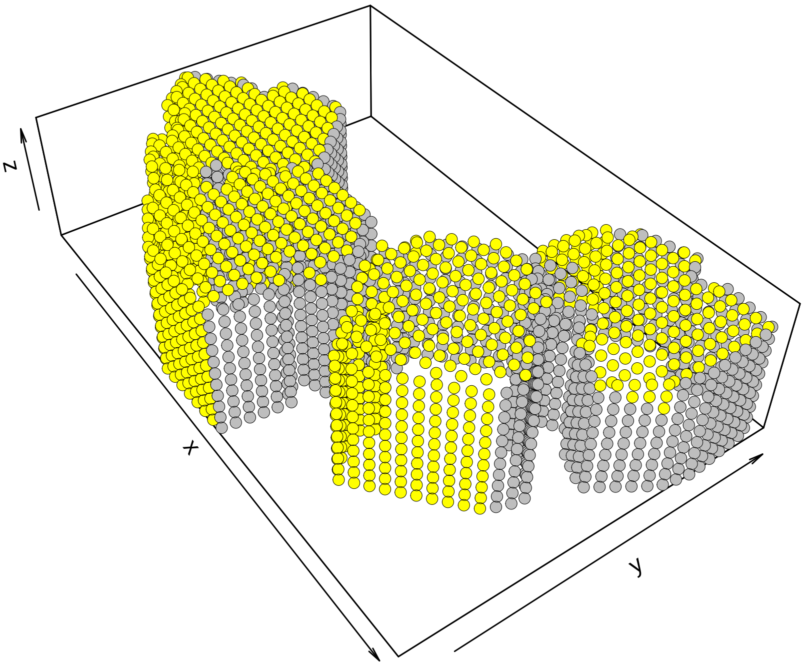


Figure 9: Buildings surface points in shadow (grey) and in direct sunlight (yellow) on 2004-12-24 13:30:00

```
#>          [,1]   [,2]
#> [1,] 88.83113 46.724
```

The following expression calculates the shadow footprint for this particular solar position.

```
> footprint = shadowFootprint(
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos2
+ )
```

The resulting object `footprint` is a polygonal layer (`"SpatialPolygonsDataFrame"` object) which can be readily used in other spatial calculations. For example, the footprint and park polygons can be *intersected* to calculate the proportion of shaded park area within total park area, as follows.

```
> park_shadow = gIntersection(park, footprint)
> shade_prop = gArea(park_shadow) / gArea(park)
> shade_prop
```

```
#> [1] 0.3447709
```

The numeric result `shade_prop` gives the proportion of shaded park area, 0.34 in this case (Figure 10).

The shadow footprint calculation can also be repeated for a sequence of times, rather than a single one, to monitor the daily (monthly, annual, etc.) course of shaded park area proportion. To do that, we first need to prepare the set of solar positions in the evaluated dates/times. Again, this can be done using function `solarpos`. For example, the following code creates a matrix named `solar_pos_seq` containing solar positions over the 2004-06-24 at hourly intervals -

```
> time_seq = seq(
+   from = as.POSIXct("2004-06-24 03:30:00", tz = "Asia/Jerusalem"),
+   to = as.POSIXct("2004-06-24 22:30:00", tz = "Asia/Jerusalem"),
+   by = "1 hour"
+ )
> solar_pos_seq = solarpos(
+   crds = location_geo,
+   dateTime = time_seq
+ )
```

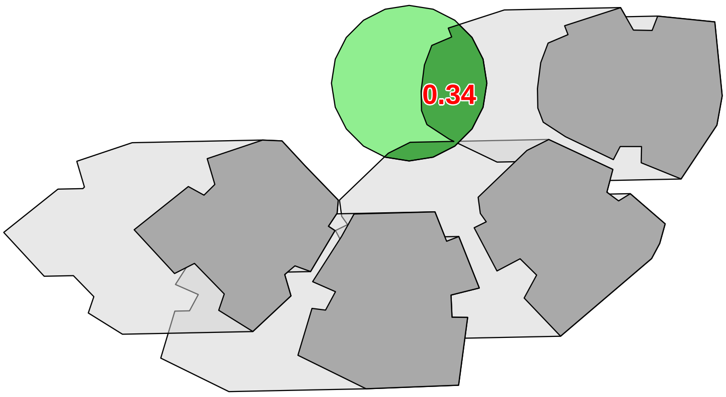


Figure 10: Shaded park proportion on 2004-06-24 09:30:00

Note that the choice of an *hourly* interval is arbitrary. Shorter intervals (e.g. 30 mins) can be used for increased accuracy.

To calculate the shaded park proportion at each time step we can loop over the solar_pos_seq matrix, each time -

- Calculating shadow footprint
- Intersecting the shadow footprint with the park outline
- Calculating the ratio of intersection and total park area

The code of such a for loop is given below.

```
> shadow_props = rep(NA, nrow(solar_pos_seq))
> for(i in 1:nrow(solar_pos_seq)) {
+   if(solar_pos_seq[i, 2] < 0) shadow_props[i] = 1 else {
+       footprint =
+         shadowFootprint(
+           obstacles = build,
+           obstacles_height_field = "BLDG_HT",
+           solar_pos = solar_pos_seq[i, , drop = FALSE]
+           )
+     park_shadow = gIntersection(park, footprint)
+     if(is.null(park_shadow))
+       shadow_props[i] = 0
+     else
+       shadow_props[i] = gArea(park_shadow) / gArea(park)
+     }
+ }
```

The loop creates a numeric vector named shadow_props. This vector contains shaded proportions for the park in agreement with the times we specified in time_seq. Note that two conditional statements are being used to deal with special cases -

- Shadow proportion is set to 1 (i.e. maximal) when sun is below the horizon
- Shadow proportion is set to 0 (i.e. minimal) when no intersections are detected between the park and the shadow footprint

Plotting shadow_props as function of time_seq (Figure 11) summarizes the daily course of shaded park proportion on the 2004-06-24. The individual value of 0.34 which we have calculated for 09:30 in the previous example (Figure 10) is highlighted in red.

Sky View Factor

The SVF function can be used to estimate the SVF at any 3D point location. For example, the following expression calculates the SVF on the ground[4] at the centroid of the build layer (Figure 4).

[4] Recall (Table 1) that the inShadow and SVF functions accept either 2D or 3D points, whereas 2D points are treated as ground locations

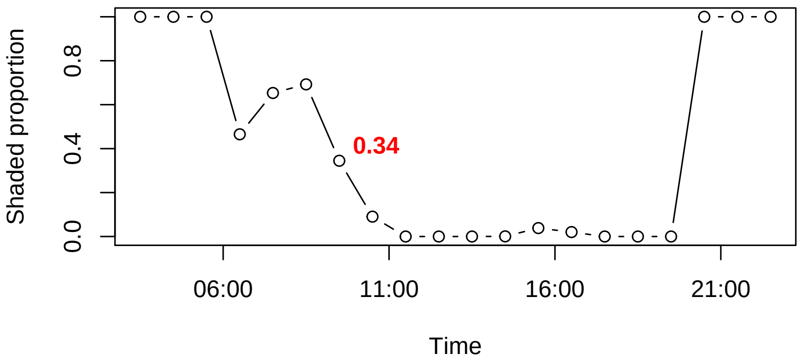


Figure 11: Shaded park proportion at each hourly time step on 2004-06-24

```
> s = SVF(
+   location = location,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT"
+ )
```

The resulting SVF is 0.396, meaning that about 39.6% of the sky area are visible (Figure 12) from this particular location.

```
> s
```

```
#> [1] 0.3959721
```

Note that the `SVF` function has a tuning parameter named `res_angle` which can be used to modify angular resolution (default is 5°, as shown in Figure 4). A smaller `res_angle` value will give more accurate SVF but slower calculation.

Given a "template" grid, the latter calculation can be repeated to generate a continuous surface of SVF estimates for a grid of ground locations. In the following code section we calculate an SVF surface using the same raster template with a resolution of 2 meters from the shadow height example (see above).

```
> svf_surface = SVF(
+   location = r,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   parallel = 5
+ )
```

Note that the `parallel=5` option is used once again to make the calculation run simultaneously on 5 cores. The resulting SVF surface is shown in Figure 12. As could be expected, SVF values are lowest in the vicinity of buildings due to their obstruction of the sky.

Solar radiation

Shadow height, shadow footprint and SVF can be considered as low-level geometric calculations. Frequently, the ultimate aim of an analysis is the estimation of insolation, which is dependent on shadow and SVF but also on surface orientation and meteorological solar radiation conditions. Thus, the low-level geometric calculations are frequently combined and wrapped with meteorological solar radiation estimates to take the geometry into account when evaluating insolation over a given time interval. The **shadow** package provides this kind of wrapper function named `radiation`.

The `radiation` function needs several parameters to run -

- **3D points grid** representing surfaces where the solar radiation is evaluated. It is important to specify whether each grid point is on a "roof" or on a "facade", and the azimuth it is facing (only for "facade"). A grid with those attributes can be automatically produced using the `surfaceGrid` function (see above)
- **Obstacles layer** defined with `obstacles`, having an `obstacles_height_field` attribute (see above)
- **Solar positions** defined with `solar_pos` (see above)

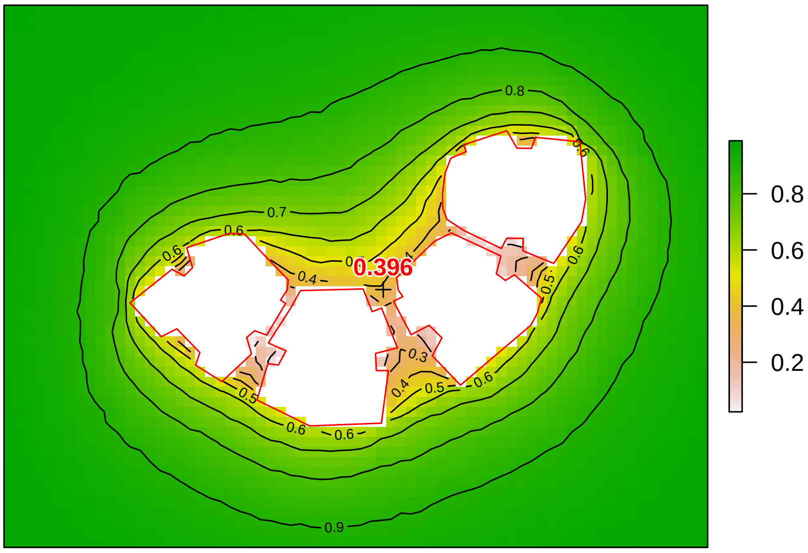


Figure 12: Sky View Factor (SVF) surface, with SVF value for an individual point (indicated by black + symbol at the center of the image)

- **Meteorological estimates** defined with `solar_normal` and `solar_diffuse`, corresponding to the same time intervals given by `solar_pos`

Given this set of inputs, the `radiation` function:

- calculates whether each `grid` surface point is in shadow or not, for each solar position `solar_pos`, using the `inShadow` function (Equation P.2.1),
- calculates the Coefficient of Direct Normal Irradiance reduction, for each `grid` surface point at each solar position `solar_pos`, using the `coefDirect` function (Equations P.2.5 and P.2.6),
- combines shadow, the coefficient and the meteorological estimate `solar_normal` to calculate the **direct** radiation (Equation P.2.7),
- calculates the SVF for each `grid` surface point, using the `SVF` function (Equations P.2.3 and P.2.4),
- combines the SVF and the meteorological estimate `solar_diffuse` to calculate the **diffuse** radiation (Equation P.2.8)
- and calculates the sums of the **direct**, **diffuse** and **total** (i.e. direct+diffuse) solar radiation per `grid` surface point for the entire period (Equation P.2.9).

To demonstrate the `radiation` function, we need one more component not used in the previous examples: the reference solar radiation data. The `shadow` package comes with a sample Typical Meteorological Year (TMY) dataset named `tmy` that can be used for this purpose. This dataset was compiled for the same geographical area where the buildings are located, and therefore can be realistically used in our example.

The `tmy` object is a `data.frame` with 8,760 rows, where each row corresponds to an hourly interval over an entire year ($24 \times 365 = 8,760$). The attributes given for each hourly interval include solar position (`sun_az`, `sun_elev`) and solar radiation measurements (`solar_normal`, `solar_diffuse`). Both solar radiation measurements are given in W/m^2 units.

```
> head(tmy, 10)
```

```
#>                   time sun_az sun_elev solar_normal solar_diffuse  dbt  ws
#> 1  1999-01-01 01:00:00  66.73   -70.94            0             0  6.6 1.0
#> 2  1999-01-01 02:00:00  82.02   -58.68            0             0  5.9 1.0
#> 3  1999-01-01 03:00:00  91.00   -45.99            0             0  5.4 1.0
#> 4  1999-01-01 04:00:00  98.13   -33.32            0             0  4.9 1.0
#> 5  1999-01-01 05:00:00 104.81   -20.86            0             0  4.4 1.0
#> 6  1999-01-01 06:00:00 111.73    -8.76            0             6  4.8 1.0
#> 7  1999-01-01 07:00:00 119.41     2.91          118            24  7.3 1.0
#> 8  1999-01-01 08:00:00 128.39    13.30          572            45 11.2 1.0
#> 9  1999-01-01 09:00:00 139.20    22.46          767            57 16.0 1.0
#> 10 1999-01-01 10:00:00 152.33    29.63          809            66 16.3 2.1
```

The Direct Normal Irradiance (solar_normal) is the amount of solar radiation received per unit area by a surface that is always held normal to the incoming rays from the sun's current position in the sky. This is an estimate of maximal **direct radiation**, obtained on an optimally tilted surface. The Diffuse Horizontal Irradiance (solar_diffuse) is the amount of radiation received per unit area at a surface that has not arrived on a direct path from the sun, but has been scattered by molecules and particles in the atmosphere. This is an estimate of **diffuse radiation**.

To use the solar positions from the tmy dataset, we create a separate matrix with just the sun_az and sun_elev columns -

```
> solar_pos = as.matrix(tmy[, c("sun_az", "sun_elev")])
```

The first few rows of this matrix are -

```
> head(solar_pos)

#>   sun_az sun_elev
#> 2  66.73   -70.94
#> 3  82.02   -58.68
#> 4  91.00   -45.99
#> 5  98.13   -33.32
#> 6 104.81   -20.86
#> 7 111.73    -8.76
```

Now we have everything needed to run the radiation function. We are hereby using the same grid layer with 3D points covering the roofs and facades of the four buildings created above using the surfaceGrid function (Figure 9), the layer of obstacles, and the solar position and measured solar radiation at a reference weather station from the tmy table.

```
> rad = radiation(
+   grid = grid,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos,
+   solar_normal = tmy$solar_normal,
+   solar_diffuse = tmy$solar_diffuse,
+   parallel = 5
+ )
```

The returned object rad is a data.frame with the summed direct, diffuse and total (i.e. direct+diffuse) solar radiation estimates, as well as the SVF, for each specific surface location in grid. Summation takes place over the entire period given by solar_pos, solar_normal and solar_diffuse. In the present case it is an **annual** insolation. The units of measurement are therefore Wh/m^2 summed over an entire year.

For example, the following printout -

```
> head(rad)

#>         svf  direct  diffuse   total
#> 1 0.9999875 1242100 473334.1 1715434
#> 2 0.9999830 1242100 473332.0 1715432
#> 3 0.9999778 1242100 473329.5 1715429
#> 4 0.9999685 1242100 473325.1 1715425
#> 5 0.9999538 1242099 473318.2 1715417
#> 6 0.9999396 1242099 473311.4 1715411
```

refers to the first six surface points which are part of the same roof, thus sharing similar annual solar radiation estimates. Overall, however, the differences in insolation are very substantial among different locations on the buildings surfaces, as shown in Figure 13. For example, the roofs receive about twice as much direct radiation as the south-facing facades. The code for producing Figure 13, using function scatter3D (see Figure 9), can be found on the help page of the radiation function and is thus omitted here to save space. Note that the figure shows radiation estimates in kWh/m^2 units, i.e. the values from the rad table (above) divided by 1000.

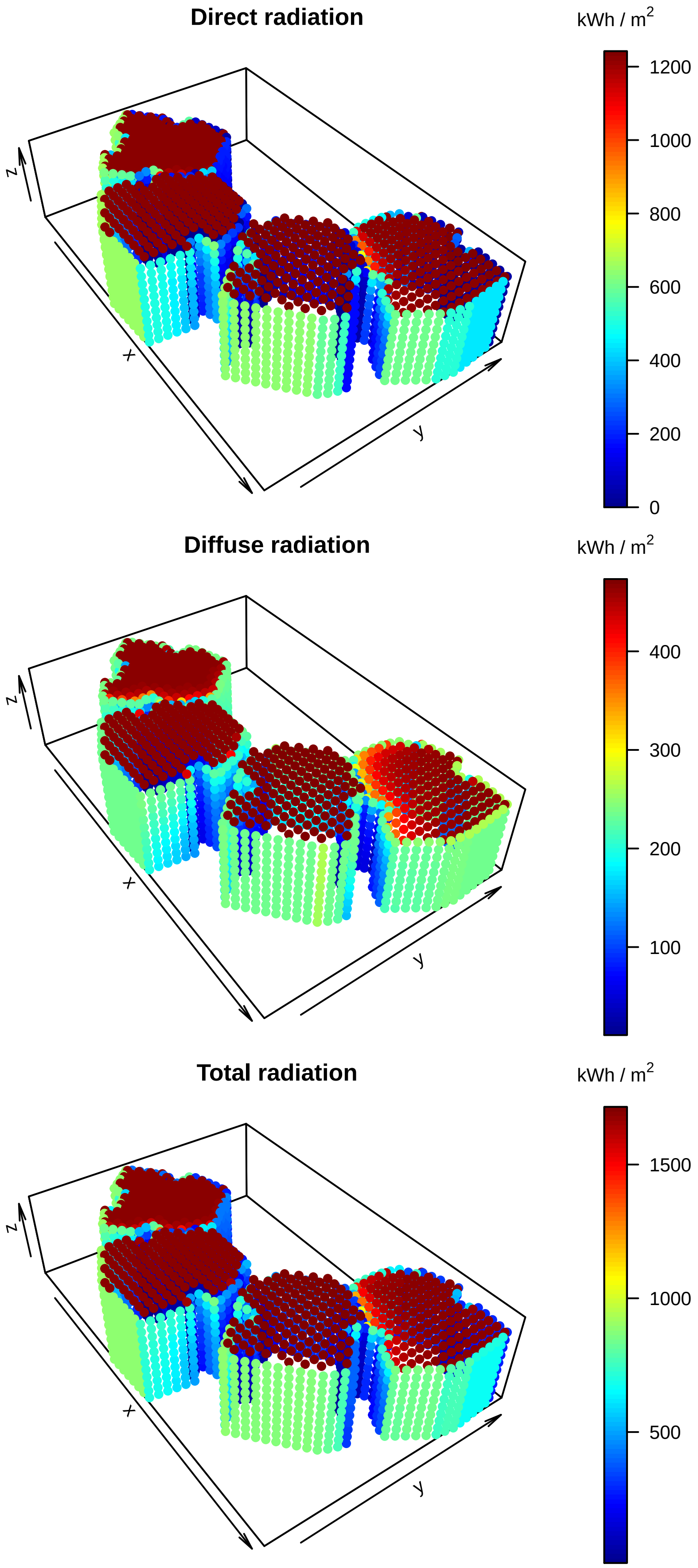


Figure 13: Annual direct, diffuse and total radiation estimates per grid point (kWh/m^2). Note that the Y-axis points to the north. Also note that the color scale range is different in each panel.

Discussion

The **shadow** package introduces a simple geometric model for shadow-related calculations in an urban environment. Specifically, the package provides functions for calculating shadow height, shadow footprint and SVF. The latter can be combined with TMY data to estimate insolation on built surfaces. It is, to the best of our knowledge, the only R package aimed at shadow calculations in a *vector-based* representation of the urban environment. It should be noted that the **insol** package provides similar functionality for a *raster-based* environment, but the latter is more suitable for modelling large-scale natural environments rather than detailed urban landscapes.

The unique aspect of our approach is that calculations are based on a vector layer of polygons extruded to a given height, known as 2.5D, such as building footprints with a height attribute. The vector-based 2.5D approach has several advantages over the two commonly used alternative ones: vector-based 3D and raster-based models. Firstly, the availability of 2.5D input data is much greater compared to both specialized 3D models and high-resolution raster surfaces. Building layers for entire cities are generally available from various sources, ranging from local municipality GIS systems to global crowd-sourced datasets (e.g. OpenStreetMap) (Haklay and Weber, 2008). Secondly, processing does not require closed-source software, or interoperability with complex specialized software, as opposed to working with 3D models. Thirdly, results are easily associated back to the respective urban elements such as buildings, parks, roofs facades, etc., as well as their attributes, via a spatial join operation (e.g. using function `over` in R package **sp**). For example, we can easily determine which building is responsible for shading the green park in the above shadow footprint example (Figure 10). This is unlike a raster-based approach, where the input is a continuous surface with no attributes, thus having no natural association to individual urban elements or objects.

However, it should be noted that the 2.5D vector-based approach requires several assumptions and has some limitations. When the assumptions do not hold, results may be less accurate compared to the above-mentioned alternative approaches. For example, it is impossible to represent geometric shapes that are not a simple extrusion in 2.5D (though, as mentioned above, urban surveys providing such detailed data are not typically available). An ellipsoid tree, a bridge with empty space underneath, a balcony extruding outwards from a building facade, etc., can only be represented with a polyhedral surface in a full vector-based 3D environment (Gröger and Plümer, 2012; Biljecki et al., 2016). Recently, classes for representing true-3D urban elements, such as the Simple Feature type `POLYHEDRALSURFACE`, have been implemented in R package **sf** (Pebesma, 2018). However, functions for working with those classes, such as calculating 3D-intersection, are still lacking. Implementing such functions in R could bring new urban analysis capabilities to the R environment in the future, in which solar analysis of 3D city models probably comprises a major use case (Biljecki et al., 2015).

It should also be noted that a vector-based calculation may be generally slower than a raster based calculation. This becomes important when the study area is very large. Though the present algorithms can be optimized to some extent, they probably cannot compete with raster-based calculations where sun ray intersections can be computed using fast ray-tracing algorithms based on matrix input (Amanatides et al., 1987), as opposed to computationally intensive search for intersections between a line and a polygonal layer in a vector-based environment. For example, calculating the SVF surface shown in Figure 12 requires processing 72 angular sections × 3,780 raster cells = 272,160 SVF calculations, which takes about 7.3 minutes using five cores on an ordinary desktop computer (Intel® Core™ i7-6700 CPU @ 3.40GHz × 8). The annual radiation estimate shown in Figure 13 however takes about 3.9 hours to calculate, as it requires SVF calculation for 2,693 grid points, as well as 727 ground locations × 8,760 hours = 6,368,520 shadow height calculations.

To summarize, the **shadow** package can be used to calculate shadow, SVF and solar radiation in an urban environment using widely available polygonal building data, inside the R environment (e.g. Vulkan et al., 2018). Potential use cases include urban environment applications such as evaluation of micro-climatic influence for urban planning, studying urban well-being (e.g. climatic comfort) and estimating photovoltaic energy production potential.

Acknowledgements

The **shadow** package was developed as part of a study funded by the Israel Ministry of National Infrastructures, Energy and Water Resources under research grant # 021-11-215.

The authors would like thank the Editor and an anonymous reviewer for the review of this article and for the thoughtful comments.

Bibliography

J. Amanatides, A. Woo, and others. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987. [p306]

B. Beckers. *Solar Energy at Urban Scale*. John Wiley & Sons, 2013. [p290]

F. Biljecki, J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin. Applications of 3D city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4):2842–2889, 2015. URL `https://doi.org/10.3390/ijgi4042842`. [p287, 306]

F. Biljecki, H. Ledoux, and J. Stoter. An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems*, 59:25–37, 2016. URL `https://doi.org/10.1016/j.compenvurbsys.2016.04.005`. [p306]

R. Bivand and N. Lewin-Koh. *Maptools: Tools for Reading and Handling Spatial Objects*, 2017. URL `https://CRAN.R-project.org/package=maptools`. R package version 0.9-2. [p294]

R. Bivand and C. Rundel. *Rgeos: Interface to Geometry Engine - Open Source ('GEOS')*, 2017. URL `https://CRAN.R-project.org/package=rgeos`. R package version 0.3-26. [p294]

R. S. Bivand, E. Pebesma, and V. Gomez-Rubio. *Applied Spatial Data Analysis with R, Second Edition*. Springer-Verlag, 2013. URL `http://www.asdar-book.org/`. [p288]

F. Bourbia and F. Boucheriba. Impact of street design on urban microclimate for semi arid climate (Constantine). *Renewable Energy*, 35(2):343–347, 2010. URL `https://doi.org/10.1016/j.renene.2009.07.017`. [p287]

J. G. Corripio. *Insol: Solar Radiation*, 2014. URL `https://CRAN.R-project.org/package=insol`. R package version 1.1.1. [p287]

M. Dorman. *Shadow: Geometric Shadow Calculations*, 2019. URL `https://CRAN.R-project.org/package=shadow`. R package version 0.6.0. [p288]

E. Erell, D. Pearlmutter, and T. Williamson. *Urban Microclimate: Designing the Spaces between Buildings*. Earthscan/James & James Science Publishers, 2011. URL `https://doi.org/10.4324/9781849775397`. [p290]

ESRI. *ArcGIS Desktop: Release 10.5*. Environmental Systems Research Institute, CA, 2017. URL `https://www.arcgis.com`. [p287]

S. Freitas, C. Catita, P. Redweik, and M. C. Brito. Modelling solar potential in the urban environment: State-of-the-art review. *Renewable and Sustainable Energy Reviews*, 41:915–931, 2015. URL `https://doi.org/10.1016/j.rser.2014.08.060`. [p287, 292]

P. Fu and P. M. Rich. Design and implementation of the Solar Analyst: An ArcView extension for modeling solar radiation at landscape scales. In *Proceedings of the Nineteenth Annual ESRI User Conference*, pages 1–31, 1999. [p287]

B. Givoni. *Climate Considerations in Building and Urban Design*. John Wiley & Sons, 1998. [p291]

Google. *SketchUp: Release 17*. Trimble Inc., CA, 2017. URL `https://www.sketchup.com/`. [p287]

GRASS Development Team. *Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.2*. Open Source Geospatial Foundation, 2017. URL `http://grass.osgeo.org`. [p287]

C. Grimmond, S. Potter, H. Zutter, and C. Souch. Rapid methods to estimate sky-view factors applied to urban areas. *International Journal of Climatology*, 21(7):903–913, 2001. URL `https://doi.org/10.1002/joc.659`. [p290]

G. Gröger and L. Plümer. CityGML–Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71:12–33, 2012. URL `https://doi.org/10.1016/j.isprsjprs.2012.04.004`. [p288, 306]

T. Gál and J. Unger. A new software tool for SVF calculations using building and tree-crown databases. *Urban Climate*, 10:594–606, 2014. URL `https://doi.org/10.1016/j.uclim.2014.05.004`. [p290]

M. Haklay and P. Weber. OpenStreetMap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008. URL `https://doi.org/10.1109/MPRV.2008.80`. [p288, 306]

R. J. Hijmans. *Raster: Geographic Data Analysis and Modeling*, 2017. URL https://CRAN.R-project.org/package=raster. R package version 2.6-7. [p294]

J. Hofierka and M. Suri. The solar radiation model for Open source GIS: Implementation and applications. In *Proceedings of the Open Source GIS-GRASS Users Conference*, volume 2002, pages 51–70, 2002. [p287]

J. Hofierka and M. Zlocha. A new 3-D solar radiation model for 3-D city models. *Transactions in GIS*, 16(5):681–690, 2012. URL https://doi.org/10.1111/j.1467-9671.2012.01337.x. [p288]

L. Kumar, A. K. Skidmore, and E. Knowles. Modelling topographic variation in solar radiation in a GIS environment. *International Journal of Geographical Information Science*, 11(5):475–497, 1997. URL https://doi.org/10.1080/136588197242266. [p287]

B. W. Lewis. *Threejs: Interactive 3D Scatter Plots, Networks and Globes*, 2017. URL https://CRAN.R-project.org/package=threejs. R package version 0.3.1. [p294]

J. Liang, J. Gong, J. Zhou, A. N. Ibrahim, and M. Li. An open-source 3D solar radiation model integrated with a 3D Geographic Information System. *Environmental Modelling & Software*, 64:94–101, 2015. URL https://doi.org/10.1016/j.envsoft.2014.11.019. [p287]

F. Lindberg, C. S. B. Grimmond, A. Gabey, B. Huang, C. W. Kent, T. Sun, N. E. Theeuwes, L. Järvi, H. C. Ward, I. Capel-Timms, and others. Urban Multi-Scale Environmental Predictor (UMEP): An integrated tool for city-based climate services. *Environmental Modelling & Software*, 99:70–87, 2018. URL https://doi.org/10.1016/j.envsoft.2017.09.020. [p287]

E. Pebesma. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1):439–446, 2018. URL https://journal.r-project.org/archive/2018/RJ-2018-009/index.html. [p306]

E. Pebesma and R. S. Bivand. Classes and methods for spatial data: The sp package. *R news*, 5(2):9–13, 2005. URL https://CRAN.R-project.org/doc/Rnews/. [p288]

S. Pusat, İsmail Ekmekçi, and M. T. Akkoyunlu. Generation of typical meteorological year for different climates of Turkey. *Renewable Energy*, 75:144–151, 2015. URL https://doi.org/10.1016/j.renene.2014.09.039. [p293]

QGIS Development Team. *QGIS Geographic Information System*. Open Source Geospatial Foundation, 2017. URL http://qgis.osgeo.org. [p287]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL https://www.R-project.org/. [p297]

C. Ratti and P. Richens. Raster analysis of urban form. *Environment and Planning B: Planning and Design*, 31(2):297–309, 2004. URL https://doi.org/10.1068/b2665. [p287]

P. Redweik, C. Catita, and M. Brito. Solar energy potential on roofs and facades in an urban landscape. *Solar Energy*, 97:332–341, 2013. URL https://doi.org/10.1016/j.solener.2013.08.036. [p287, 288]

K. Soetaert. *plot3D: Plotting Multi-Dimensional Data*, 2017. URL https://CRAN.R-project.org/package=plot3D. R package version 1.1.1. [p299]

G. Van Rossum and F. L. Drake. *The Python Language Reference Manual*. Network Theory Ltd., 2011. URL http://www.network-theory.co.uk/python/language/. [p287]

A. Vulkan, I. Kloog, M. Dorman, and E. Erell. Modelling the potential for PV installation in residential buildings in dense urban areas. *Energy and Buildings*, 2018. URL https://doi.org/10.1016/j.enbuild.2018.03.052. [p306]

M. Weisthal. Assessment of potential energy savings in israel through climate-aware residential building design. Master's thesis, Ben-Gurion University of the Negev, Jacob Blaustein Institutes for Desert Research, Albert Katz International School for Desert Studies, 2014. URL http://aranne5.bgu.ac.il/others/WeisthalMorel.pdf. [p289]

Michael Dorman
BGU
Department of Geography and Environmental Development

Ben-Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel
dorman@post.bgu.ac.il

Evyatar Erell
BGU
The Jacob Blaustein Institutes for Desert Research
and the Department of Geography and Environmental Development
Ben-Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel

Adi Vulkan
BGU
Department of Geography and Environmental Development
Ben-Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel

Itai Kloog
BGU
Department of Geography and Environmental Development
Ben-Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel

Integration of networks and pathways with StarBioTrek package

by Claudia Cava and Isabella Castiglioni

Abstract High-throughput genomic technologies bring to light a comprehensive hallmark of molecular changes of a disease. It is increasingly evident that genes are not isolated from each other and the identification of a gene signature can only partially elucidate the de-regulated biological functions in a disease. The comprehension of how groups of genes (pathways) are related to each other (pathway-cross talk) could explain biological mechanisms causing diseases. Biological pathways are important tools to identify gene interactions and decrease the large number of genes to be studied by partitioning them into smaller groups. Furthermore, recent scientific studies have demonstrated that an integration of pathways and networks, instead of a single component of the pathway or a single network, could lead to a deeper understanding of the pathology.

StarBioTrek is an R package for the integration of biological pathways and networks which provides a series of functions to support the user in their analyses. In particular, it implements algorithms to identify pathways cross-talk networks and gene network drivers in pathways. It is available as open source and open development software in the Bioconductor platform.

Introduction

In recent years new genomic technologies have made possible to define new marker gene signatures (Desmedt et al., 2009; Parker et al., 2009; Cava et al., 2014b). However, gene expression-based signatures present some constraints because they do not consider metabolic role of the genes and are affected by genetic heterogeneity across patient cohorts (Cava et al., 2015; Donato et al., 2013).

Pathway analysis can help the researchers in the identification of the biological roles of candidate genes exceeding these limitions (Folger et al., 2011). Indeed, considering the activity of entire biological pathways rather than the expression levels of individual genes can charactherize the whole tissue. In particular, there are several methods in computations and data used to perform the pathway analyses. They can be characterized in two different levels: gene-sets and pathway topology (García-Campos et al., 2015). Indeed, the existing tools integrating pathway data can be grouped into these groups based on the pathway definition.

In the first group we can include the tools that are based on gene sets definition as simple lists of biological molecules, in which the listed components share a common biological role. In this group, for example, we can include CoRegNet and Gene Set Enrichment Analysis (GSEA). CoRegNet reconstructs a co-regulatory network from gene expression profiles integrating, also, regulatory interactions, such as transcription factor binding site and ChIP data, presenting some analyses to identify master regulators of a given set of genes (Nicolle et al., 2015). One of the first and most popular methods is GSEA (Subramanian et al., 2005) that uses a list of ranked genes based on their differential gene expression between two labels. It then evaluates their distribution on a priori defined set of genes, thus generating an enrichment score (ES) for each set of genes.

In contrast, tools based on pathway topology do not only contain the components of a pathway but also describe the interactions between them. However, these methods still analyze the pathways as independent from each other and not considering the influence that a pathway can exert over another. In this second group we can include analysis methods that take into account the topological structure of a pathway, such as **NetPathMiner**, **ToPASeq**, and **XGR**. **NetPathMiner** (Mohamed et al., 2014) implements methods for the construction of various types of genome scale networks for network path mining. It generates a network representation from a pathway file supporting metabolic networks. Since the network is generated, the network edges can be weighted using gene expression data (e.g., Pearson correlation). Using machine learning methods and Markov mixture models, the pathways can be classified or clustered based on their association with a response label. The **ToPASeq** package implements seven different methods covering broad aspects for topology-based pathway analysis of RNA-seq data (Ihnatova and Budinska, 2015). With respect to other tools, **XGR** (Fang et al., 2016) is designed for enhanced interpretation of genomic data generating also SNP-modulated gene networks and pathways. However, compared to our tool, the others are not focused on the pathway cross-talk analyses.

In line with this scenario given the few methods focused on the pathway cross-talk network, the development of new methodologies to measure pathway activity and cross-talk among pathways integrating also the information of networks and gene expression data (e.g., TCGA data) could lead to a deeper knowledge of the pathology.

Furthermore, functional pathway representation attributes the same functional significance to each gene without considering the impact of gene interactions in performing that function. What kinds of interactions are there among genes in functional pathways? Specifically, biological system interactions are composed of multiple layers of dynamic interaction networks (Cantini et al., 2015). These dynamic networks can be decomposed, for example, into: co-expression, physical, co-localization, genetic, pathway, and shared protein domains.

We developed a series of algorithms (see (Cava et al., 2018; Colaprico et al., 2015; Cava et al., 2016)), implemented in **StarBioTrek** package able to work on all levels of the pathway analysis.

Starting from the gene expression data of two groups of samples (e.g., normal vs. disease), such algorithms aim at building a pathway cross-talk model by attributing a score for each pairwise pathway. Several scores are implemented in the tool using the gene expression levels inside the pathways. The interacting pathways are filtered considering pathways that are able to classify better the two groups of samples. In addition, the genes inside the pathways can be weighted defining key network drivers in the pathways as those gene drivers that are highly connected in biological networks.

In summary, **StarBioTrek** package proposes an approach that integrates knowledge on the functional pathways and multiple gene-gene (protein-protein) interactions into gene selection algorithms. The challenge is to identify more stable biomarker signatures, which are also more easily interpretable from a biological perspective. The integration of biological networks and pathways can also give further hypotheses of the mechanisms of driver genes.

Package organization

StarBioTrek makes accessible data of biological pathways and networks in order to perform analyses without having to navigate and access different web-based databases, without the need to download data, and by integrating and locally processing the full data sets in a short time. Specifically, it allows the users to: i) query and download biological pathways and networks from several repositories such as KEGG, Reactome and GeneMANIA(Zuberi et al., 2013; Cava et al., 2017; Franz et al., 2018) importing several functions from graphite (Sales et al., 2012), and harmonize annotations for genes and proteins (query/ download/ annotation harmonization); (ii) integrate pathways and biological networks with a series of implemented algorithms.

Get data

Pathway and network data

The functions of **StarBioTrek** import a large amount of data (e.g., biological pathways and networks).

Specifically, the function `pathwayDatabases` can easily query some features of interest of the user such as species or specific pathway database from graphite (Sales et al., 2012). Then, the function `GetData` imports the selected data.

```
> library(graphite)
> pathwayDatabases()
  species database
1      athaliana     kegg
2      athaliana pathbank
3      athaliana reactome
4        btaurus     kegg
5        btaurus reactome
6       celegans     kegg
7       celegans reactome
8    cfamiliaris     kegg
9    cfamiliaris reactome
10 dmelanogaster     kegg
11 dmelanogaster reactome
12        drerio     kegg
13        drerio reactome
14         ecoli     kegg
15         ecoli pathbank
16       ggallus     kegg
17       ggallus reactome
```

```
18      hsapiens biocarta
19      hsapiens humancyc
20      hsapiens     kegg
21      hsapiens      nci
22      hsapiens  panther
23      hsapiens pathbank
24      hsapiens pharmgkb
25      hsapiens reactome
26      hsapiens    smpdb
27     mmusculus     kegg
28     mmusculus pathbank
29     mmusculus reactome
30   rnorvegicus     kegg
31   rnorvegicus pathbank
32   rnorvegicus reactome
33   scerevisiae     kegg
34   scerevisiae pathbank
35   scerevisiae reactome
36       sscrofa     kegg
37       sscrofa reactome
38       xlaevis     kegg
> path <- GetData(species="hsapiens", pathwaydb="kegg")
```

Since the user selected the data of interest, the function GetPathData allows us to obtain a list of genes grouped by functional role:

```
> pathwayALLGENE <- GetPathData(path_ALL=path[1:3])
[1] "Downloading............  Glycolysis / Gluconeogenesis   1 of 3 pathways"
[1] "Downloading............  Citrate cycle (TCA cycle)   2 of 3 pathways"
[1] "Downloading............  Pentose phosphate pathway   3 of 3 pathways"
```

The function ConvertedIDgenes will converter the gene nomenclature (e.g., ENTREZ ID) to Gene Symbol.

```
> pathway <- ConvertedIDgenes(path_ALL=path[1:10])
```

The function getNETdata of **StarBioTrek** imports biological networks from GeneMANIA. The biological networks can be selected among physical interactions, co-localization, genetic interactions, pathways, and shared protein domain networks. Furthermore, it supports 9 species (Arabidopsis thaliana, Caenorhabditis elegans, Danio rerio, Drosophila melanogaster, Escherichia coli, Homo sapiens, Mus musculus, Rattus norvegicus, and Saccharomyces cerevisiae); for default it considers Homo sapiens. Specifically, the function call

```
> netw <- getNETdata(network="SHpd")
[1]"genemania.org/data/current/Homo_sapiens/Shared_protein_domains.INTERPRO.txt n.1 of 2"
[1]"genemania.org/data/current/Homo_sapiens/Shared_protein_domains.PFAM.txt n.2 of 2"
[1]"Preprocessing of the network n. 1 of 2"
[1]"Preprocessing of the network n. 2 of 2"
```

imports biological networks (i.e., shared protein domains interactions from INTERPRO and PFAM databases) for *Homo sapiens*. Otherwise, the user can select one of the 9 species or using the following parameters the user can select different network types: PHint for Physical interactions, COloc for Co-localization, GENint for Genetic interactions, PATH for Pathway, and SHpd for Shared protein domains. Finally, **StarBioTrek** provides the functions for the harmonization of gene nomenclature in the pathways and biological networks. Biological data are processed for downstream analyses mapping Ensembl Gene ID to gene symbols. Figure 1 shows an overview of network types supported by **StarBioTrek** with the function getNETdata.

Analysing pathways

Starting from a gene expression matrix (DataMatrix), **StarBioTrek** groups the gene expression levels according to their biological roles in pathways for each sample.

```
> listpathgene <- GE_matrix(DataMatrix=tumo[,1:2], genes.by.pathway=pathway[1:10])
> str(listpathgene)
```

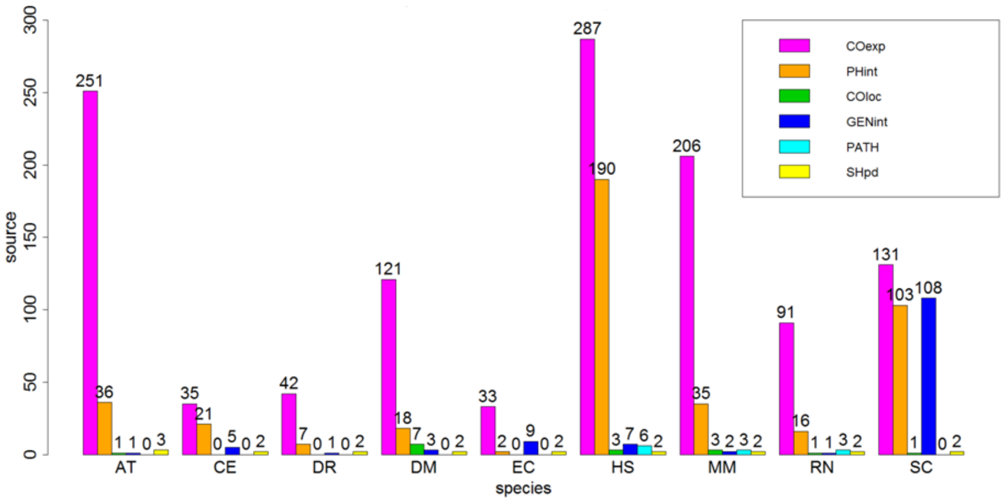


Figure 1: Network overview. Number of sources for network data. Barplot is divided by species: AT: *Arabidopsis thaliana*, CE: *Caenorhabditis elegans*, DR: *Danio rerio*, DM: *Drosophila melanogaster*, EC: *Escherichia coli*, HS: *Homo sapiens*, MM: *Mus musculus*, RN: *Rattus norvegicus*, SC: *Saccharomyces cerevisiae*, and by network type: COexp: Co-expression, PHint: Physical interactions, COloc: Co-localization, GENint: Genetic interactions, PATH: Pathway, SHpd: Shared protein domains

```
List of 2
 $ Cell_cycle           :'data.frame':        114 obs. of  2 variables:
  ..$ TCGA-E9-A1RC-01A: num [1:114] 4218 695 4231 7029 1211 ...
  ..$ TCGA-BH-A0B1-01A: num [1:114] 3273 692 6733 6468 1290 ...
 $ p53_signaling pathway:'data.frame':        64 obs. of  2 variables:
  ..$ TCGA-E9-A1RC-01A: num [1:64] 989 1614 1592 3456 900 ...
  ..$ TCGA-BH-A0B1-01A: num [1:64] 816 1274 1770 3190 405 ...
```

This function allows the user to have in a short time the gene expression levels grouped by pathways.

Pathway summary indexes

As described in Cava et al. (2014a) there are different measures to summarize the expression levels of each pathway, such as the mean:

```
> score_mean <- average(pathwayexpsubset=listpathgene)
```

or standard deviation:

```
> score_stdev <- stdv(gslist=listpathgene)
```

Dissimilarity distances: Pathway cross-talk indexes

Dissimilarity distances have been proved useful in many application fields. Recent studies (Cava et al., 2013, 2014c) used with success dissimilarity representation among patients, considering the expression levels of individual genes. To our knowledge, dissimilarity representation is not used in pathway-based expression profiles. Our goal is to give a dissimilarity representation, which can express, through a function D(x,y), the dissimilarity between two pathways x and y, such as Euclidean distance between pairs of pathways:

```
> scoreeucdistat <- eucdistcrtlk(dataFilt=Data_CANCER_normUQ_fil,
                                 pathway_exp=pathway[1:10])
```

or discriminating score (Colaprico et al., 2015):

```
> crosstalkstdv <- dsscorecrtlk(dataFilt=Data_CANCER_normUQ_fil,
                                pathway_exp=pathway[1:10])
```

Integration data

Integration between pathways and networks from GeneMANIA

Biological pathways can involve a large number of genes that are not equivocally relevant for a functional role in the cell. Therefore, the integration of network and pathway-based methods can boost the power to identify the key genes in the pathways.

The function takes as arguments: a list of pathways as obtained by the function `ConvertedIDgenes` and the networks as obtained by the function `getNETdata`.

```
> listanetwork <- pathnet(genes.by.pathway=pathway[1:10], data=netw)
```

It creates a network of interacting genes for each pathway. The output of the function is a selection of interacting genes according to the network N in a pathway P, namely a list with two columns where on the same row there are the two interacting genes.

The function `listpathnet` takes as inputs the output obtained by the function `pathnet` and the pathways as obtained by the function `ConvertedIDgenes`:

```
> listpath <- listpathnet(lista_net=listanetwork, pathway=pathway[1:10])
```

creating a list of vectors for each pathway containing only genes that have at least one interaction with other genes belonging to the pathway.

Integration between pathways and networks from protein-protein interaction

The function `GetPathNet` allows us to obtain a list of interacting genes (protein-protein interactions from **graphite** package) for each pathway:

```
> pathwaynet <- GetPathNet(path_ALL=path[1:3])
```

using as its argument the output obtained by `GetData`.

Analyzing networks and pathways: implemented algorithms

Pathway cross-talk network

The first algorithm implemented in **StarBioTrek** explores a pathway cross-talk network from gene expression data to better understand the underlying pathological mechanism. The algorithm generates a network of pathways that shows a different behavior between two groups of samples (i.e., normal vs. disease).

Specifically,

```
# get pathways from KEGG database
path <- GetData(species="hsapiens", pathwaydb="kegg")
pathway <- ConvertedIDgenes(path_ALL=path)

# create a measure of pathway cross-talk (i.e., Euclidean distance) between pairwise
# of pathways starting from gene expression data (i.e.TCGA) with in the columns the
# samples and in the rows the genes
scoreeucdistat <- eucdistcrtlk(dataFilt=Data_CANCER_normUQ_fil, pathway=pathway)


# split samples' TCGA ID into normal and tumor groups
tumo <- SelectedSample(Dataset=Data_CANCER_normUQ_fil, typesample="tumour")
norm <- SelectedSample(Dataset=Data_CANCER_normUQ_fil, typesample="normal")

# divide the dataset in 60/100 for training and 40/100 for testing
nf <- 60

# a support vector machine is applied
res_class <- svm_classification(TCGA_matrix=scoreeucdistat[1:10,], nfs=nf,
                                normal=colnames(norm), tumour=colnames(tumo))
```

Since the AUC values are obtained for each pair of pathways, they can be ranked in order to obtain the pathway cross-talk interactions able to classify the two classes (i.e. normal vs. tumor samples) with the best performance. Such selection can be done considering AUC values:

```
cutoff=0.80
er <- res_class[res_class[,1]>cutoff, ]
```

The outputs are the only pathway interactions that are obtained with AUC values > 0.80. The implemented algorithm was used in (Colaprico et al., 2015) and (Cava et al., 2016) to screen pathway cross-talk associated to breast cancer.

The pseudocode of the implemented algorithm is summarized below.

```
Data: 1) a matrix of gene expression data (TCGA data). The samples are in the columns
      and the genes in the rows; 2) a matrix where the pathways are in the columns and the
      genes inside the pathways are in the rows
Result: pathway interactions that are able to classify two groups of samples with the best
        performances
Being $a$ and $b$ two pathways in a set of pathways $P$;
for all nodes(a,b) in P do
    a score distance between the nodes $a$ and $b$;
    if $AUC > cut\text{-}off$ then
        keep ($a$,$b$) as edge;
    else
        remove ($a$,$b$) as edge;
    end
end
```

Algorithm 1: Algorithm implemented in (Colaprico et al., 2015) and (Cava et al., 2016)

Driver genes for each pathway

Here, we propose an algorithm for the integrative analysis of networks and pathways. Our method is inspired on a well-validated method (the GANPA/LEGO) (Fang et al., 2012; Dong et al., 2016), based on the hypothesis that if one gene is functionally connected in the pathway with more genes than those expected (according to the functional networks), has a key role in that pathway. The algorithm, an extension of the GANPA/LEGO method, defines driver genes in a pathway if they are highly connected in a biological network.

The function

```
IPPI(patha=pathway_matrix[,1:10], netwa=netw_IPPI)
```

is used to identify driver genes for each pathway. The inputs of the function are pathways and network data. It calculates the degree centrality values of genes inside the network and the degree centrality of genes inside pathways.

The pseudocode of the implemented algorithm is summarized below.

```
Data: 1) a matrix where the pathways are in the columns and the genes inside the pathways
      are in the rows; 2) a data frame where the nodes are presented in the columns and the
      rows represent the edges
Result: a list of genes with high degree centrality for each pathway
Being $(i \in N)$ & $(i \in P)$ where $P$ is a vector containing the genes inside a pathway of size $k$
  and $N$ is an indirect graph of size $m$;
for all nodes $i$ in $N$ do
    calculate the degree centrality $d_{iN}$;
end
for all nodes $i$ in $P$ do
    calculate the degree centrality $d_{iP}$, being the neighbors of $i$, $\mathrm{i}_{ng} \in P$;
end
Calculate degree centrality expected $d_{iE}$ in $P$
if $d_{iE} < d_{iP} / k_p$ then
    $i \longleftarrow$potential gene drivers of $P$;
else
    $i \longleftarrow i + 1$;
end
```

Algorithm 2: Algorithm implemented in (Cava et al., 2018)

In the first step, given the gene i within the network N with m genes, the function computes the degree centrality d_{iN} as the number of neighbor genes belonging to N to which the gene i is directly connected.

In the second step, given gene i within the pathway P with k genes, the function then computes the degree centrality d_{iP} considering only the relations among gene i and the other genes in the networks belonging to pathway P. Overall, by combining the information of the network N within the pathway P, is obtained a selection of interacting genes according to the network N.

Then, the function computes the degree centrality expected d_{iE} by supposing equal probability for the existence of edges between nodes ($d_{iN}/m = d_{iE}/k$). Thus, $d_{iE} = d_{iN}\ xk/m$.

The function characterizes a gene as a 'network driver' in the pathway P, when d_{iP} of involving gene, normalized to the size of the pathway (k), is higher than d_{iE}, $d_{iP}/k > d_{iE}$.

The speculation is that if one gene is functionally linked (according to the functional network) with more genes in the pathway than expected, its function is central in that pathway.

The function IPPI was used in (Cava et al., 2018) to find driver genes in the pathways that are also de-regulated in a pan-cancer analysis.

Visualization

StarBioTrek presents several functions for the preparation to the visualization of gene-gene interactions and pathway cross-talk using the **qgraph** package (Epskamp et al., 2012). The function plotcrosstalk prepares the data:

```
> formatplot <- plotcrosstalk(pathway_plot=pathway[1:6],gs_expre=tumo)
```

It computes a Pearson correlation between the genes (according to a gene expression matrix, such as tumor) in which each gene is grouped in a gene set given by the user (e.g., pathway). Each gene is presented in a gene set if it is involved univocally in that gene set.

The functions of **qgraph**

```
> library(qgraph)
> qgraph(formatplot[[1]], minimum = 0.25, cut = 0.6, vsize = 5, groups = formatplot[[2]],
         legend = TRUE, borders = FALSE, layoutScale=c(0.8,0.8))
```

and

```
> qgraph(formatplot[[1]], groups=formatplot[[2]], layout="spring", diag = FALSE,
  cut = 0.6, legend.cex = 0.5, vsize = 6, layoutScale=c(0.8,0.8))
```

show the network with different layouts. The graphical output of the functions are presented in the Figure 2 and Figure 3. The color of interactions indicates the type of correlation: green edges are positive correlations and red edges are negative correlations. The thickness of the edge is proportional to the strength of correlation.

The outputs of the functions that compute the pairwise distance metrics can be easily used with heatmap plotting libraries such as heatmap or pheatmap as reported in the Figure 4.

Furthermore, the function circleplot of **StarBioTrek** implemented using the functions of **GOplot** (Walter et al., 2015) provides a visualization of driver genes (with a score indicating the role of genes in that pathway), as reported in Figure 5.

```
> formatplot <- plotcrosstalk(pathway_plot=pathway[1:6], gs_expre=tumo)
> score <- runif(length(formatplot[[2]]), min=-10, max=+10)
> circleplot(preplot=formatplot, scoregene=score)
```

Case studies

In this section we will present two case studies for the usage of the **StarBioTrek** package. In particular, the first case study uses the first implemented algorithm reported above to identify pathway cross-talk network. The second case study uses the second implemented algorithm to identify gene drivers for each pathway.

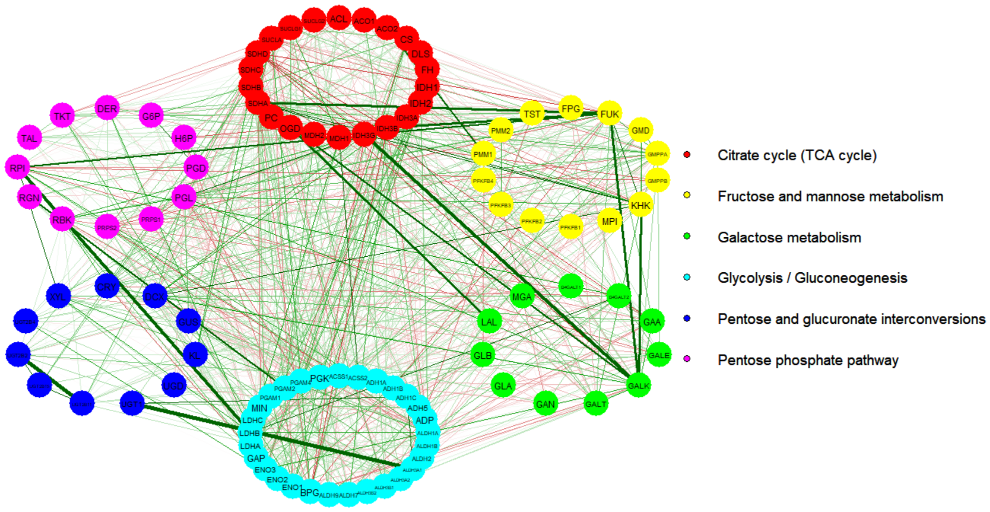

● Citrate cycle (TCA cycle)
● Fructose and mannose metabolism
● Galactose metabolism
● Glycolysis / Gluconeogenesis
● Pentose and glucuronate interconversions
● Pentose phosphate pathway

Figure 2: Graphical output of the function `plotcrosstalk` and `qgraph` with layout 1

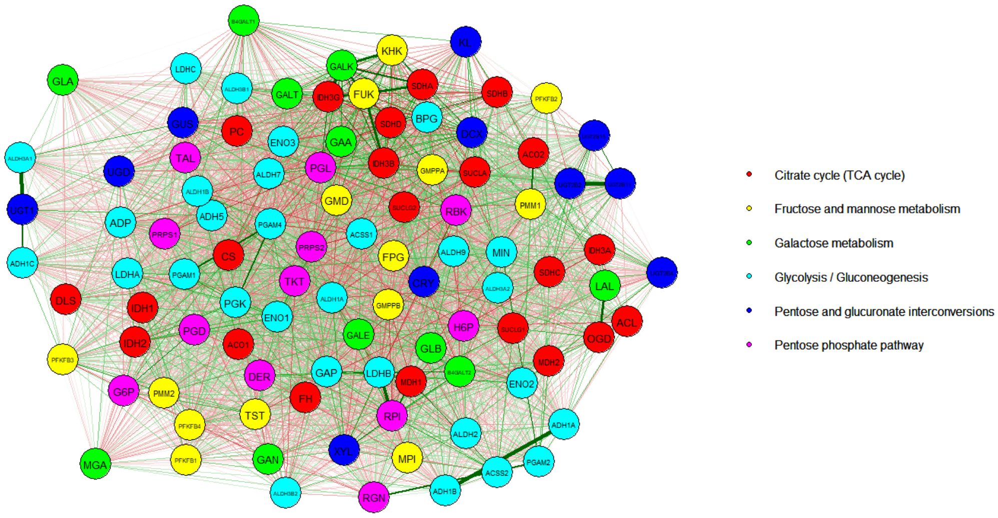

● Citrate cycle (TCA cycle)
● Fructose and mannose metabolism
● Galactose metabolism
● Glycolysis / Gluconeogenesis
● Pentose and glucuronate interconversions
● Pentose phosphate pathway

Figure 3: Graphical output of the function `plotcrosstalk` and `qgraph` with layout 2

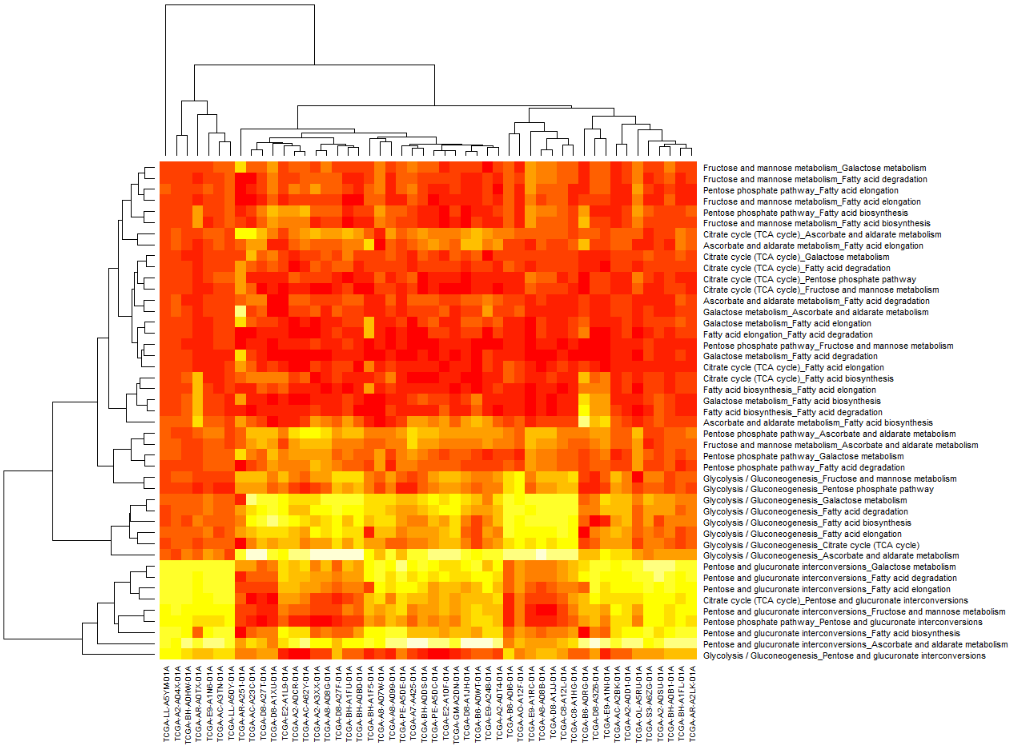


Figure 4: Heatmap of pathway cross-talk. Each row represent a distance metric between two pathways (the pathways are seperated by an underscore). The columns represent the samples.

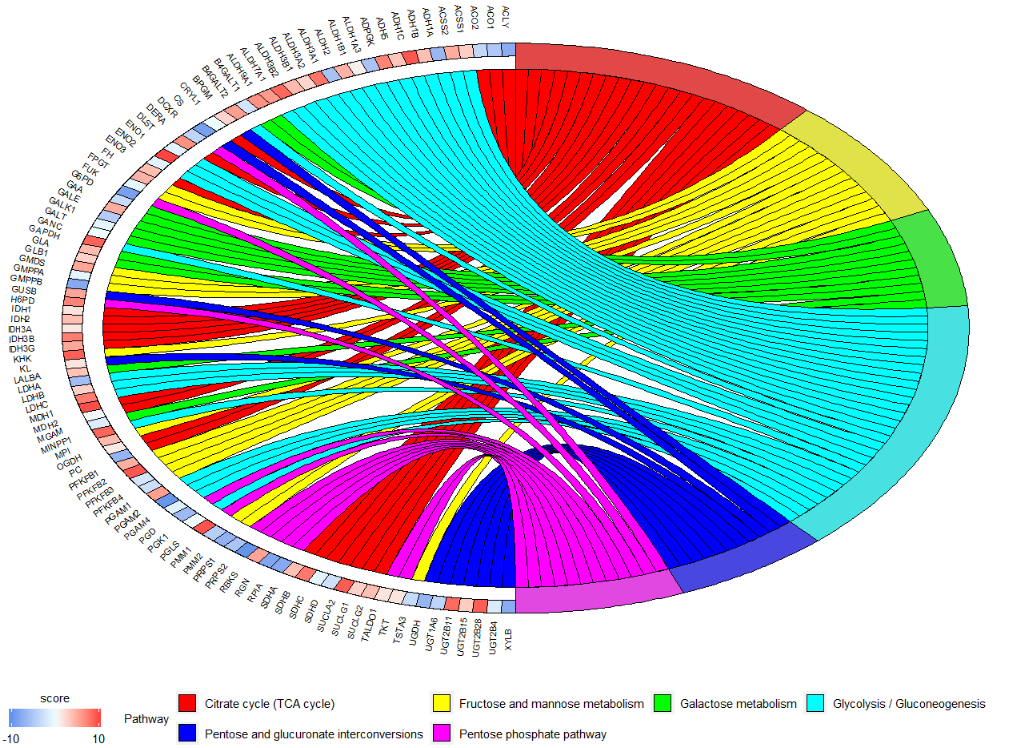


Figure 5: Circleplot of pathway cross-talk.The figure shows the relation between gene drivers and pathways. The pathways are represented with different colours. The intensity of colour of each block of genes is based on the score.

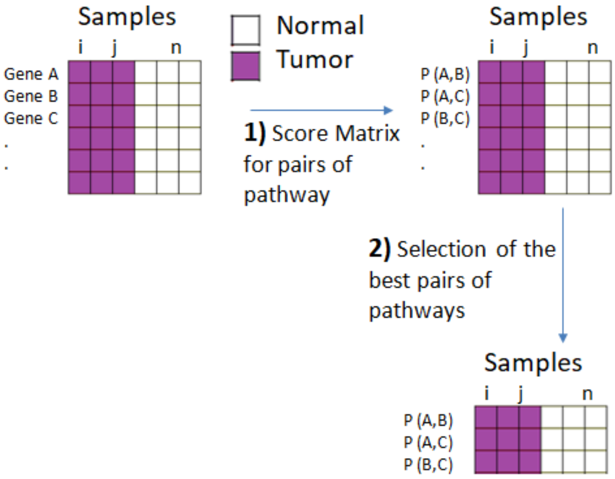


Figure 6: The computational approach. The matrix of gene expression data (samples in the columns and genes in the rows) is the input of our algorithm. The samples are grouped in two classes (e.g., normal vs. tumor). In the first step a matrix score is generated using all pairwise combinations of pathways. In the second step the score matrix is used as input for SVM classification. The pathway interactions with the best AUC performances are selected.

Pathway cross-talk network in breast cancer

Starting from gene expression data of breast cancer samples and normal samples we grouped 15243 genes in pathways according to their functional role in the cell. Pathway data were derived from the function call:

```
path <- GetData(species="hsapiens", pathwaydb="kegg")
pathway <- ConvertedIDgenes(path_ALL=path)
```

For each pair of pathways we calculated a discriminating score as a measure of cross-talk. This measure can be used considering e.g. the pathways enriched with differentially expressed genes.

```
crosstalkscore <- dsscorecrtlk(dataFilt=Data_CANCER_normUQ_fil,
                               pathway_exp=pathway[1:10])
```

Discriminating score is given by |M1-M2|/S1+S2 where M1 and M2 are means and S1 and S2 standard deviations of expression levels of genes in a pathway 1 and in a pathway 2. In order to identify the best pathways for breast cancer classification (breast cancer vs. normal) we implemented a Support Vector Machine. We divided the original dataset in training data set (60/100) and the rest of original data in the testing set (40/100). In order to validate the classifier, we used a k-fold cross-validation ($k = 10$) obtaining Area Under the Curve (AUC).

```
tumo <- SelectedSample(Dataset=Data_CANCER_normUQ_fil, typesample="tumour")
norm <- SelectedSample(Dataset=Data_CANCER_normUQ_fil, typesample="normal")
nf <- 60
res_class <- svm_classification(TCGA_matrix=crosstalkscore, nfs=nf,
                                normal=colnames(norm), tumour=colnames(tumo))
```

Ranking AUC values obtained we selected the pathway cross-talk network with the best AUC. The approach of the algorithm is shown in Figure 6.

Gene network drivers in pathways

In the second case study, we downloaded KEGG pathways

```
path <- GetData(species="hsapiens",pathwaydb="kegg")
pathway <- ConvertedIDgenes(path_ALL=path)
```

and network data for different network types from GeneMANIA

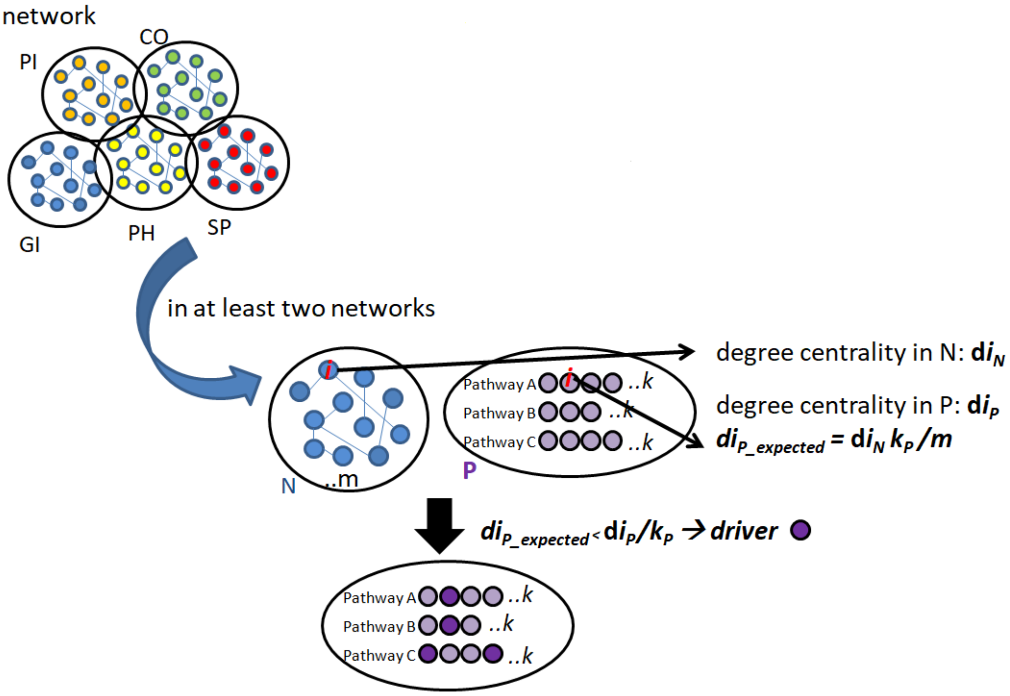

network
CO
PI
GI
PH
SP
in at least two networks
i
N
..m
Pathway A
Pathway B
Pathway C
..k
P
degree centrality in N: di_N
degree centrality in P: di_P
di_P_expected = di_N k_P /m
di_P_expected < di_P/k_P → driver
Pathway A
Pathway B
Pathway C
..k

Figure 7: The computational approach. The first step involves a network N (e.g. physical interaction) of size m and for each gene, i in N the algorithm calculates its degree centrality, DC (d_{iN}). The second step involves a set of functional pathways (e.g. pathway P) and for each gene i, the DC (d_{iP}) is calculated using the information on interacting genes from N. For the speculation of equal probability for existing edges between nodes, the algorithm computes the expected DC of gene i in the pathway P. If the DC observed for the gene i (d_{iP}) is higher than expected (d_{iP} expected), i could be a potential driver in the pathway P

```
# for Physical interactions
netw <- getNETdata(network="PHint")

# for Co-localization
netw <- getNETdata(network="COloc")

# for Genetic interactions
netw <- getNETdata(network="GENint")

# for Pathway interactions
netw <- getNETdata(network="PATH")

# for Shared_protein_domains
netw <- <getNETdata(network="SHpd")
```

We processed the data obtained by the function `getNETdata` in order to obtain a data format supported by the function `IPPI`. The function `IPPI` was applied for each of the 5 network types.

We obtained that genes with genetic interaction found the lowest number of potential gene network drivers. On the other hand, the network that includes proteins with shared protein domains found the highest number of potential driver genes. Finally, we defined a gene as a “network driver” in the pathway, when in at least two networks one gene is functionally connected in the pathway with more genes than those expected (according to the two networks).

The approach is shown in Figure 7.

Conclusions

We have described **StarBioTrek**, an R package for the integrative analysis of biological networks and pathways. The package supports the user during the import and data analysis of data. **StarBioTrek** implements two algorithms: i) the identification of gene network drivers in the pathways; ii) the building of pathway cross talk network.

Bibliography

L. Cantini, E. Medico, S. Fortunato, and M. Caselle. Detection of gene communities in multi-networks reveals cancer drivers. *Scientific reports*, 5:17386, 2015. URL `https://doi.org/10.1038/srep17386`. [p311]

C. Cava, I. Zoppis, M. Gariboldi, I. Castiglioni, G. Mauri, and M. Antoniotti. Copy–number alterations for tumor progression inference. In *Conference on Artificial Intelligence in Medicine in Europe*, pages 104–109. Springer, 2013. URL `https://doi.org/10.1007/978-3-642-38326-7_16`. [p313]

C. Cava, G. Bertoli, and I. Castiglioni. Pathway-based expression profile for breast cancer diagnoses. In *Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE*, pages 1151–1154. IEEE, 2014a. URL `https://doi.org/10.1109/embc.2014.6943799`. [p313]

C. Cava, G. Bertoli, M. Ripamonti, G. Mauri, I. Zoppis, P. A. Della Rosa, M. C. Gilardi, and I. Castiglioni. Integration of mRNA expression profile, copy number alterations, and microRNA expression levels in breast cancer to improve grade definition. *PloS one*, 9(5):e97681, 2014b. URL `https://doi.org/10.1371/journal.pone.0097681`. [p310]

C. Cava, I. Zoppis, M. Gariboldi, I. Castiglioni, G. Mauri, and M. Antoniotti. Combined analysis of chromosomal instabilities and gene expression for colon cancer progression inference. *Journal of Clinical Bioinformatics*, 4(1):2, 2014c. URL `https://doi.org/10.1186/2043-9113-4-2`. [p313]

C. Cava, G. Bertoli, and I. Castiglioni. Integrating genetics and epigenetics in breast cancer: Biological insights, experimental, computational methods and therapeutic potential. *BMC Systems Biology*, 9(1):62, 2015. URL `https://doi.org/10.1186/s12918-015-0211-x`. [p310]

C. Cava, A. Colaprico, G. Bertoli, G. Bontempi, G. Mauri, and I. Castiglioni. How Interacting Pathways Are Regulated by miRNAs in Breast Cancer Subtypes. *BMC Bioinformatics*, 17(12): 348, 2016. URL `https://doi.org/10.1186/s12859-016-1196-1`. [p311, 315]

C. Cava, A. Colaprico, G. Bertoli, A. Graudenzi, T. C. Silva, C. Olsen, H. Noushmehr, G. Bontempi, G. Mauri, and I. Castiglioni. SpidermiR: An R/Bioconductor package for integrative analysis with miRNA data. *International journal of molecular sciences*, 18(2):274, 2017. URL `https://doi.org/10.3390/ijms18020274`. [p311]

C. Cava, G. Bertoli, A. Colaprico, C. Olsen, G. Bontempi, and I. Castiglioni. Integration of multiple networks and pathways identifies cancer driver genes in pan-cancer analysis. *BMC Genomics*, 19 (1):25, 2018. URL `https://doi.org/10.1186/s12864-017-4423-x`. [p311, 315, 316]

A. Colaprico, C. Cava, G. Bertoli, G. Bontempi, and I. Castiglioni. Integrative Analysis with Monte Carlo Cross-Validation Reveals miRNAs Regulating Pathways Cross-Talk in Aggressive Breast Cancer. *BioMed Research International*, 2015(831314):17, 2015. [p311, 313, 315]

C. Desmedt, A. Giobbie-Hurder, P. Neven, R. Paridaens, M.-R. Christiaens, A. Smeets, F. Lallemand, B. Haibe-Kains, G. Viale, R. D. Gelber, and others. The Gene expression Grade Index: a potential predictor of relapse for endocrine-treated breast cancer patients in the BIG 1–98 trial. *BMC Medical Genomics*, 2(1):40, 2009. URL `https://doi.org/10.1186/1755-8794-2-40`. [p310]

M. Donato, Z. Xu, A. Tomoiaga, J. G. Granneman, R. G. MacKenzie, R. Bao, N. G. Than, P. H. Westfall, R. Romero, and S. Draghici. Analysis and correction of crosstalk effects in pathway analysis. *Genome research*, 23(11):1885–1893, 2013. URL `https://doi.org/10.1101/gr.153551.112`. [p310]

X. Dong, Y. Hao, X. Wang, and W. Tian. LEGO: a novel method for gene set over-representation analysis by incorporating network-based gene weights. *Scientific Reports*, 6:18871, 2016. URL `https://doi.org/10.1038/srep18871`. [p315]

S. Epskamp, A. O. Cramer, L. J. Waldorp, V. D. Schmittmann, D. Borsboom, and others. qgraph: Network Visualizations of Relationships in Psychometric Data. *Journal of Statistical Software*, 48 (4):1–18, 2012. URL `https://doi.org/10.18637/jss.v048.i04`. [p316]

H. Fang, B. Knezevic, K. L. Burnham, and J. C. Knight. XGR Software for Enhanced Interpretation of Genomic Summary Data, Illustrated by Application to Immunological Traits. *Genome medicine*, 8(1):129, 2016. URL https://doi.org/10.1186/s13073-016-0384-y. [p310]

Z. Fang, W. Tian, and H. Ji. A network-based gene-weighting approach for pathway analysis. *Cell research*, 22(3):565, 2012. URL https://doi.org/10.1038/cr.2011.149. [p315]

O. Folger, L. Jerby, C. Frezza, E. Gottlieb, E. Ruppin, and T. Shlomi. Predicting selective drug targets in cancer through metabolic networks. *Molecular systems biology*, 7(1):501, 2011. URL https://doi.org/10.1038/msb.2011.35. [p310]

M. Franz, H. Rodriguez, C. Lopes, K. Zuberi, J. Montojo, G. D. Bader, and Q. Morris. GeneMANIA Update 2018. *Nucleic acids research*, 46(W1):W60–W64, 2018. [p311]

M. A. García-Campos, J. Espinal-Enríquez, and E. Hernández-Lemus. Pathway analysis: State of the art. *Frontiers in Physiology*, 6:383, 2015. URL https://doi.org/10.3389/fphys.2015.00383. [p310]

I. Ihnatova and E. Budinska. ToPASeq: An R package for topology-based pathway analysis of microarray and RNA-Seq data. *BMC bioinformatics*, 16(1):350, 2015. URL https://doi.org/10.1186/s12859-015-0763-1. [p310]

A. Mohamed, T. Hancock, C. H. Nguyen, and H. Mamitsuka. NetPathMiner: R/Bioconductor Package for Network Path Mining through Gene Expression. *Bioinformatics*, 30(21):3139–3141, 2014. [p310]

R. Nicolle, F. Radvanyi, and M. Elati. CoRegNet: Reconstruction and integrated analysis of co-regulatory networks. *Bioinformatics*, 31(18):3066–3068, 2015. [p310]

J. S. Parker, M. Mullins, M. C. Cheang, S. Leung, D. Voduc, T. Vickery, S. Davies, C. Fauron, X. He, Z. Hu, and others. Supervised risk predictor of breast cancer based on intrinsic subtypes. *Journal of Clinical Oncology*, 27(8):1160, 2009. URL https://doi.org/10.1200/jco.2008.18.1370. [p310]

G. Sales, E. Calura, D. Cavalieri, and C. Romualdi. Graphite-a Bioconductor package to convert pathway topology to gene network. *BMC Bioinformatics*, 13(1):20, 2012. URL https://doi.org/10.1186/1471-2105-13-20. [p311]

A. Subramanian, P. Tamayo, V. K. Mootha, S. Mukherjee, B. L. Ebert, M. A. Gillette, A. Paulovich, S. L. Pomeroy, T. R. Golub, E. S. Lander, and others. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005. URL https://doi.org/10.1073/pnas.0506580102. [p310]

W. Walter, F. Sánchez-Cabo, and M. Ricote. GOplot: An R Package for Visually Combining Expression Data with Functional Analysis. *Bioinformatics*, 31(17):2912–2914, 2015. [p316]

K. Zuberi, M. Franz, H. Rodriguez, J. Montojo, C. T. Lopes, G. D. Bader, and Q. Morris. GeneMANIA prediction server 2013 update. *Nucleic Acids Research*, 41(W1):W115–W122, 2013. [p311]

Claudia Cava
Institute of Molecular Bioimaging and Physiology, National Research Council (IBFM-CNR)
Via F.Cervi 93,20090
Segrate-Milan, Italy
ORCiD https://orcid.org/0000-0001-7191-5417
claudia.cava@ibfm.cnr.it

Isabella Castiglioni
Institute of Molecular Bioimaging and Physiology, National Research Council (IBFM-CNR)
Via F.Cervi 93,20090
Segrate-Milan, Italy
ORCiD https://orcid.org/0000-0002-5540-4104
isabella.castiglioni@ibfm.cnr.it

ciuupi: An R package for Computing Confidence Intervals that Utilize Uncertain Prior Information

by Rheanna Mainzer and Paul Kabaila

Abstract We have created the R package **ciuupi** to compute confidence intervals that utilize uncertain prior information in linear regression. Unlike post-model-selection confidence intervals, the confidence interval that utilizes uncertain prior information (CIUUPI) implemented in this package has, to an excellent approximation, coverage probability throughout the parameter space that is very close to the desired minimum coverage probability. Furthermore, when the uncertain prior information is correct, the CIUUPI is, on average, shorter than the standard confidence interval constructed using the full linear regression model. In this paper we provide motivating examples of scenarios where the CIUUPI may be used. We then give a detailed description of this interval and the numerical constrained optimization method implemented in R to obtain it. Lastly, using a real data set as an illustrative example, we show how to use the functions in **ciuupi**.

Introduction

Suppose that $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ is a random n-vector of responses, $\boldsymbol{X}$ is a known $n \times p$ matrix with linearly independent columns, $\boldsymbol{\beta}$ is an unknown parameter p-vector and $\boldsymbol{\varepsilon} \sim N(\boldsymbol{0}, \sigma^2 \boldsymbol{I}_n)$, where σ^2 is unknown. Let $\boldsymbol{a}$ be a specified nonzero p-vector and suppose that the parameter of interest is $\theta = \boldsymbol{a}^\top \boldsymbol{\beta}$. We refer to the model $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ as the "full" model and the confidence interval for θ based on this model as the "standard" confidence interval. It is often the case that applied statisticians want to utilize uncertain prior information in their analysis. This uncertain prior information may arise from previous experience, scientific background or expert opinion. For example, a common assumption in a factorial experiment is that higher order interaction terms are equal to zero. We consider the uncertain prior information that $\tau = \boldsymbol{c}^\top \boldsymbol{\beta} - t = 0$, where $\boldsymbol{c}$ is a specified nonzero p-vector that is linearly independent of $\boldsymbol{a}$ and t is a specified number. Our interest lies in computing (within a reasonable amount of time) a confidence interval for θ, with minimum coverage probability $1 - \alpha$, that utilizes the uncertain prior information that $\tau = 0$.

One could incorporate uncertain prior information in statistical inference using a Bayesian approach. In other words, a $1 - \alpha$ credible interval for θ could be constructed using an informative prior distribution for τ. However, the **ciuupi** package uses a frequentist approach to utilize the uncertain prior information that $\tau = 0$. Utilizing uncertain prior information in frequentist inference has a distinguished history, which includes Hodges and Lehmann (1952), Pratt (1961), Stein (1962), Cohen (1972), Bickel (1984), Kempthorne (1983, 1987, 1988), Casella and Hwang (1983, 1987), Goutis and Casella (1991), Tseng and Brown (1997), and Efron (2006).

The standard confidence interval has the desired coverage probability throughout the parameter space. However, it does not utilize the uncertain prior information that $\tau = 0$. One may attempt to utilize this uncertain prior information by carrying out a preliminary hypothesis test of the null hypothesis $\tau = 0$ against the alternative hypothesis $\tau \neq 0$. This attempt is based on the following two hopes. Firstly, if the prior information is correct then this test will lead to a confidence interval that is narrower than the standard confidence interval. Secondly, if this prior information happens to be incorrect then this test will effectively lead to the standard confidence interval. Unfortunately, this attempt fails miserably because, for certain values of $\boldsymbol{a}$, $\boldsymbol{c}$ and $\boldsymbol{X}$, this post-model-selection confidence interval has minimum coverage probability far below $1 - \alpha$ (see e.g. Kabaila and Giri, 2009b), making it unacceptable.

Kabaila and Giri (2009a) proposed a family of confidence intervals, with minimum coverage probability $1 - \alpha$, that utilize the uncertain prior information that $\tau = 0$ as follows. This family of confidence intervals have expected length that is less than the expected length of the standard interval when the prior information is correct and maximum (over the parameter space) expected length that is not too much larger than the expected length of the standard confidence interval. In addition, these confidence intervals have the same expected length as the standard confidence interval when the data strongly contradict the prior information. The admissibility result of Kabaila et al. (2010) implies that a confidence interval with the desired minimum coverage probability and expected length that is less than that of the standard confidence interval when the prior information is correct, must have an expected length that exceeds that of the standard interval for some parameter values.

Unfortunately, computing these confidence intervals is quite time consuming. Furthermore, there is no existing R package to compute these confidence intervals. Thus, if one wants to compute the confidence interval proposed by Kabaila and Giri (2009a) and originally computed using MATLAB programs, they may have to write their own programs to do so. The time and skill required to write such programs present large barriers to the use of this confidence interval in applied statistics.

? (**?**, Appendix A) described the family of confidence intervals proposed by Kabaila and Giri (2009a) when σ^2 is known. Each confidence interval in this family is specified by a different tradeoff between its performance when the prior information is correct and its performance when this prior information happens to be incorrect. **?** (**?**) then specified an attractive tradeoff that leads to a unique confidence interval. This interval and its coverage probability and expected length properties can now be easily and quickly computed using the R package **ciuupi**.

This confidence interval has the following three practical applications. Firstly, if σ^2 has been accurately estimated from previous data, as in the factorial experiment example described later, then it may be treated as being effectively known. Secondly, for $n - p$ sufficiently large ($n - p \geqslant 30$, say), if we replace the assumed known value of σ^2 by its usual estimator in the formula for the confidence interval then the resulting interval has, to a very good approximation, the same coverage probability and expected length properties as when σ^2 is known. Thirdly, some more complicated models (including those considered by **?**, **?**) can be approximated by the linear regression model with σ^2 known when certain unknown parameters are replaced by estimates.

The only information needed to assess the coverage probability and expected length of the confidence interval that utilizes uncertain prior information (CIUUPI) are the values of $\boldsymbol{a}$, $\boldsymbol{c}$, $\boldsymbol{X}$ and $1 - \alpha$. We stress that this assessment does not use the observed response $\boldsymbol{y}$. Indeed, if we want to choose between the CIUUPI and some other confidence interval, such as the standard confidence interval, then this choice must be made prior to any examination of the observed response $\boldsymbol{y}$.

In this paper we provide motivating examples of scenarios where this confidence interval may be used. We then describe, in detail, the CIUUPI computed by the **ciuupi** package and the numerical constrained optimization method implemented to obtain it. We contrast and compare the CIUUPI with a $1 - \alpha$ credible interval for θ constructed using an informative prior distribution for τ. Lastly, instructions on how to use the functions in **ciuupi** are given, using a real data set, from a factorial experiment, as an illustrative example. We hope that, by making **ciuupi** freely available, statisticians who have uncertain prior information of the type that we specify and wish to utilize it will be encouraged to use the CIUUPI instead of the potentially misleading post-model-selection confidence interval.

Motivating examples

The following motivating examples are provided by Kabaila and Giri (2013). These are examples of scenarios where the **ciuupi** package may be used to find a confidence interval for the parameter of interest θ that utilizes the uncertain prior information that $\tau = 0$.

- Pooling of normal means. Suppose that $y_i = \beta_1 + \varepsilon_i$ for $i = 1, \dots, n_1$ and $y_i = \beta_1 + \beta_2 + \varepsilon_i$ for $i = n_1 + 1, \dots, n_1 + n_2$, where the ε_i's are independent and identically distributed (i.i.d.) $N(0, \sigma^2)$. The parameter of interest is $\theta = \beta_1$ and we have uncertain prior information that $\tau = \beta_2 = 0$.
- One-way analysis of variance for two treatments. Suppose that $y_{ij} = \beta_i + \varepsilon_{ij}$ for $i = 1, 2$ and $j = 1, \dots, n_i$, where the ε_i's are i.i.d. $N(0, \sigma^2)$. The parameter of interest is $\theta = \beta_1$ and we have uncertain prior information that $\tau = \beta_1 - \beta_2 = 0$.
- A 2^k factorial experiment with two or more replicates. The parameter of interest θ is a specified contrast. For factorial experiments it is commonly believed that higher order interactions are negligible. Suppose that the uncertain prior information is that the highest order interaction is zero.
- One-way analysis of covariance with two treatments and normal errors. The parameter of interest θ is the difference in expected responses for the two treatments, for a specified value of the covariate. The uncertain prior information is that the hypothesis of 'parallellism' is satisfied.
- Polynomial regression. Suppose that $y_i = \beta_1 + \beta_2 x_i + \cdots + \beta_p x_i^{p-1} + \varepsilon_i$ for $i = 1, \dots, n$, where the ε_i's are i.i.d. $N(0, \sigma^2)$. The parameter of interest θ is the expected response for a specified value of the explanatory variable x. The uncertain prior information is that $\tau = \beta_p = 0$.
- Linear regression with at least one interaction term. The parameter of interest θ is a given linear combination of the regression parameters. The uncertain prior information is that a specified interaction term is 0.

In addition to the above examples, **?** have used the **ciuupi** package to aid in the computation of a confidence interval that utilizes uncertain prior information in the following more complicated scenario that arises in the analysis of both clustered and longitudinal data. Suppose that $y_{ij} = \beta_0 + \beta_1 x_{ij} + \beta_2 \overline{x}_i + \eta_i + \varepsilon_{it}$ for $i = 1, \dots, N$ and $j = 1, \dots, J$, where $\overline{x}_i = J^{-1} \sum_{j=1}^{J} x_{ij}$, the η_i's are i.i.d. $N(0, \sigma_\eta^2)$, and the ε_{ij}'s are i.i.d. $N(0, \sigma_\varepsilon^2)$. The parameter of interest is $\theta = \beta_1$ and we have uncertain prior information that $\tau = \beta_2 = 0$.

The confidence interval that utilizes uncertain prior information computed by ciuupi

Let $\widehat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$, the least squares estimator of $\boldsymbol{\beta}$. Then $\widehat{\theta} = \boldsymbol{a}^\top \widehat{\boldsymbol{\beta}}$ and $\widehat{\tau} = \boldsymbol{c}^\top \widehat{\boldsymbol{\beta}} - t$ are the least squares estimators of θ and τ, respectively. Now let $v_\theta = \mathrm{Var}(\widehat{\theta})/\sigma^2 = \boldsymbol{a}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{a}$ and $v_\tau = \mathrm{Var}(\widehat{\tau})/\sigma^2 = \boldsymbol{c}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{c}$. The known correlation between $\widehat{\theta}$ and $\widehat{\tau}$ is $\rho = \boldsymbol{a}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{c} / (v_\theta \, v_\tau)^{1/2}$. Let $\gamma = \tau / (\sigma \, v_\tau^{1/2})$, a scaled version of τ, and $\widehat{\gamma} = \widehat{\tau} / (\sigma \, v_\tau^{1/2})$, an estimator of γ. Assume that σ^2 is known.

The confidence interval that utilizes uncertain prior information about τ has the form

$$CI(b, s) = \left[\widehat{\theta} - v_\theta^{1/2} \, \sigma \, b\,(\widehat{\gamma}) - v_\theta^{1/2} \sigma \, s\,(\widehat{\gamma}) \, , \; \widehat{\theta} - v_\theta^{1/2} \, \sigma \, b\,(\widehat{\gamma}) + v_\theta^{1/2} \sigma \, s\,(\widehat{\gamma}) \right], \tag{T.3.1}$$

where $b : \mathbb{R} \to \mathbb{R}$ is an odd continuous function and $s : \mathbb{R} \to \mathbb{R}$ is an even continuous fuction. In addition, $b(x) = 0$ and $s(x) = z_{1-\alpha/2}$ for all $|x| \geqslant 6$, where the quantile z_a is defined by $P(Z \leqslant z_a) = a$ for $Z \sim N(0, 1)$. The functions b and s are fully specified by the vector $(b(1), b(2), \dots, b(5), s(0), s(1), \dots, s(5))$ as follows. By assumption, $b(0) = 0$, $(b(-1), b(-2), \dots, b(-5))$ $= (-b(1), -b(2), \dots, -b(5))$ and $(s(-1), \dots, s(-5)) = (s(1), \dots, s(5))$. The values of $b(x)$ and $s(x)$ for any $x \in [-6, 6]$ are found by cubic spline interpolation for the given values of $b(i)$ and $s(i)$ for $i = -6, -5, \dots, 0, 1, \dots, 5, 6$. The functions b and s are computed such that $CI(b, s)$ has minimum coverage probability $1 - \alpha$ and the desired expected length properties. This numerical computation method is described in detail in the next section. Note that the functions b and s are computed assuming that σ^2 is known.

As stated in the introduction, for $n - p$ sufficiently large ($n - p \geqslant 30$, say), if we replace the assumed known value of σ^2 by $\widehat{\sigma}^2 = (\boldsymbol{y} - \boldsymbol{X}\widehat{\boldsymbol{\beta}})^\top (\boldsymbol{y} - \boldsymbol{X}\widehat{\boldsymbol{\beta}})/(n - p)$ in the formula for $CI(b, s)$ then the resulting interval has, to a very good approximation, the same coverage probability and expected length properties as when σ^2 is known. In **ciuupi**, if no value of σ^2 is supplied then the user is given the option of replacing σ^2 by $\widehat{\sigma}^2$, with a warning that $n - p$ needs to be sufficiently large ($n - p \geqslant 30$, say).

Numerical constrained optimization method used to compute the vector $(b(1), b(2), \dots, b(5), s(0), s(1), \dots, s(5))$

Let

$$k(x) = \Psi \left(b(x) - s(x), \, b(x) + s(x); \, \rho \, (x - \gamma) \, , \, 1 - \rho^2 \right)$$

and

$$k^\dagger(x) = \Psi \left(-z_{1-\alpha/2}, \, z_{1-\alpha/2}; \, \rho \, (x - \gamma) \, , \, 1 - \rho^2 \right),$$

where $\Psi(\ell, u \, ; \, \mu, \, \sigma^2) = P\,(\ell \leqslant Z \leqslant u)$ for $Z \sim N(\mu, \, \sigma^2)$. A computationally convenient expression for the coverage probability of $CI(b, s)$ is

$$CP(\gamma; b, s, \rho) = 1 - \alpha + \int_0^6 \left(k(x) - k^\dagger(x) \right) \phi(x - \gamma) + \left(k(-x) - k^\dagger(-x) \right) \phi(x + \gamma) \, dx, \tag{T.3.2}$$

where ϕ denotes the $N(0, 1)$ pdf. This coverage probability depends on the unknown parameter γ, the functions b and s, the known correlation ρ and the desired minimum coverage probability $1 - \alpha$. Giri (2008) has shown that $CP(\gamma; b, s, \rho)$ is an even function of γ.

Define the scaled expected length of $CI(b, s)$ to be the expected length of $CI(b, s)$ divided by the expected length of the standard $1 - \alpha$ confidence interval, given by

$$\left[\widehat{\theta} - z_{1-\alpha/2} \, v_\theta^{1/2} \, \sigma, \, \widehat{\theta} + z_{1-\alpha/2} \, v_\theta^{1/2} \, \sigma \right]. \tag{T.3.3}$$

This scaled expected length of $CI(b, s)$ is given by

$$SEL(\gamma; s, \rho) = 1 + \frac{1}{z_{1-\alpha/2}} \int_{-6}^{6} \Big(s(x) - z_{1-\alpha/2} \Big) \, \phi\,(x - \gamma)\, dx.$$

This scaled expected length depends on the unknown parameter γ, the function s, the known correlation ρ and the desired minimum coverage probability $1-\alpha$. Giri (2008) has shown that $SEL(\gamma; s, \rho)$ is an even function of γ.

We compute the functions b and s such that $CI(b, s)$ has minimum coverage probability $1-\alpha$ and the desired expected length properties as follows. For given $\lambda \in [0, \infty)$, we minimize the objective function

$$(SEL\,(\gamma = 0; s, \rho) - 1) + \lambda \int_{-\infty}^{\infty} (SEL(\gamma; s, \rho) - 1)\; d\gamma, \tag{T.3.4}$$

with respect to the vector $\big(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5)\big)$, subject to the coverage constraint $CP(\gamma) \geqslant 1-\alpha$ for all γ. Equivalently, minimize the objective function

$$\xi\,(SEL(\gamma = 0; s, \rho) - 1) + (1 - \xi) \int_{-\infty}^{\infty} (SEL(\gamma; s, \rho) - 1)\, d\gamma \tag{T.3.5}$$

subject to this constraint, where $\xi = 1/(1+\lambda)$. A computationally convenient formula for the objective function (T.3.4) is

$$\frac{2}{z_{1-\alpha/2}} \int_{0}^{6} \Big(s(h) - z_{1-\alpha/2} \Big)\, (\lambda + \phi(h))\, dh. \tag{T.3.6}$$

Since we are minimizing this objective function, we can leave out the constant at the front of the integral.

When λ is large, this numerical computation recovers the standard confidence interval (T.3.3) for θ. As λ decreases towards 0, this computation puts increasing weight on achieving a small value of $SEL(\gamma = 0; s, \rho)$, i.e. an improved confidence interval performance when the uncertain prior information that $\tau = 0$ is correct. However, as λ decreases, $\max_\gamma SEL(\gamma; s, \rho)$ increases, i.e. the performance of the confidence interval when the prior information happens to be incorrect is degraded. Following **?**, we choose λ such that the "gain" when the prior information is correct, as measured by

$$1 - (SEL\,(\gamma = 0; s, \rho))^2\,, \tag{T.3.7}$$

is equal to the maximum possible "loss" when the prior information happens to be incorrect, as measured by

$$\left(\max_\gamma SEL\,(\gamma; s, \rho) \right)^2 - 1. \tag{T.3.8}$$

We denote this value of λ by λ^*. Our computational implementation of the constraint $CP(\gamma) \geqslant 1-\alpha$ for all γ is to require that $CP(\gamma) \geqslant 1-\alpha$ for all $\gamma \in \{0, 0.05, 0.1, \ldots, 8\}$. By specifying constraints on the coverage probability $CP(\gamma)$ for such a fine grid of nonnegative values of γ, we ensure that, to an exceedingly good approximation, $CP(\gamma; b, s, \rho) \geqslant 1-\alpha$ for all values of γ.

In summary, we compute the vector $\big(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5)\big)$ by minimizing (T.3.6), where λ is chosen such that (T.3.7) = (T.3.8), subject to the constraints $CP(\gamma; b, s, \rho) \geqslant 1-\alpha$ for all $\gamma \in \{0, 0.05, 0.1, \ldots, 8\}$. Once $(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5))$ has been computed in this way, we can easily compute the confidence interval that utilizes the uncertain prior information (CIUUPI) for observed response $\boldsymbol{y}$.

This constrained optimization procedure is carried out using the `slsqp` function in the **nloptr** package (see Johnson, 2014). Perhaps surprisingly, the large number of constraints on the coverage probability $CP(\gamma; b, s, \rho)$ is handled well by the `slsqp` function. The integrals in (T.3.2) and (T.3.6) are computed as follows. For greater accuracy, each integral is split into a sum of six integrals, with lower and upper endpoints consisting of successive knots. Each of these integrals is then computed using Gauss Legendre quadrature with five nodes. Gauss Legendre quadrature was found to be both faster and more accurate than the R function `integrate`. This quadrature is carried out using the `gauss.quad` function in the **statmod** package (see Smyth, 2005).

A comparison of the CIUUPI with a Bayesian interval estimator

Kabaila and Dharmarathne (2015) compare Bayesian and frequentist interval estimators for θ in the linear regression context considered in this paper when σ^2 is unknown. They find that the Bayesian and frequentist interval estimators differ substantially. In this section we compare a $1-\alpha$ credible interval for θ with the CIUUPI, assuming that σ^2 is known.

For ease of comparison of the CIUUPI with a credible interval, we re-express the regression sampling model as follows. Let the $n \times p$ matrix $\widetilde{\boldsymbol{X}}$ be obtained from $\boldsymbol{X}$ using the transformation described in Appendix B of Kabaila and Dharmarathne (2015). The attractive property of $\widetilde{\boldsymbol{X}}$ is that

$$(\widetilde{\boldsymbol{X}}^\top \widetilde{\boldsymbol{X}})^{-1} = \begin{pmatrix} \boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{p-2} \end{pmatrix}, \text{ where } \boldsymbol{V} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}.$$

We re-express the regression sampling model as $\widetilde{\boldsymbol{y}} = \widetilde{\boldsymbol{X}} \left[\vartheta, \gamma, \boldsymbol{\chi}^\top \right]^\top + \widetilde{\boldsymbol{\varepsilon}}$, where $\widetilde{\boldsymbol{y}} = \boldsymbol{y}/\sigma$, $\widetilde{\boldsymbol{\varepsilon}} = \boldsymbol{\varepsilon}/\sigma$ and $\vartheta = \theta/(\sigma\, v_\theta^{1/2})$. Obviously, $\widetilde{\boldsymbol{\varepsilon}} \sim N(\boldsymbol{0}, \boldsymbol{I}_n)$. Let $(\widehat{\vartheta}, \widehat{\gamma}, \widehat{\boldsymbol{\chi}})$ denote the least squares estimator of $(\vartheta, \gamma, \boldsymbol{\chi})$. Note that $\widehat{\vartheta} = \widehat{\theta}/(\sigma\, v_\theta^{1/2})$. Clearly, $(\widehat{\vartheta}, \widehat{\gamma})$ has a bivariate normal distribution with mean (ϑ, γ) and covariance matrix $\boldsymbol{V}$ and, independently, $\widehat{\boldsymbol{\chi}} \sim N(\boldsymbol{\chi}, \boldsymbol{I}_{p-2})$. Dividing the endpoints of the CIUUPI by $\sigma\, v_\theta^{1/2}$, we obtain the following confidence interval for ϑ:

$$\left[\widehat{\vartheta} - b(\widehat{\gamma}) - s(\widehat{\gamma}),\ \widehat{\vartheta} - b(\widehat{\gamma}) + s(\widehat{\gamma}) \right], \tag{T.4.1}$$

where the functions b and s have been obtained using the constrained optimization described in the previous section.

The uncertain prior information that $\tau = 0$ implies the uncertain prior information that $\gamma = 0$. The properties of a Bayesian $1 - \alpha$ credible interval depend greatly on the prior distribution chosen for $(\vartheta, \gamma, \boldsymbol{\chi})$. We have chosen a prior distribution that leads to a credible interval with some similarities to the CIUUPI. Assume that the prior probability density function of $(\vartheta, \gamma, \boldsymbol{\chi})$ is proportional to $\xi^*\, \delta(\tau) + (1 - \xi^*)$, where $\xi^* = 1/(1 + \lambda^*)$ and δ denotes the Dirac delta function. In other words, we assume an improper prior density for τ that consists of a mixture of an infinite rectangular unit-height 'slab' and a Dirac delta function 'spike', combined with noninformative prior densities for the other parameters. This prior density is a Bayesian analogue of the weight function used in the weighted average over γ, (T.3.5). It may be shown that the marginal posterior density of ϑ is

$$w(\widehat{\gamma})\, \phi\left(\vartheta;\, \widehat{\vartheta} - \rho\widehat{\gamma}, 1 - \rho^2\right) + (1 - w\,(\widehat{\gamma}))\, \phi\left(\vartheta;\, \widehat{\vartheta}, 1\right), \tag{T.4.2}$$

where $w(\widehat{\gamma}) = 1/\big(1 + \lambda^* \sqrt{2\pi}\, \exp(\widehat{\gamma}^2/2)\big)$ and $\phi(\,\cdot\,; \mu, \nu)$ denotes the $N(\mu, \nu)$ pdf. We note that this posterior density is a mixture of two normal probability density functions, such that the weight given to the posterior density centred at $\widehat{\vartheta}$ increases with increasing $\widehat{\gamma}^2$, when $\lambda^* > 0$. It is evident from (T.4.2) that the highest posterior density Bayesian $1 - \alpha$ credible interval may consist of the union of two disjoint intervals. For this reason, we consider the shortest $1 - \alpha$ credible interval.

Note that the graph of the function (T.4.2) of ϑ consists of the graph of the function

$$w(\widehat{\gamma})\, \phi\left(\vartheta; -\rho\widehat{\gamma}, 1 - \rho^2\right) + (1 - w(\widehat{\gamma}))\, \phi\left(\vartheta; 0, 1\right),$$

shifted to the right by $\widehat{\vartheta}$. We can therefore express the shortest $1 - \alpha$ credible interval for ϑ in the form $[\widehat{\vartheta} + l(\widehat{\gamma}), \widehat{\vartheta} + u(\widehat{\gamma})]$, for the appropriate functions l and u. We compare this interval with the frequentist $1 - \alpha$ confidence interval (T.4.1) as follows. Let $b_B(\widehat{\gamma}) = -(l(\widehat{\gamma}) + u(\widehat{\gamma}))/2$ and $s_B(\widehat{\gamma}) = (u(\widehat{\gamma}) - l(\widehat{\gamma}))/2$. Then $[\widehat{\vartheta} + l(\widehat{\gamma}), \widehat{\vartheta} + u(\widehat{\gamma})]$ is equal to

$$\left[\widehat{\vartheta} - b_B(\widehat{\gamma}) - s_B(\widehat{\gamma}),\ \widehat{\vartheta} - b_B(\widehat{\gamma}) + s_B(\widehat{\gamma}) \right], \tag{T.4.3}$$

which has a similar form to (T.4.1), but with b and s replaced by b_B and s_B respectively. Therefore, we may compare the interval (T.4.1) with (T.4.3) by comparing the functions b and s with the functions b_B and s_B, respectively. We will also compare the interval (T.4.1) with (T.4.3) by comparing the frequentist coverage probability function of (T.4.3).

Using the ciuupi package

In this section we use a real data set to illustrate how each of the six functions in **ciuupi** works. Table 1 below gives the name of each of the functions and a short description of what it does. In the following subsections we show how the functions in Table 1 are used in R.

Function	Description
bsciuupi	Compute the optimized vector $\big(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5)\big)$
bsspline	Evaluate $b(x)$ and $s(x)$ at x
cpci	Evaluate $CP(\gamma; b, s, \rho)$ at γ
selci	Evaluate $SEL(\gamma; b, s, \rho)$ at γ
ciuupi	Compute the CIUUPI, i.e. compute $CI(b, s)$
cistandard	Compute the standard confidence interval

Table 1: Functions in the **ciuupi** package

Factorial experiment example

Consider the 2 × 2 factorial experiment described by Kabaila and Giri (Discussion 5.8, 2009a), which has been extracted from a real 2^3 factorial data set provided by Box et al. (1963). The two factors are the time of addition of HNO_3 and the presence of a 'heel'. These factors are labelled A and B, respectively. Define $x_1 = -1$ and $x_1 = 1$ for "Time of addition of HNO_3" equal to 2 hours and 7 hours, respectively. Also define $x_2 = -1$ and $x_2 = 1$ for "heel absent" and "heel present", respectively. Assume a model of the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \varepsilon,$$

where $\varepsilon \sim N(0, \sigma^2)$. This model can be written in matrix form as

$$\boldsymbol{y} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}$$

where $\boldsymbol{\beta} = \big(\beta_0,\, \beta_1,\, \beta_2,\, \beta_{12}\big)$,

$$\boldsymbol{X} = \begin{pmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

and $\boldsymbol{\varepsilon} \sim N(0,\, \sigma^2 \boldsymbol{I}_n)$. According to Box et al. (1963), a very accurate estimate of σ, obtained from previous related experiments, is 0.8.

Suppose that the parameter of interest θ is $\big($expected response when factor A is high and factor B is low$\big) - \big($expected response when factor A is low and factor B is low$\big)$. In other words, $\theta = 2(\beta_1 - \beta_{12})$, so that $\theta = \boldsymbol{a}^\top \boldsymbol{\beta}$, where $\boldsymbol{a} = (0, 2, 0, -2)$. Our aim is to find a confidence interval, with minimum coverage 0.95, for θ. We suppose that there is uncertain prior information that the two-factor interaction is zero. In other words, we suppose that there is uncertain prior information that $\beta_{12} = 0$. The uncertain prior information is, then, that $\tau = \boldsymbol{c}^\top \boldsymbol{\beta} - t = 0$, where $\boldsymbol{c} = (0, 0, 0, 1)$ and $t = 0$. Now that we have specified $\boldsymbol{a}$, $\boldsymbol{c}$ and $\boldsymbol{X}$, we can compute $\rho = \boldsymbol{a}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{c} / (v_\theta \, v_\tau)^{1/2} = -1/\sqrt{2} = -0.707$.

Evaluating the confidence interval (no examination of the observed response)

First suppose that we have not yet examined the observed response $\boldsymbol{y}$ and that we are interested in knowing how the confidence interval that utilizes uncertain prior information (CIUUPI) performs for given values of $1 - \alpha$, $\boldsymbol{a}$, $\boldsymbol{c}$ and $\boldsymbol{X}$. We begin by storing the values of α, $\boldsymbol{a}$, $\boldsymbol{c}$ and $\boldsymbol{X}$ in R as follows.

```
# Specify alpha, a, c and x.
alpha <- 0.05
a <- c(0, 2, 0, -2)
c <- c(0, 0, 0, 1)
x <- cbind(rep(1, 4), c(-1, 1, -1, 1), c(-1, -1, 1, 1), c(1, -1, -1, 1))
```

Next we use the numerical constrained optimization to compute the values at the knots of the functions b and s that define the CIUUPI. We must specify whether natural cubic spline interpolation (natural = 1) or clamped cubic spline interpolation (natural = 0) is used in the description of these functions. In the case of clamped cubic spline interpolation the first derivatives of b and s are set to zero at -6 and 6. Natural cubic spline interpolation is the default, and is carried out using `splinefun` in the **stats** package. Clamped cubic spline interpolation is carried out using `cubicspline` in the **pracma** package. The nonlinear constrained optimization using natural cubic spline interpolation for the description of the functions b and s is much faster and results in a coverage probability that is slightly closer to $1 - \alpha$ throughout the parameter space. For this example we are able to obtain the vector $(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5))$ in 6.56 minutes when using natural cubic spline interpolation and in 21.27 minutes when using clamped cubic spline interpolation. This computation was carried out on a PC with an Intel i7-7500 CPU (3.4GHz) and 32GB of RAM. The following code is used to obtain the vector $\big(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5)\big)$ that specifies the CIUUPI, which is obtained from the numerical constrained optimization that uses natural cubic spline interpolation for the description of the functions b and s.

```
# Compute (b(1), b(2), ..., b(5), s(0), s(1), ..., s(5)) that specifies the CIUUPI
bsvec <- bsciuupi(alpha, a = a, c = c, x = x)
bsvec
```

Alternatively, since we know that $\rho = -0.707$, we could obtain the vector $\big(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5)\big)$ that specifies the CIUUPI using the code

```
# Compute (b(1), b(2), ..., b(5), s(0), s(1), ..., s(5)) that specifies the CIUUPI,
# given rho
bsvec2 <- bsciuupi(alpha, rho = -0.707)
```

Now that we have the vector $\big(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5)\big)$ that specifies the CIUUPI, we can graph the functions b and s using the following code:

```
# Compute the functions b and s that specify the CIUUPI on a grid of values
splineval <- bsspline(seq(0, 8, by = 0.1), bsvec, alpha)

# The first 5 values of bsvect are b(1),b(2),...,b(5).
# The last 6 values are s(0),s(1),...,s(5).
xseq <- seq(0, 6, by = 1)
bvec <- c(0, bsvec[1:5], 0)
svec <- c(bsvec[6:11], qnorm(1 - alpha/2))

# Plot the functions b and s
plot(seq(0, 8, by = 0.1), splineval[, 2], type = "l", main = "b function",
     ylab = " ", las = 1, lwd = 2, xaxs = "i", col = "blue", xlab = "x")
points(xseq, bvec, pch = 19, col = "blue")
plot(seq(0, 8, by = 0.1), splineval[, 3], type = "l", main = "s function",
     ylab = " ", las = 1, lwd = 2, xaxs = "i", col = "blue", xlab = "x")
points(xseq, svec, pch = 19, col = "blue")
```

Figure 1 shows the graphs of the functions b and s that specify the CIUUPI, when these functions are described using natural cubic spline interpolation, for this example. For comparison, Figure 2 shows the graphs of the functions b and s that specify the CIUUPI, when these functions are described using clamped cubic spline interpolation. These figures are quite similar; there is a small difference in both the b and s functions near $x = 6$.

We can also use the vector $\big(b(1), b(2), \ldots, b(5), s(0), s(1), \ldots, s(5)\big)$ that specifies the CIUUPI to evaluate and then plot the coverage probability $CP(\gamma; b, s, \rho)$ and scaled expected length $SEL(\gamma; s, \rho)$ as functions of γ. This is done using the following code.

```
# Compute the coverage probability and scaled expected for a grid of values of gamma
gam <- seq(0, 10, by = 0.1)
cp <- cpciuupi(gam, bsvec, alpha, a = a, c = c, x = x)
sel <- selciuupi(gam, bsvec, alpha, a = a, c = c, x = x)

# Plot the coverage probability and squared scaled expected length
plot(gam, cp, type = "l", lwd = 2, ylab = "", las = 1, xaxs = "i",
main = "Coverage Probability", col = "blue",
xlab = expression(paste("|", gamma, "|")), ylim = c(0.9495, 0.9505))
abline(h = 1-alpha, lty = 2)
plot(gam, sel^2, type = "l", lwd = 2, ylab = "", las = 1, xaxs = "i",
main = "Squared SEL", col = "blue",
xlab = expression(paste("|", gamma, "|")), ylim = c(0.83, 1.17))
abline(h = 1, lty = 2)
```

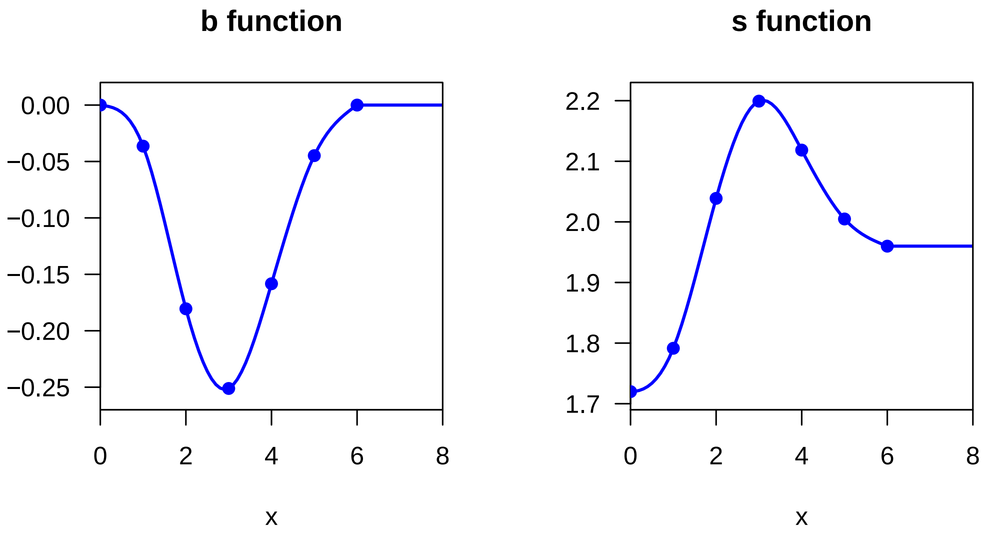


Figure 1: Graphs of the functions b and s for the factorial experiment example for the CIUUPI, with minimum coverage probability 0.95, when they are described using natural cubic spline interpolation, for the factorial experiment example.

Figure 3 shows the graphs of $CP(\gamma; b, s, \rho)$ and the square of $SEL(\gamma; b, s, \rho)$ for the CIUUPI (where the functions b and s have been specified by natural cubic spline interpolation) produced by this code.

We can see from Figure 3 that, regardless of the value of γ, the coverage probability of the CIUUPI is extremely close to $1 - \alpha$. We can also see that the expected length of the CIUUPI is less than the expected length of the standard confidence interval when γ is small, with the minimum scaled expected length achieved when $\gamma = 0$. For moderate values of $|\gamma|$, the expected length of the standard interval is less than the expected length of the CIUUPI. However, for large $|\gamma|$, the expected length of the CIUUPI is essentially the same as the expected length of the standard interval.

For comparison, Figure 4 shows the graphs of $CP(\gamma; b, s, \rho)$ and the square of $SEL(\gamma; b, s, \rho)$ for the CIUUPI when the functions b and s are described by clamped cubic spline interpolation.

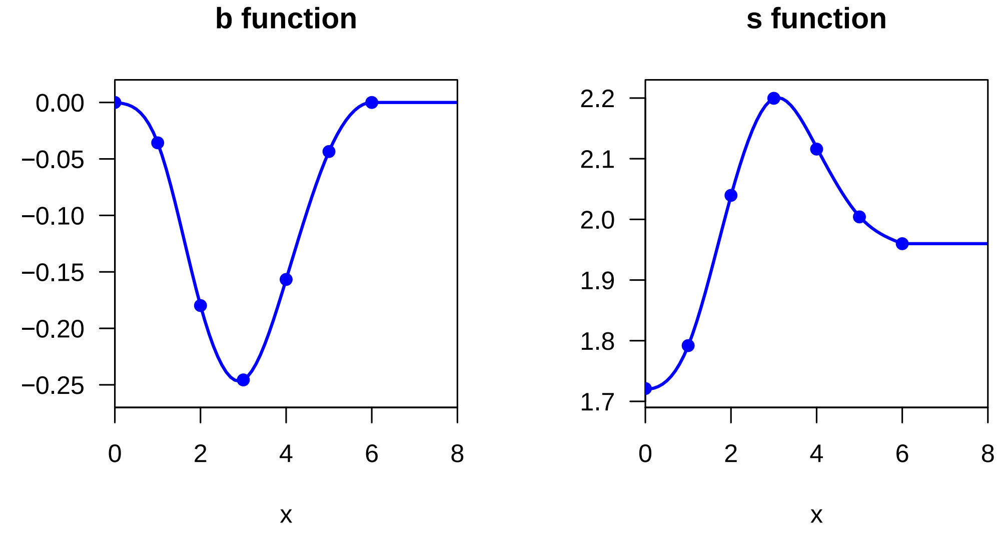


Figure 2: Graphs of the functions b and s for the factorial experiment example for the CIUUPI, with minimum coverage probability 0.95, when they are described using clamped cubic spline interpolation, for the factorial experiment example.

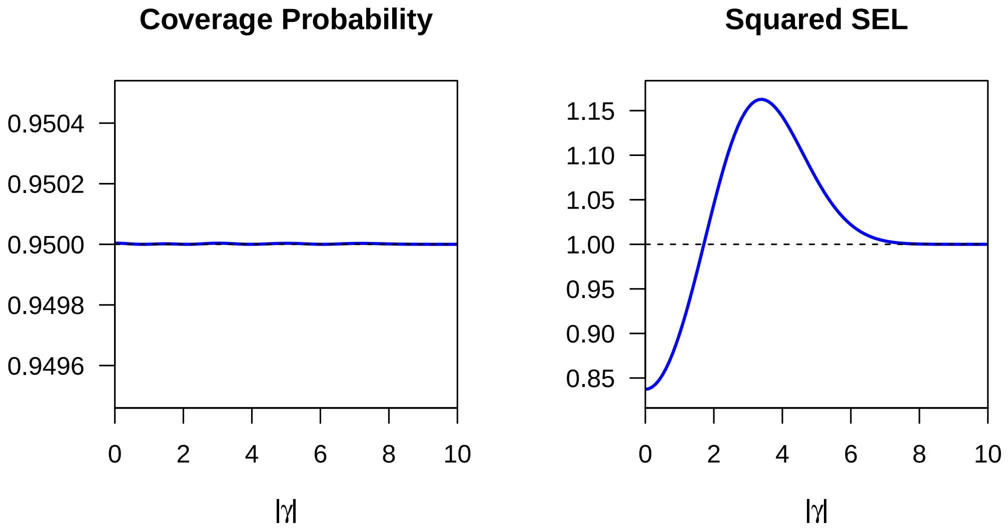


Figure 3: Graphs of the $CP(\gamma; b, s, \rho)$ and the square of $SEL(\gamma; b, s, \rho)$ functions for the CIUUPI, with minimum coverage probability 0.95, where the functions b and s are described by natural cubic spline interpolation, for the factorial experiment example.

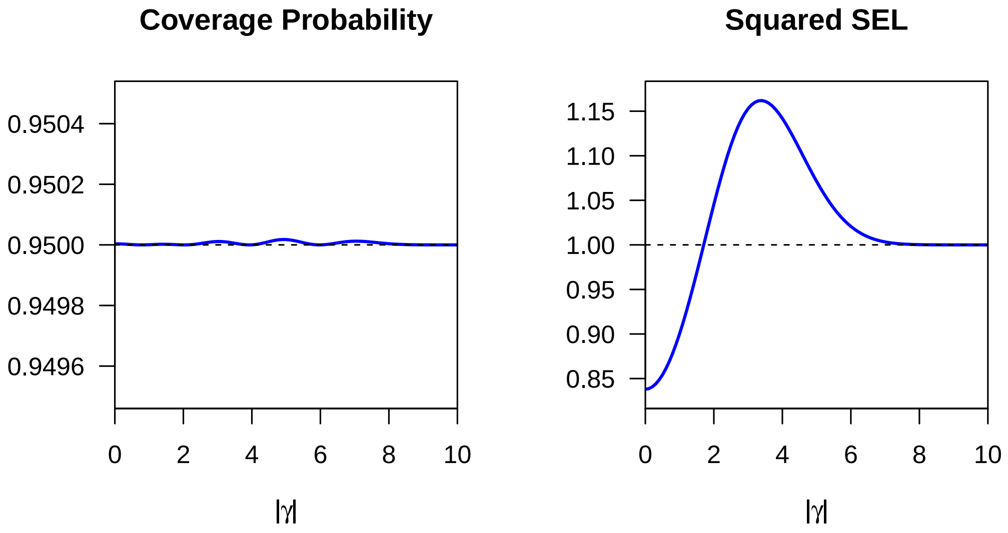


Figure 4: Graphs of the $CP(\gamma; b, s, \rho)$ and the square of $SEL(\gamma; b, s, \rho)$ functions for the CIUUPI, with minimum coverage probability 0.95, where the functions b and s are described by clamped cubic spline interpolation, for the factorial experiment example.

Figures (5) and (6) show the differences between the Bayesian 95% credible interval and the 95% CIUUPI. Figure (5) shows the graphs of the b and b_B functions (left panel), and the s and s_B functions (right panel), for the factorial experiment example. Note that, similarly to b and s, b_B is an odd continuous function and s_B is an even continuous function. Figure (6) shows the graph of the frequentist coverage probability of the Bayesian 95% credible interval, for the factorial experiment example. This coverage probability is also an even function of γ. Unlike the coverage probability of the CIUUPI, the minimum over γ of the frequentist coverage probability of the Bayesian 95% credible interval is substantially less than 0.95.

Computing the confidence interval (using the observed response)

The observed response for the factorial experiment example data is $\boldsymbol{y} = (87.2, 88.4, 86.7, 89.2)$ and σ is assumed to take the value 0.8. We use the function `ciuupi` to return the confidence interval (T.3.1) for θ that utilizes the uncertain prior information that $\tau = 0$. Continuing from the previous example, this is done in R as follows:

```
# Using the vector (b(1),b(2),...,b(5),s(0),s(1),...,s(5)), compute the CIUUPI
# for this particular data
t <- 0
y <- c(87.2, 88.4, 86.7, 89.2)

ci <- ciuupi(alpha, a, c, x, bsvec, t, y, natural = 1, sig = 0.8); ci
```

We obtain the output

```
            lower    upper
ciuupi   -0.7710755 3.218500
```

For comparison purposes, the function `standard_CI` will return the standard confidence interval (T.3.3) for θ. The code

```
# Compute the standard confidence interval
cistandard(a = a, x = x, y = y, alpha = alpha, sig = 0.8)
```

will return

```
             lower    upper
standard -1.017446 3.417446
```

The 95% confidence interval that utilizes uncertain prior information $[-0.77, 3.22]$ is much shorter than the standard confidence interval $[-1.02, 3.42]$. These are observed values of confidence intervals that have, to an excellent approximation, the same coverage probability. For comparison, a 95% Bayesian credible interval for θ is $[-0.25, 3.51]$. Although this interval is shorter than the CIUUPI, it can be seen from Figure (6) that the minimum over γ of the frequentist coverage of the Bayesian credible interval is substantially less than 0.95.

Discussion

It is very common in applied statistics to carry out preliminary data-based model selection using, for example, hypothesis tests or minimizing a criterion such as the AIC. As pointed out by Leamer (1978, chapter 5), such model selection may be motivated by the desire to utilize uncertain prior information in subsequent statistical inference. He goes even further when he states, on p.123, that "The mining of data that is common among non-experimental scientists constitutes prima facie evidence of the existence of prior information". One may attempt to utilize such prior information by constructing confidence intervals, using the same data, based on the assumption that the selected model had been given to us *a priori*, as the true model. This assumption is false and it can lead to confidence intervals that have minimum coverage probability far below the desired minimum coverage $1 - \alpha$ (see e.g. Kabaila, 2009, Leeb and Pötscher, 2005), making them invalid.

A numerical constrained optimization approach to the construction of valid confidence intervals and sets that utilize uncertain prior information has been applied by Farchione and Kabaila (2008), Kabaila and Giri (2009a), Kabaila and Giri (2013), Kabaila and Giri (2014), Kabaila and Tissera (2014) and Abeysekera and Kabaila (2017). In each case, numerical constrained optimization was performed using programs written in MATLAB, restricting the accessibility of these confidence intervals and sets. The R package **ciuupi** is a first step in making these types of confidence intervals and sets more widely accessible.

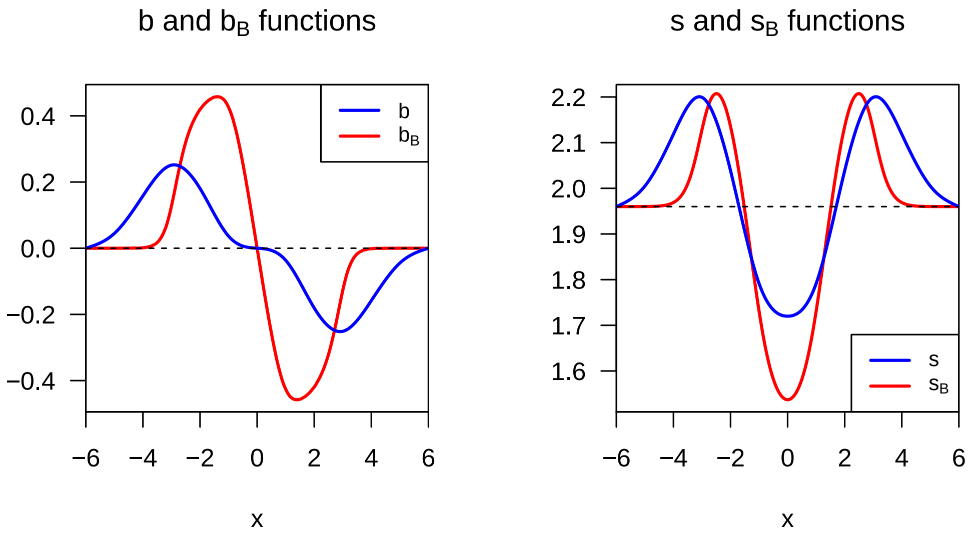


Figure 5: Graphs of the b and b_B functions (left panel), and the s and s_B functions (right panel), for the factorial experiment example.

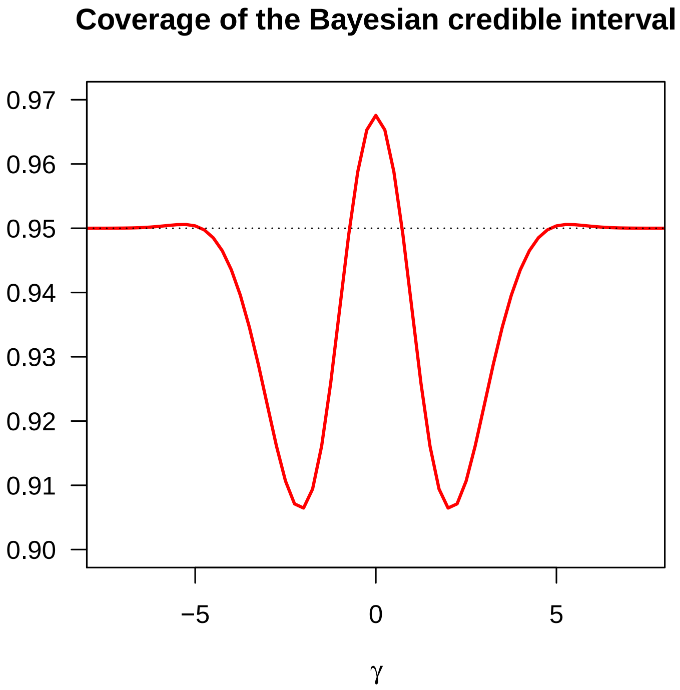


Figure 6: The frequentist coverage probability of the Bayesian 95% confidence interval, for the factorial experiment example.

Bibliography

W. Abeysekera and P. Kabaila. Optimized recentered confidence spheres for the multivariate normal mean. *Electronic Journal of Statistics*, 11:1798–1826, 2017. URL https://doi.org/10.1214/17-EJS1272. [p333]

P. J. Bickel. Parametric robustness: Small biases can be worthwhile. *The Annals of Statistics*, 12: 864–879, 1984. ISSN 0090-5364. URL https://doi.org/10.1214/aos/1176346707. [p323]

G. E. P. Box, L. R. Connor, W. R. Cousins, O. L. Davies, F. R. Hinsworth, and G. P. Sillitto. *The Design and Analysis of Industrial Experiments*. Oliver and Boyd, London, 2nd edition, 1963. [p328]

G. Casella and J. T. Hwang. Empirical Bayes Confidence Sets for the Mean of a Multivariate Normal Distribution. *Journal of the American Statistical Association*, 78:688–698, 1983. URL https://doi.org/10.2307/2288139. [p323]

G. Casella and J. T. Hwang. Employing vague prior information in the construction of confidence sets. *Journal of Multivariate Analysis*, 21:79–104, 1987. URL https://doi.org/10.1016/0047-259X(87)90100-X. [p323]

A. Cohen. Improved confidence intervals for the variance of a normal distribution. *Journal of the American Statistical Association*, 67:382–387, 1972. ISSN 0162-1459. URL https://doi.org/10.2307/2284389. [p323]

B. Efron. Minimum volume confidence regions for a multivariate normal mean vector. *Journal of the Royal Statistical Society B*, 68:655–670, 2006. URL https://doi.org/10.1111/j.1467-9868.2006.00560.x. [p323]

D. Farchione and P. Kabaila. Confidence intervals for the normal mean utilizing prior information. *Statistics & Probability Letters*, 78:1094–1100, 2008. URL https://doi.org/10.1016/j.spl.2007.11.003. [p333]

K. Giri. *Confidence Intervals in Regression Utilizing Prior Information*. PhD thesis, Department of Mathematics and Statistics, La Trobe University, 2008. [p325, 326]

C. Goutis and G. Casella. Improved invariant confidence intervals for a normal variance. *The Annals of Statistics*, 19:2015–2031, 1991. URL https://doi.org/10.1214/aos/1176348384. [p323]

J. L. Hodges and E. L. Lehmann. The use of previous experience in reaching statistical decisions. *Annals of Mathematical Statistics*, 23:396–407, 1952. ISSN 0003-4851. URL https://doi.org/10.1214/aoms/1177729384. [p323]

S. G. Johnson. The nlopt nonlinear-optimization package. http://ab-initio.mit.edu/nlopt, 2014. [p326]

P. Kabaila. The coverage properties of confidence regions after model selection. *International Statistical Review*, 77:405–414, 2009. URL https://doi.org/10.1111/j.1751-5823.2009.00089.x. [p333]

P. Kabaila and G. Dharmarathne. A Comparison of Bayesian and Frequentist Interval Estimators in Regression That Utilize Uncertain Prior Information. *Australian & New Zealand Journal of Statistics*, 57(1):99–118, 2015. URL https://doi.org/10.1111/anzs.12104. [p326, 327]

P. Kabaila and K. Giri. Confidence intervals in regression utilizing prior information. *Journal of Statistical Planning and Inference*, 139:3419–3429, 2009a. URL https://doi.org/10.1016/j.jspi.2009.03.018. [p323, 324, 328, 333]

P. Kabaila and K. Giri. Upper bounds on the minimum coverage probability of confidence intervals in regression after model selection. *Australian & New Zealand Journal of Statistics*, 51:271–287, 2009b. URL https://doi.org/10.1111/j.1467-842X.2009.00544.x. [p323]

P. Kabaila and K. Giri. Further properties of frequentist confidence intervals in regression that utilize uncertain prior information. *Australian & New Zealand Journal of Statistics*, pages 259–270, 2013. URL https://doi.org/10.1111/anzs.12038. [p324, 333]

P. Kabaila and K. Giri. Simultaneous confidence intervals for the population cell means, for two-by-two factorial data, that utilize uncertain prior information. *Communications in Statistics – Theory and Methods*, 43:4074–4087, 2014. URL https://doi.org/10.1080/03610926.2012.718846. [p333]

P. Kabaila and D. Tissera. Confidence intervals in regression that utilize uncertain prior information about a vector parameter. *Australian & New Zealand Journal of Statistics*, 56:371–383, 2014. URL https://doi.org/10.1111/anzs.12090. [p333]

P. Kabaila, K. Giri, and H. Leeb. Admissibility of the usual confidence interval in linear regression. *Electronic Journal of Statistics*, 4:300–312, 2010. URL https://doi.org/10.1214/10-EJS563. [p323]

P. J. Kempthorne. Minimax-Bayes Compromise Estimators. *Proc. Bus. and Econ. Statist. Sec. Amer. Statist. Assoc.*, pages 563–573, 1983. [p323]

P. J. Kempthorne. Numerical specification of discrete least favorable prior distributions. *Society for Industrial and Applied Mathematics. Journal on Scientific and Statistical Computing*, 8:171–184, 1987. ISSN 0196-5204. URL https://doi.org/10.1137/0908028. [p323]

P. J. Kempthorne. Controlling risks under different loss functions: The compromise decision problem. *The Annals of Statistics*, 16:1594–1608, 1988. ISSN 0090-5364. URL https://doi.org/10.1214/aos/1176351055. [p323]

E. E. Leamer. *Specification Searches: Ad Hoc Inference with Nonexperimental Data.* John Wiley & Sons, 1978. [p333]

H. Leeb and B. M. Pötscher. Model selection and inference: Facts and fiction. *Econometric Theory*, 21:21–59, 2005. URL https://doi.org/10.1017/S0266466605050036. [p333]

J. W. Pratt. Length of confidence intervals. *Journal of the American Statistical Association*, 56: 549–567, 1961. ISSN 0162-1459. URL https://doi.org/10.2307/2282079. [p323]

G. K. Smyth. Numerical integration. *Encyclopedia of Biostatistics*, pages 3088–3095, 2005. URL https://doi.org/10.1002/9781118445112.stat05029. [p326]

C. M. Stein. Confidence sets for the mean of a multivariate normal distribution. *Journal of the Royal Statistical Society. Series B (Methodological).*, 24:365–396, 1962. URL https://doi.org/10.1111/j.2517-6161.1962.tb00458.x. [p323]

Y. L. Tseng and L. D. Brown. Good exact confidence sets for a multivariate normal mean. *The Annals of Statistics*, 5:2228–2258, 1997. URL https://doi.org/10.1214/aos/1069362396. [p323]

Dr. Rheanna Mainzer
School of Mathematics and Statistics
The University of Melbourne
Australia
rheanna.mainzer@unimelb.edu.au

Dr. Paul Kabaila
Department of Mathematics and Statistics
La Trobe University
Australia
P.Kabaila@latrobe.edu.au

ipwErrorY: An R Package for Estimation of Average Treatment Effect with Misclassified Binary Outcome

by Di Shu and Grace Y. Yi

Abstract It has been well documented that ignoring measurement error may result in severely biased inference results. In recent years, there has been limited but increasing research on causal inference with measurement error. In the presence of misclassified binary outcome variable, Shu and Yi (2017) considered the inverse probability weighted estimation of the average treatment effect and proposed valid estimation methods to correct for misclassification effects for various settings. To expedite the application of those methods for situations where misclassification in the binary outcome variable is a real concern, we implement correction methods proposed by Shu and Yi (2017) and develop an R package **ipwErrorY** for general users. Simulated datasets are used to illustrate the use of the developed package.

Introduction

Causal inference methods have been widely used in empirical research (e.g., Rothman et al., 2008; Imbens and Rubin, 2015; Hernán and Robins, 2019). The propensity score, defined to be the probability of an individual to receive the treatment, plays an important role in conducting causal inference (Rosenbaum and Rubin, 1983). Many causal inference methods have been developed based on the propensity score (e.g., Rosenbaum, 1987, 1998; Robins et al., 2000; Lunceford and Davidian, 2004). These methods commonly require modeling the treatment assignment, which can be difficult in some applications. To protect against misspecification of the treatment model, various methods have been proposed (e.g., Robins et al., 1994; Scharfstein et al., 1999; Bang and Robins, 2005). Among them, doubly robust methods are often advocated since the resulting estimators are still consistent when either the treatment model or the outcome model (but not both) is misspecified; such an attractive property is referred to as double robustness.

Although many methods are available for causal inference such as for the estimation of average treatment effects (ATE), those methods are vulnerable to poor quality data. Typically when data are error-contaminated, most existing methods would be inapplicable. It has been well documented that measurement error in variables can often lead to seriously biased inference results in many settings (e.g., Fuller, 1987; Gustafson, 2003; Carroll et al., 2006; Buonaccorsi, 2010; Yi, 2017).

In the context of causal inference with error-prone data, there has been limited but increasing research on the impact of measurement error on causal inference and the development of correction methods to deal with measurement error. For instances, McCaffrey et al. (2013) proposed a correction estimation method when baseline covariates are error-prone. Babanezhad et al. (2010) examined the bias arising from ignoring misclassification in the treatment variable. Braun et al. (2016) developed a correction method to correct for treatment misclassification using validation data.

In settings with misclassification in the binary outcome variable, Shu and Yi (2017) explored the estimation of ATE using the inverse probability weighted (IPW) method. They derived the asymptotic bias caused by misclassification and developed consistent estimation methods to eliminate the misclassification effects. Their development covers practical scenarios where (1) the misclassification probabilities are known, or (2) the misclassification probabilities are unknown but validation data or replicates of outcome measurements are available for their estimation. They further propose a doubly robust estimator to provide protection against possible misspecification of the treatment model.

The methods developed by Shu and Yi (2017) enjoy wide applications, because misclassified binary outcome data arise commonly in practice. For example, the self-reported smoking status without being confirmed by biochemical tests is subject to misclassification; results of screening tests are often subject to false positive error and/or false negative error. For datasets with outcome misclassification, ignoring misclassification effects may lead to severely biased results. To expedite the application of the correction methods for general users, we develop an R package, called **ipwErrorY** (Shu and Yi, 2019), to implement the methods by Shu and Yi (2017) for practical settings where the commonly-used logistic regression model is employed for the treatment model and the outcome model. The package focuses on measurement error in the *outcome* Y only but not on other types of measurement error, such as measurement error in covariates.

The remainder is organized as follows. First, we introduce the notation and framework. Secondly, we describe the methods to be implemented in R. Thirdly, we present the implementation steps and illustrate the use of the package with examples. Finally, a discussion is given. The developed R package **ipwErrorY** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=ipwErrorY`.

Notation and framework

Let Y_1 be the binary potential outcome that would have been observed had the individual been treated, and Y_0 be the binary potential outcome that would have been observed had the individual been untreated. Let X be the vector of baseline covariates; T be the observed binary treatment variable; and Y be the observed binary outcome.

Being consistent with the usual causal inference framework (e.g., Lunceford and Davidian, 2004), we make the following standard assumptions:

Assumption 1 (Consistency): $Y = TY_1 + (1-T)Y_0$;

Assumption 2 (No Unmeasured Confounding): $(Y_1, Y_0) \perp\!\!\!\perp T|X$;

Assumption 3 (Positivity): $0 < P(T = 1|X) < 1$.

The objective is to estimate the ATE, defined as $\tau_0 = E(Y_1) - E(Y_0)$. Suppose we have a sample of size n. For $i = 1, \cdots, n$, we add subscript i to notations X, T and Y to denote the corresponding variable for individual i.

In the presence of outcome misclassification, instead of Y, its surrogate version Y^* is observed. We assume that

$$P(Y^* = a|Y = b, X, T) = P(Y^* = a|Y = b) \quad \text{for} \quad a, b = 0, 1. \tag{V.2.1}$$

That is, conditional on the true value of the outcome, the misclassification probability is assumed to be homogeneous for all the individuals, regardless of their covariate information or treatment status. For ease of exposition, write $p_{ab} = P(Y^* = a|Y = b)$. Then the sensitivity and the specificity are expressed by p_{11} and p_{00}, respectively, and the misclassification probabilities are $p_{01} = 1 - p_{11}$ and $p_{10} = 1 - p_{00}$.

Estimation methods

In this section we present estimation methods for τ_0 under three scenarios. We first start with the case where misclassification probabilities are given, and then consider settings where misclassification probabilities are unknown but can be estimated by using additional data sources.

Estimation with known misclassification probabilities

In this subsection we assume that misclassification probabilities p_{11} and p_{10} are known. For $i = 1, \ldots, n$, let $e_i = P(T_i = 1|X_i)$ be the conditional probability for individual i to receive the treatment, also termed as the propensity score (e.g., Rosenbaum and Rubin, 1983), a quantity that plays an important role in causal inference. To correct for outcome misclassification effects, Shu and Yi (2017) proposed an estimator of τ_0 given by

$$\widehat{\tau} = \frac{1}{p_{11} - p_{10}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \frac{T_i Y_i^*}{\widehat{e}_i} - \frac{1}{n} \sum_{i=1}^{n} \frac{(1-T_i) Y_i^*}{1 - \widehat{e}_i} \right\}, \tag{V.3.1}$$

where $\widehat{e}_i$ is an estimate of the propensity score $e_i = P(T_i = 1|X_i)$ obtained by fitting the treatment model relating T to X.

The estimator $\widehat{\tau}$, given by (V.3.1), is a consistent estimator of τ_0, provided regularity conditions including Assumptions 1-3. The sandwich variance estimate of $\widehat{\tau}$ can be obtained using the theory of estimating functions (e.g., Newey and McFadden, 1994; Heyde, 1997; Yi, 2017, Ch.1).

To estimate the propensity score e_i for $i = 1, \ldots, n$, we specifically characterize e_i using the widely-used logistic regression model. That is, the treatment model is given by

$$\text{logit } P(T_i = 1|X_i) = \gamma_0 + \gamma_X^\top X_i, \tag{V.3.2}$$

for $i = 1, \ldots, n$, where $\gamma = (\gamma_0, \gamma_X^\top)^\top$ is the vector of parameters. As a result, an unbiased estimating function of γ is taken as the score function

$$\left\{T_i - \frac{1}{1+\exp(-\gamma_0 - \gamma_X^\top X_i)}\right\}(1, X_i^\top)^\top. \tag{V.3.3}$$

Let $\theta = (\tau, \gamma^\top)^\top$. Shu and Yi (2017) showed that

$$\Psi(Y_i^*, T_i, X_i; \theta) = \begin{pmatrix} \left\{T_i - \dfrac{1}{1+\exp(-\gamma_0 - \gamma_X^\top X_i)}\right\}(1, X_i^\top)^\top \\ \dfrac{T_i Y_i^*}{e_i} - \dfrac{(1-T_i)Y_i^*}{1-e_i} - (p_{11} - p_{10})\tau \end{pmatrix} \tag{V.3.4}$$

is an unbiased estimating function of θ. Solving $\sum_{i=1}^n \Psi(Y_i^*, T_i, X_i; \theta) = 0$ for θ yields an estimator of θ, denoted by $\widehat{\theta}$.

Let $\theta_0 = (\tau_0, \gamma_0^\top)^\top$ be the true value of θ. Define $A(\theta_0) = E\left\{-(\partial/\partial\theta^\top)\Psi(Y^*, T, X; \theta)|_{\theta=\theta_0}\right\}$ and $B(\theta_0) = E\{\Psi(Y^*, T, X; \theta)\Psi^\top(Y^*, T, X; \theta)|_{\theta=\theta_0}\}$. Under regularity conditions, we have that

$$\sqrt{n}(\widehat{\theta} - \theta_0) \xrightarrow{d} N\left(0, A(\theta_0)^{-1}B(\theta_0)A(\theta_0)^{-1\,\top}\right) \quad \text{as } n \to \infty. \tag{V.3.5}$$

Consequently, the variance of $\widehat{\theta}$ can be estimated by the empirical sandwich estimator:

$$\widehat{Var}(\widehat{\theta}) = \frac{1}{n} A_n(\widehat{\theta})^{-1} B_n(\widehat{\theta}) A_n(\widehat{\theta})^{-1\,\top}, \tag{V.3.6}$$

where

$$A_n(\widehat{\theta}) = -\frac{1}{n}\sum_{i=1}^n \frac{\partial}{\partial\theta^\top}\Psi(Y_i^*, T_i, X_i; \theta)|_{\theta=\widehat{\theta}} \tag{V.3.7}$$

and

$$B_n(\widehat{\theta}) = \frac{1}{n}\sum_{i=1}^n \Psi(Y_i^*, T_i, X_i; \theta)\Psi^\top(Y_i^*, T_i, X_i; \theta)|_{\theta=\widehat{\theta}}\,. \tag{V.3.8}$$

Then the variance estimate of $\widehat{\tau}$ in (V.3.1) is given by $\widehat{Var}(\widehat{\tau}) = \widehat{v}_{11}$, where $\widehat{v}_{11}$ is the element of the first row and the first column of $\widehat{Var}(\widehat{\theta})$. A $(1-\alpha)100\%$ confidence interval of τ_0 is given by $\widehat{\tau} \pm z_{\alpha/2} \times \sqrt{\widehat{Var}(\widehat{\tau})}$, where α is a specified value between 0 and 1, and $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of the standard normal distribution.

Estimation with validation data

In this subsection we assume that misclassification probabilities p_{11} and p_{10} are unknown and that there is an internal validation subsample $\mathcal{V}$ of size n_V which collects measurements of variables X, T, Y and Y^*.

With the validation data, p_{11} and p_{10} can be estimated as

$$\widehat{p}_{11} = \frac{\sum_{i\in\mathcal{V}} Y_i Y_i^*}{\sum_{i\in\mathcal{V}} Y_i} \quad \text{and} \quad \widehat{p}_{10} = \frac{\sum_{i\in\mathcal{V}}(1-Y_i)Y_i^*}{\sum_{i\in\mathcal{V}}(1-Y_i)}, \tag{V.3.9}$$

respectively.

To estimate τ_0, one may use error-free outcome data of the validation subsample to construct an estimator

$$\widehat{\tau}_\text{V} = \frac{1}{n_\text{V}}\sum_{i\in\mathcal{V}}\frac{T_i Y_i}{\widehat{e}_i} - \frac{1}{n_\text{V}}\sum_{i\in\mathcal{V}}\frac{(1-T_i)Y_i}{1-\widehat{e}_i}, \tag{V.3.10}$$

of τ_0, or alternatively, one may apply (V.3.1) to estimate τ_0 using non-validation data with the resulting estimator given by

$$\widehat{\tau}_\text{N} = \frac{1}{\widehat{p}_{11} - \widehat{p}_{10}}\left\{\frac{1}{n-n_\text{V}}\sum_{i\notin\mathcal{V}}\frac{T_i Y_i^*}{\widehat{e}_i} - \frac{1}{n-n_\text{V}}\sum_{i\notin\mathcal{V}}\frac{(1-T_i)Y_i^*}{1-\widehat{e}_i}\right\}, \tag{V.3.11}$$

where $\widehat{p}_{11}$ and $\widehat{p}_{10}$ are given by (V.3.9).

Although the validation data based estimator $\widehat{\tau}_{\mathrm{V}}$ given by (V.3.10) and the non-validation data based estimator $\widehat{\tau}_{\mathrm{N}}$ given by (V.3.11) are both consistent estimators of τ_0, they both incur efficiency loss due to the inability of utilizing all the available data.

Shu and Yi (2017) considered the linear combination of $\widehat{\tau}_{\mathrm{V}}$ and $\widehat{\tau}_{\mathrm{N}}$

$$\widehat{\tau}(c) = c\widehat{\tau}_{\mathrm{V}} + (1-c)\widehat{\tau}_{\mathrm{N}}, \tag{V.3.12}$$

where c is a constant between 0 and 1.

For any c, the consistency of $\widehat{\tau}(c)$ is immediate due to the consistency of $\widehat{\tau}_{\mathrm{V}}$ and $\widehat{\tau}_{\mathrm{N}}$. However, the efficiency of $\widehat{\tau}(c)$ depends on the choice of c. Typically, $Var\{\widehat{\tau}(c)\}$ is minimized at

$$c_{\mathrm{OPT}} = \frac{Var(\widehat{\tau}_{\mathrm{N}}) - Cov(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})}{Var(\widehat{\tau}_{\mathrm{V}}) + Var(\widehat{\tau}_{\mathrm{N}}) - 2Cov(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})}, \tag{V.3.13}$$

suggesting that $\widehat{\tau}(c_{\mathrm{OPT}}) = c_{\mathrm{OPT}}\widehat{\tau}_{\mathrm{V}} + (1-c_{\mathrm{OPT}})\widehat{\tau}_{\mathrm{N}}$ is the optimal estimator among the linear combination estimators formulated as (V.3.12). Furthermore, c_{OPT} can be estimated by

$$\widehat{c}_{\mathrm{OPT}} = \frac{\widehat{Var}(\widehat{\tau}_{\mathrm{N}}) - \widehat{Cov}(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})}{\widehat{Var}(\widehat{\tau}_{\mathrm{V}}) + \widehat{Var}(\widehat{\tau}_{\mathrm{N}}) - 2\widehat{Cov}(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})}, \tag{V.3.14}$$

where $\widehat{Var}(\widehat{\tau}_{\mathrm{N}})$, $\widehat{Cov}(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})$ and $\widehat{Var}(\widehat{\tau}_{\mathrm{V}})$ are the estimates for $Var(\widehat{\tau}_{\mathrm{N}})$, $Cov(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})$ and $Var(\widehat{\tau}_{\mathrm{V}})$, respectively.

To obtain $\widehat{Var}(\widehat{\tau}_{\mathrm{N}})$, $\widehat{Cov}(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})$ and $\widehat{Var}(\widehat{\tau}_{\mathrm{V}})$, Shu and Yi (2017) constructed an unbiased estimating function by combining the estimating functions (V.3.10) and (V.3.11), where they introduced different symbols, say τ_{V} and τ_{N}, to denote the parameter τ for which (V.3.10) and (V.3.11), respectively, are used to estimate; both τ_{V} and τ_{N} have the true value τ_0. Let $\theta = (\tau_{\mathrm{V}}, \tau_{\mathrm{N}}, \gamma^\top, p_{11}, p_{10})^\top$. Define

$$\Psi_c(Y_i^*, T_i, X_i, Y_i; \theta) = \begin{pmatrix} \left\{T_i - \dfrac{1}{1+\exp(-\gamma_0 - \gamma_X^\top X_i)}\right\}(1, X_i^\top)^\top \\ (Y_iY_i^* - p_{11}Y_i)\cdot I(i \in \mathcal{V}) \cdot \dfrac{n}{n_{\mathrm{V}}} \\ \{(1-Y_i)Y_i^* - p_{10}(1-Y_i)\}\cdot I(i \in \mathcal{V}) \cdot \dfrac{n}{n_{\mathrm{V}}} \\ \left\{\dfrac{T_iY_i}{e_i} - \dfrac{(1-T_i)Y_i}{1-e_i} - \tau_{\mathrm{V}}\right\}\cdot I(i \in \mathcal{V}) \cdot \dfrac{n}{n_{\mathrm{V}}} \\ \left\{\dfrac{T_iY_i^*}{e_i} - \dfrac{(1-T_i)Y_i^*}{1-e_i} - (p_{11}-p_{10})\tau_{\mathrm{N}}\right\} I(i \notin \mathcal{V}) \cdot \dfrac{n}{n-n_{\mathrm{V}}} \end{pmatrix}, \tag{V.3.15}$$

where $I(\cdot)$ is the indicator function. Then $\Psi_c(Y_i^*, T_i, X_i, Y_i; \theta)$ is an unbiased combined estimating function of θ. Solving $\sum_{i=1}^n \Psi_c(Y_i^*, T_i, X_i, Y_i; \theta) = 0$ for θ yields an estimator of θ, denoted by $\widehat{\theta}$. The variance of $\widehat{\theta}$ can be estimated by the empirical sandwich estimator, denoted as $\widehat{Var}(\widehat{\theta})$. Let $\widehat{v}_{i,j}$ be the element of the ith row and the jth column of $\widehat{Var}(\widehat{\theta})$. Then $\widehat{Var}(\widehat{\tau}_{\mathrm{V}}) = \widehat{v}_{1,1}$, $\widehat{Cov}(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}}) = \widehat{v}_{1,2}$, and $\widehat{Var}(\widehat{\tau}_{\mathrm{N}}) = \widehat{v}_{2,2}$.

Finally, Shu and Yi (2017) pointed out the two associated conditions: $Var(\widehat{\tau}_{\mathrm{V}}) + Var(\widehat{\tau}_{\mathrm{N}}) - 2Cov(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}}) \geqslant 0$ and $0 \leqslant c \leqslant 1$. If one or both conditions are violated with empirical estimates, $\widehat{c}_{\mathrm{OPT}}$ is then set to be 1 if $\widehat{\tau}_{\mathrm{V}}$ has smaller variance than $\widehat{\tau}_{\mathrm{N}}$ and 0 otherwise. The resulting optimal linear combination estimator $\widehat{\tau}(\widehat{c}_{\mathrm{OPT}})$ is

$$\widehat{\tau}_{\mathrm{OPT}} = \widehat{c}_{\mathrm{OPT}}\widehat{\tau}_{\mathrm{V}} + (1-\widehat{c}_{\mathrm{OPT}})\widehat{\tau}_{\mathrm{N}}, \tag{V.3.16}$$

with the variance estimate given by

$$\widehat{Var}(\widehat{\tau}_{\mathrm{OPT}}) = \{\widehat{Var}(\widehat{\tau}_{\mathrm{V}}) + \widehat{Var}(\widehat{\tau}_{\mathrm{N}}) - 2\widehat{Cov}(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})\}\widehat{c}_{\mathrm{OPT}}^2 - \{2\widehat{Var}(\widehat{\tau}_{\mathrm{N}}) - 2\widehat{Cov}(\widehat{\tau}_{\mathrm{V}}, \widehat{\tau}_{\mathrm{N}})\}\widehat{c}_{\mathrm{OPT}} + \widehat{Var}(\widehat{\tau}_{\mathrm{N}}). \tag{V.3.17}$$

A $(1-\alpha)100\%$ confidence interval of τ_0 is given by $\widehat{\tau}_{\mathrm{OPT}} \pm z_{\alpha/2} \times \sqrt{\widehat{Var}(\widehat{\tau}_{\mathrm{OPT}})}$, where α is a specified value between 0 and 1, and $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of the standard normal distribution.

Estimation with replicates

In this subsection we assume that misclassification probabilities p_{11} and p_{10} are unknown and that two repeated outcome measurements are available for each individual. Suppose $Y_i^*(1)$ and $Y_i^*(2)$ are two independent replicates of Y_i. Let η denote the prevalence $P(Y=1)$ and π_r be the probability of obtaining r outcome observations equal to 1 among two repeated outcome measurements for $r=0,1$. Then

$$\pi_0 = \eta(1-p_{11})^2 + (1-\eta)(1-p_{10})^2; \tag{V.3.18}$$

$$\pi_1 = 2\eta(1-p_{11})p_{11} + 2(1-\eta)(1-p_{10})p_{10}. \tag{V.3.19}$$

Let $\theta = (\tau, \gamma^\top, \eta, p_{11}, p_{10})^\top$. Shu and Yi (2017) considered an unbiased estimating function of θ, given by

$$\Psi_r\{Y_i^*(1), Y_i^*(2), T_i, X_i; \theta\} = \begin{pmatrix} \left\{T_i - \dfrac{1}{1+\exp(-\gamma_0 - \gamma_X^\top X_i)}\right\}(1, X_i^\top)^\top \\ \\ \{1-Y_i^*(1)\}\cdot\{1-Y_i^*(2)\} - \pi_0 \\ \\ Y_i^*(1)\cdot\{1-Y_i^*(2)\} + Y_i^*(2)\cdot\{1-Y_i^*(1)\} - \pi_1 \\ \\ \dfrac{T_i Y_i^*}{e_i} - \dfrac{(1-T_i)Y_i^*}{1-e_i} - (p_{11}-p_{10})\tau \end{pmatrix}, \tag{V.3.20}$$

where $Y_i^* = \{Y_i^*(1) + Y_i^*(2)\}/2$, together with a constraint imposed for achieving parameters identifiability (e.g., White et al., 2001; Yi and He, 2017).

Let $\widehat{\tau}_{\mathrm{R}}$ denote the estimator of τ_0 obtained by solving

$$\sum_{i=1}^{n} \Psi_r\{Y_i^*(1), Y_i^*(2), T_i, X_i; \theta\} = 0 \tag{V.3.21}$$

for θ. The variance of $\widehat{\tau}_{\mathrm{R}}$ can be estimated by the empirical sandwich estimator $\widehat{Var}(\widehat{\tau}_{\mathrm{R}})$. A $(1-\alpha)100\%$ confidence interval of τ_0 is given by $\widehat{\tau}_{\mathrm{R}} \pm z_{\alpha/2} \times \sqrt{\widehat{Var}(\widehat{\tau}_{\mathrm{R}})}$, where α is a specified value between 0 and 1, and $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of the standard normal distribution.

Finally, we comment that when implementing (V.3.21), one of the following constraints is often used in applications: (C1) sensitivity equals specificity (i.e., $p_{11} = p_{00}$), (C2) sensitivity p_{11} is known, (C3) specificity p_{00} is known, and (C4) prevalence η is known. These four constraints are implemented in our R package.

Choosing a suitable identifiability constraint is primarily driven by the nature of data. When the false positive rate p_{10} and the false negative rate p_{01} are close, it is reasonable to impose the constraint that the sensitivity equals the specificity. When there is prior information on the value of the sensitivity, the specificity, or the prevalence, it is plausible to add the identifiability constraint (C2), (C3) or (C4). For example, in smoking cessation studies, patients who quit smoking (with $Y=1$) are unlikely to report that they still smoke, so it may be reasonable to set the constraint $p_{11}=1$. Sometimes, researchers may use the disease prevalence reported from another similar study for their own study, when such a prevalence is perceived to be close to that of the target population.

Doubly robust estimation

To protect against model misspecification, Shu and Yi (2017) proposed a doubly robust estimator of τ_0:

$$\widehat{\tau}_{\mathrm{DR}} = \widehat{E}(Y_1) - \widehat{E}(Y_0), \tag{V.3.22}$$

where

$$\widehat{E}(Y_1) = \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{T_i Y_i^*}{\widehat{e}_i(p_{11}-p_{10})} - \frac{T_i - \widehat{e}_i}{\widehat{e}_i}\widehat{q}_{i1} - \frac{T_i}{\widehat{e}_i}\left(\frac{p_{10}}{p_{11}-p_{10}}\right)\right\}, \tag{V.3.23}$$

$$\widehat{E}(Y_0) = \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{(1-T_i)Y_i^*}{(1-\widehat{e}_i)(p_{11}-p_{10})} + \frac{T_i - \widehat{e}_i}{1-\widehat{e}_i}\widehat{q}_{i0} - \frac{1-T_i}{1-\widehat{e}_i}\left(\frac{p_{10}}{p_{11}-p_{10}}\right)\right\}, \tag{V.3.24}$$

$\widehat{q}_{i1}$ is an estimate of $q_{i1} = P(Y_i = 1|T_i = 1, X_i)$ and $\widehat{q}_{i0}$ is an estimate of $q_{i0} = P(Y_i = 1|T_i = 0, X_i)$.

The estimator $\widehat{\tau}_{\text{DR}}$ enjoys the double robustness property in the sense that it is still consistent if one of the treatment model and the outcome model is incorrectly specified. In our developed R package, we particularly implement the following two scenarios.

Scenario 1 (Shared covariate effects for the treated and untreated groups):

Suppose the outcome model is postulated as

$$\text{logit } P(Y_i = 1|T_i, X_i) = \beta_0 + \beta_{\text{T}} T_i + \beta^\top X_i, \tag{V.3.25}$$

where β_0, β_{T} and β are the parameters. The model reflects the setting where the treated and untreated groups share the same covariate effect β on the outcome.

By (V.2.1) and (V.3.25), the observed likelihood function contributed from individual i is

$$\begin{aligned} & L_i(\beta_0, \beta_{\text{T}}, \beta) \\ = & P(Y_i^*|X_i, T_i) \\ = & P(Y_i = 1|X_i, T_i)P(Y_i^*|X_i, T_i, Y_i = 1) + P(Y_i = 0|X_i, T_i)P(Y_i^*|X_i, T_i, Y_i = 0) \\ = & \frac{1}{1 + \exp\{-\beta_0 - \beta_{\text{T}} T_i - \beta^\top X_i\}} \cdot \{p_{11} Y_i^* + (1 - p_{11})(1 - Y_i^*)\} \\ & + \frac{\exp\{-\beta_0 - \beta_{\text{T}} T_i - \beta^\top X_i\}}{1 + \exp\{-\beta_0 - \beta_{\text{T}} T_i - \beta^\top X_i\}} \cdot \{p_{10} Y_i^* + (1 - p_{10})(1 - Y_i^*)\}. \end{aligned} \tag{V.3.26}$$

With regularity conditions, maximizing the observed likelihood $\prod_{i=1}^n L_i(\beta_0, \beta_{\text{T}}, \beta)$ with respect to $(\beta_0, \beta_{\text{T}}, \beta^\top)^\top$ gives a consistent estimator of $(\beta_0, \beta_{\text{T}}, \beta^\top)^\top$, denoted as $(\widehat{\beta}_0, \widehat{\beta}_{\text{T}}, \widehat{\beta}^\top)^\top$. It follows that q_{i1} and q_{i0}, are, respectively, estimated by

$$\widehat{q}_{i1} = \frac{1}{1 + \exp(-\widehat{\beta}_0 - \widehat{\beta}_{\text{T}} - \widehat{\beta}^\top X_i)} \tag{V.3.27}$$

and

$$\widehat{q}_{i0} = \frac{1}{1 + \exp(-\widehat{\beta}_0 - \widehat{\beta}^\top X_i)}. \tag{V.3.28}$$

Scenario 2 (Possibly different covariate effects for the treated and untreated groups):

Suppose that the outcome model is postulated as

$$\text{logit } P(Y_i = 1|T_i = 1, X_i) = \beta_{01} + \beta_1^\top X_i \tag{V.3.29}$$

for the treated group and

$$\text{logit } P(Y_i = 1|T_i = 0, X_i) = \beta_{00} + \beta_0^\top X_i \tag{V.3.30}$$

for the untreated group, where the parameters $(\beta_{01}, \beta_1^\top)^\top$ for the treated group may differ from the parameters $(\beta_{00}, \beta_0^\top)^\top$ for the untreated group.

To obtain a consistent estimator $(\widehat{\beta}_{01}, \widehat{\beta}_1^\top)^\top$ of $(\beta_{01}, \beta_1^\top)^\top$ and a consistent estimator $(\widehat{\beta}_{00}, \widehat{\beta}_0^\top)^\top$ of $(\beta_{00}, \beta_0^\top)^\top$, we employ the observed likelihood for the treated group and the untreated group separately. For example, the observed likelihood function contributed from individual l in the treated group (i.e., $T_l = 1$) is

$$\begin{aligned} & L_{1,l}(\beta_{01}, \beta_1) \\ = & P(Y_l^*|T_l = 1, X_l) \\ = & P(Y_l = 1|T_l = 1, X_l)P(Y_l^*|Y_l = 1) + P(Y_l = 0|T_l = 1, X_l)P(Y_l^*|Y_l = 0) \\ = & \frac{1}{1 + \exp\{-\beta_{01} - \beta_1^\top X_l\}} \cdot \{p_{11} Y_l^* + (1 - p_{11})(1 - Y_l^*)\} \\ & + \frac{\exp\{-\beta_{01} - \beta_1^\top X_l\}}{1 + \exp\{-\beta_{01} - \beta_1^\top X_l\}} \cdot \{p_{10} Y_l^* + (1 - p_{10})(1 - Y_l^*)\}. \end{aligned} \tag{V.3.31}$$

Maximizing the observed likelihood $\prod_{l:T_l=1} L_{1,l}(\beta_{01}, \beta_1)$ with respect to β_{01} and β_1 gives us a consistent estimator $(\widehat{\beta}_{01}, \widehat{\beta}_1^\top)^\top$, provided regularity conditions. Similarly, we calculate the observed likelihood function $L_{0,k}(\beta_{00}, \beta_0)$ for individual k in the untreated group (i.e., $T_k = 0$), and then obtain the estimator $(\widehat{\beta}_{00}, \widehat{\beta}_0^\top)^\top$ by maximizing the observed likelihood $\prod_{l:T_k=0} L_{0,k}(\beta_{00}, \beta_0)$ with respect to β_{00} and β_0. Thus, q_{i1} and q_{i0} are estimated by

$$\widehat{q}_{i1} = \frac{1}{1 + \exp(-\widehat{\beta}_{01} - \widehat{\beta}_1^\top X_i)} \tag{V.3.32}$$

and

$$\widehat{q}_{i0} = \frac{1}{1 + \exp(-\widehat{\beta}_{00} - \widehat{\beta}_0^\top X_i)}, \tag{V.3.33}$$

respectively.

Variance estimator of $\widehat{\tau}_{\text{DR}}$:

Consistency and asymptotic normality of $\widehat{\tau}_{\text{DR}}$ can be established using the theory of estimating functions. Below we derive the sandwich variance estimator of $\widehat{\tau}_{\text{DR}}$ by constructing an unbiased estimating function using the "delta method" in the M-estimator framework (Stefanski and Boos, 2002).

Define β_{F} to be the vector of parameters for the outcome models. Under Scenario 1 with shared covariate effects for the treated and untreated groups, $\beta_{\text{F}} = (\beta_0, \beta_{\text{T}}, \beta^\top)^\top$. Under Scenario 2 with possibly different covariate effects for the treated and untreated groups, $\beta_{\text{F}} = (\beta_{\text{F1}}^\top, \beta_{\text{F0}}^\top)^\top$ with $\beta_{\text{F1}} = (\beta_{01}, \beta_1^\top)^\top$ and $\beta_{\text{F0}} = (\beta_{00}, \beta_0^\top)^\top$. Let $\theta = (\gamma^\top, \beta_{\text{F}}^\top, \mu_1, \mu_0, \tau)^\top$, where μ_1 and μ_0 represent $E(Y_1)$ and $E(Y_0)$, respectively.

We construct the following unbiased estimating function for θ:

$$\Psi_{dr}\{Y_i^*, T_i, X_i; \theta\} = \begin{pmatrix} \left\{T_i - \dfrac{1}{1+\exp(-\gamma_0 - \gamma_X^\top X_i)}\right\}(1, X_i^\top)^\top \\ \psi(Y_i^*, T_i, X_i; \beta_{\text{F}}) \\ \left\{\dfrac{T_i Y_i^*}{\widehat{e}_i(p_{11} - p_{10})} - \dfrac{T_i - \widehat{e}_i}{\widehat{e}_i}\widehat{q}_{i1} - \dfrac{T_i}{\widehat{e}_i}\left(\dfrac{p_{10}}{p_{11} - p_{10}}\right)\right\} - \mu_1 \\ \left\{\dfrac{(1-T_i) Y_i^*}{(1-\widehat{e}_i)(p_{11} - p_{10})} + \dfrac{T_i - \widehat{e}_i}{1-\widehat{e}_i}\widehat{q}_{i0} - \dfrac{1-T_i}{1-\widehat{e}_i}\left(\dfrac{p_{10}}{p_{11} - p_{10}}\right)\right\} - \mu_0 \\ \mu_1 - \mu_0 - \tau \end{pmatrix}, \tag{V.3.34}$$

where $\psi(Y_i^*, T_i, X_i; \beta_{\text{F}})$ is the unbiased estimating equation for β_{F} derived from the observed likelihood. Specifically, under Scenario 1,

$$\psi(Y_i^*, T_i, X_i; \beta_{\text{F}}) = \partial log\{L_i(\beta_0, \beta_{\text{T}}, \beta)\}/\partial(\beta_0, \beta_{\text{T}}, \beta), \tag{V.3.35}$$

and under Scenario 2,

$$\psi(Y_i^*, T_i, X_i; \beta_{\text{F}}) = (\psi_1(Y_i^*, T_i, X_i; \beta_{\text{F1}})^\top, \psi_0(Y_i^*, T_i, X_i; \beta_{\text{F0}})^\top)^\top \tag{V.3.36}$$

with

$$\psi_1(Y_i^*, T_i, X_i; \beta_{\text{F1}}) = \partial log\{L_{1,i}(\beta_{01}, \beta_1)\}/\partial(\beta_{01}, \beta_1) \cdot I(T_i = 1) \cdot \frac{n}{n_{\text{T}}} \tag{V.3.37}$$

and

$$\psi_0(Y_i^*, T_i, X_i; \beta_{\text{F0}}) = \partial log\{L_{0,i}(\beta_{00}, \beta_0)\}/\partial(\beta_{00}, \beta_0) \cdot I(T_i = 0) \cdot \frac{n}{n - n_{\text{T}}}, \tag{V.3.38}$$

where n_{T} is the size of the treated group.

By the theory of estimating functions (e.g., Newey and McFadden, 1994; Heyde, 1997; Yi, 2017, Ch.1), solving $\sum_{i=1}^n \Psi_{dr}\{Y_i^*, T_i, X_i; \theta\} = 0$ for θ yields an estimator of θ, denoted by $\widehat{\theta}$. Let θ_0 be the true value of θ. Define

$$A(\theta_0) = E\left\{-(\partial/\partial\theta^\top)\Psi_{dr}(Y^*, T, X; \theta)|_{\theta=\theta_0}\right\}$$

and

$B(\theta_0) = E\{\Psi_{dr}(Y^*, T, X; \theta)\Psi_{dr}^\top(Y^*, T, X; \theta)|_{\theta=\theta_0}\}.$

Under regularity conditions, we have that

$$\sqrt{n}(\widehat{\theta} - \theta_0) \xrightarrow{d} N\left(0, A(\theta_0)^{-1} B(\theta_0) A(\theta_0)^{-1\top}\right) \text{ as } n \to \infty. \quad \text{(V.3.39)}$$

The sandwich variance estimator of $\widehat{\theta}$ is given by

$$\widehat{Var}(\widehat{\theta}) = \frac{1}{n} A_n(\widehat{\theta})^{-1} B_n(\widehat{\theta}) A_n(\widehat{\theta})^{-1\top}, \quad \text{(V.3.40)}$$

where

$$A_n(\widehat{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial\theta^\top}\Psi_{dr}(Y_i^*, T_i, X_i; \theta)|_{\theta=\widehat{\theta}} \quad \text{(V.3.41)}$$

and

$$B_n(\widehat{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \Psi_{dr}(Y_i^*, T_i, X_i; \theta)\Psi_{dr}^\top(Y_i^*, T_i, X_i; \theta)|_{\theta=\widehat{\theta}}\,. \quad \text{(V.3.42)}$$

Then $\widehat{Var}(\widehat{\tau}_{\text{DR}})$ is the element of the last row and the last column of $\widehat{Var}(\widehat{\theta})$.

Finally, we comment that Scenario 2 allows for different covariates-outcome associations for the treated and untreated groups and provides more flexibility than Scenario 1. However, implementing Scenario 2 involves separately estimating parameters of the outcome model for the treated and untreated groups. When one group has a small size, estimation results may be unsatisfactory. In this case, imposing common covariate effects for the treated and untreated groups as in Scenario 1 can help achieve reasonable estimation results.

Implementation in R and Examples

We develop an R package **ipwErrorY**, which implements the methods (Shu and Yi, 2017) described in the previous section. The developed package imports R packages **stats** and **nleqslv** (Hasselman, 2016). To illustrate the use of **ipwErrorY**, for each method, we simulate a dataset and then apply a function to analyze the dataset. To make sure users can reproduce the results, we use the function `set.seed` to generate data. Moreover, the simulated data provide users a clear sense about the data structure.

Implementation and example with known error

The function `KnownError` produces the ATE estimate using the correction method with known misclassification probabilities along with the sandwich-variance-based standard error and $(1-\alpha)100\%$ confidence interval. Specifically, `KnownError` is defined as

```
KnownError(data, indA, indYerror, indX, sensitivity, specificity, confidence=0.95)
```

with arguments described in detail in **ipwErrorY** documentation. Below is an example to illustrate the use of `KnownError`.

We first load the package in R:

```
R> library("ipwErrorY")
```

Using sensitivity 0.95 and specificity 0.85, we create `da`, a dataset of size 2000 with “`X1`”, “`A`” and “`Yast`” being the column names for the covariate, treatment and misclassified outcome, respectively:

```
R> set.seed(100)
R> X1 = rnorm(2000)
R> A = rbinom(2000, 1, 1/(1 + exp(-0.2 - X1)))
R> Y = rbinom(2000, 1, 1/(1 + exp(-0.2 - A - X1)))
R> y1 = which(Y == 1)
R> y0 = which(Y == 0)
R> Yast = Y
R> Yast[y1] = rbinom(length(y1), 1, 0.95)
R> Yast[y0] = rbinom(length(y0), 1, 0.15)
R> da = data.frame(X1 = X1, A = A,Yast = Yast)
```

By using the function `head`, we print the first six observations of dataset `da` so that the data structure is clearly shown as follows:

```
R> head(da)
          X1 A Yast
1 -0.50219235 1    1
2  0.13153117 1    1
3 -0.07891709 1    1
4  0.88678481 0    1
5  0.11697127 1    1
6  0.31863009 1    1
```

We call the developed function KnownError with sensitivity 0.95 and specificity 0.85, and obtain a list of the estimate, the sandwich-variance-based standard error and a 95% confidence interval:

```
R> KnownError(data = da, indA = "A", indYerror = "Yast", indX = "X1",
+     sensitivity = 0.95, specificity = 0.85, confidence=0.95)
$Estimate
[1] 0.1702513

$Std.Error
[1] 0.02944824

$`95% Confidence Interval`
[1] 0.1125338 0.2279688
```

Implementation and example with validation data

The function EstValidation produces the results for the method with validation data; they include the optimal linear combination estimate, the sandwich-variance-based standard error, $(1-\alpha)100\%$ confidence interval and the estimated sensitivity and specificity. Specifically, EstValidation is defined as

```
EstValidation(maindata, validationdata, indA, indYerror, indX, indY, confidence=0.95)
```

with arguments described in detail in **ipwErrorY** documentation. Below is an example to illustrate the use of EstValidation.

Using sensitivity 0.95 and specificity 0.85, we create mainda which is the non-validation main data of size 1200, and validationda which is the validation data of size 800:

```
R> set.seed(100)
R> X1= rnorm(1200)
R> A = rbinom(1200, 1, 1/(1 + exp(-0.2 - X1)))
R> Y= rbinom(1200, 1, 1/(1 + exp(-0.2 - A - X1)))
R> y1 = which(Y == 1)
R> y0 = which(Y==0)
R> Yast = Y
R> Yast[y1] = rbinom(length(y1), 1, 0.95)
R> Yast[y0] = rbinom(length(y0), 1, 0.15)
R> mainda = data.frame(A = A, X1 = X1, Yast = Yast)
R> X1 = rnorm(800)
R> A = rbinom(800, 1, 1/(1 + exp(-0.2 - X1)))
R> Y = rbinom(800, 1, 1/(1 + exp(-0.2 - A - X1)))
R> y1 = which(Y == 1)
R> y0 = which(Y == 0)
R> Yast = Y
R> Yast[y1] = rbinom(length(y1), 1, 0.95)
R> Yast[y0] = rbinom(length(y0), 1, 0.15)
R> validationda = data.frame(A = A, X1 = X1, Y = Y, Yast = Yast)
```

We print the first six observations of non-validation data mainda and validation data validationda:

```
R> head(mainda)
  A          X1 Yast
1 1 -0.50219235    0
2 0  0.13153117    0
3 1 -0.07891709    1
4 1  0.88678481    1
5 0  0.11697127    1
```

```
6 1  0.31863009    1
R> head(validationda)
  A           X1 Y Yast
1 0 -0.0749961081 0    0
2 1 -0.9470827924 1    1
3 1  0.0003758095 1    1
4 0 -1.5249574007 0    0
5 1  0.0983516474 0    0
6 0 -1.5266078213 1    1
```

The preceding output clearly reveals that the non-validation data and validation data differ in the data structure. The non-validation data `mainda` record measurements of the treatment, covariate and misclassified outcome, indicated by the column names "`A`", "`X1`" and "`Yast`", respectively. In comparison, the validation data `validationda` record measurements of the treatment, covariate, misclassified outcome and the true outcome, indicated by the column names "`A`", "`X1`", "`Yast`", and "`Y`", respectively.

To apply the optimal linear combination method with validation data, we call the developed function `EstValidation` and obtain a list of the estimate, the sandwich-variance-based standard error, a 95% confidence interval, and the estimated sensitivity and specificity:

```
R> EstValidation(maindata = mainda, validationdata = validationda, indA = "A",
+     indYerror = "Yast",  indX = "X1",  indY = "Y", confidence=0.95)
$Estimate
[1] 0.1714068

$Std.Error
[1] 0.02714957

$`95% Confidence Interval`
[1] 0.1181946 0.2246189

$`estimated sensitivity and estimated specificity`
[1] 0.9482072 0.8557047
```

Implementation and example with replicates

The function `Est2Replicates` produces the results for the method with replicates; they include the estimate, the sandwich-variance-based standard error, $(1-\alpha)100\%$ confidence interval, and the imposed constraint(s), and the information on sensitivity and specificity. Specifically, `Est2Replicates` is defined as

```
Est2Replicates(data, indA, indYerror, indX, constraint=c("sensitivity equals
  specificity", "known sensitivity", "known specificity", "known prevalence"),
  sensitivity = NULL, specificity = NULL, prevalence = NULL, confidence=0.95)
```

with arguments described in detail in **ipwErrorY** documentation. Below is an example to illustrate the use of `Est2Replicates`.

Using sensitivity 0.95 and specificity 0.85, we create `da`, a dataset of size 2000 with "`A`", "`X1`", and {"`Yast1`", "`Yast2`"} being the column names for the treatment, covariate, and two replicates of outcome, respectively:

```
R> set.seed(100)
R> X1 = rnorm(2000)
R> A = rbinom(2000, 1, 1/(1 + exp(-0.2 - X1)))
R> Y = rbinom(2000, 1, 1/(1 + exp(-0.2 - A - X1)))
R> y1 = which(Y == 1)
R> y0 = which(Y == 0)
R> Yast1 = Y
R> Yast1[y1] = rbinom(length(y1), 1, 0.95)
R> Yast1[y0] = rbinom(length(y0), 1, 0.15)
R> Yast2 = Y
R> Yast2[y1] = rbinom(length(y1), 1, 0.95)
R> Yast2[y0] = rbinom(length(y0), 1, 0.15)
R> da = data.frame(A = A, X1 = X1, Yast1 = Yast1, Yast2 = Yast2)
```

By using the function head, we print the first six observations of dataset da so that the data structure is clearly shown as follows:

```
R> head(da)
  A          X1 Yast1 Yast2
1 1 -0.50219235     1     1
2 1  0.13153117     1     1
3 1 -0.07891709     1     1
4 0  0.88678481     1     0
5 1  0.11697127     1     1
6 1  0.31863009     1     1
```

To apply the method with replicates, we call the developed function Est2Replicates with the imposed constraint that specificity equals 0.85. The following list of the estimate, the sandwich-variance-based standard error, a 95% confidence interval, the imposed constraint and the information on sensitivity and specificity is returned:

```
R> Est2Replicates(data = da,  indA = "A", indYerror = c("Yast1", "Yast2"),
+     indX = "X1", constraint = "known specificity", sensitivity = NULL,
+     specificity = 0.85, prevalence = NULL, confidence=0.95)
$Estimate
[1] 0.1908935

$Std.Error
[1] 0.02687287

$`95% Confidence Interval`
[1] 0.1382236 0.2435634

$`imposed constraint`
[1] "known specificity"

$`estimated sensitivity and assumed specificity`
[1] 0.95 0.85
```

Implementation and example of doubly robust estimation

The function KnownErrorDR produces the ATE estimate using the doubly robust correction method along with the sandwich-variance-based standard error and $(1-\alpha)100\%$ confidence interval. Specifically, KnownErrorDR is defined as

```
KnownErrorDR(data, indA, indYerror, indXtrt, indXout, sensitivity, specificity,
  sharePara=FALSE, confidence=0.95)
```

with arguments described in detail in **ipwErrorY** documentation. Below is an example to illustrate the use of KnownErrorDR.

Using sensitivity 0.95 and specificity 0.85, we create da, a dataset of size 2000 with “A”, {“X”, “xx”} and “Yast” being the column names for the treatment, covariates and misclassified outcome, respectively:

```
R> set.seed(100)
R> X = rnorm(2000)
R> xx = X^2
R> A = rbinom(2000, 1, 1/(1 + exp(-0.1 - X - 0.2*xx)))
R> Y = rbinom(2000, 1, 1/(1 + exp(1 - A - 0.5*X - xx)))
R> y1 = which(Y == 1)
R> y0 = which(Y == 0)
R> Y[y1] = rbinom(length(y1), 1, 0.95)
R> Y[y0] = rbinom(length(y0), 1, 0.15)
R> Yast = Y
R> da = data.frame(A = A, X = X, xx = xx, Yast = Yast)
```

By using the function head, we print the first six observations of dataset da so that the data structure is clearly shown as follows:

```
R> head(da)
  A           X          xx Yast
```

```
1 1 -0.50219235 0.252197157    1
2 1  0.13153117 0.017300447    1
3 1 -0.07891709 0.006227907    1
4 0  0.88678481 0.786387298    0
5 1  0.11697127 0.013682278    1
6 1  0.31863009 0.101525133    0
```

When applying the doubly robust method with sensitivity 0.95 and specificity 0.85, covariates indicated by column names "X" and "xx" are both included in the treatment model and the outcome model. Let the outcome model be fit for the treated and untreated groups separately. We call the developed function KnownErrorDR and obtain a list of the estimate, the sandwich-variance-based standard error, and a 95% confidence interval:

```
R> KnownErrorDR(data = da, indA = "A", indYerror = "Yast", indXtrt = c("X", "xx"),
+    indXout = c("X", "xx"), sensitivity = 0.95, specificity = 0.85,
+    sharePara = FALSE, confidence=0.95)
$Estimate
[1] 0.2099162

$Std.Error
[1] 0.02811472

$`95% Confidence Interval`
[1] 0.1548124 0.2650201
```

Discussion

Misclassified binary outcome data arise frequently in practice and present a challenge in conducting causal inference. Discussion on addressing this issue is rather limited in the literature. Shu and Yi (2017) developed the IPW estimation methods for ATE with mismeasured outcome effects incorporated. To expedite the application of these correction methods, we develop an R package **ipwErrorY**. For practical settings where the treatment model and the outcome model are specified as logistic regression models, we implement the correction methods developed by Shu and Yi (2017) for settings with known misclassification probabilities, validation data, or replicates of the outcome data as well as the doubly robust method with known misclassification probabilities. Our package offers a useful and convenient tool for data analysts to perform valid inference about ATE when the binary outcome variable is subject to misclassification.

For each function of **ipwErrorY**, we implement the sandwich variance estimate to construct a normality-based confidence interval. Confidence intervals can also be constructed by bootstrapping (Efron, 1982; Efron and Tibshirani, 1993), which can be done by leveraging available functions of **ipwErrorY**. Below we provide example code to produce normality-based and percentile-based bootstrap confidence intervals for a doubly robust estimate with 200 bootstrap replicates.

```
R> drFUN<-function(dt) {
+   KnownErrorDR(data = dt, indA = "A",  indYerror = "Yast", indXtrt = c("X", "xx"),
+                indXout = c("X", "xx"), sensitivity = 0.95, specificity = 0.85,
+                sharePara = FALSE, confidence=0.95)$`Estimate`
+ }
R> EST=drFUN(dt=da)
R> set.seed(100)
R> resultsBoot=replicate(200,drFUN(dt=da[sample(1:nrow(da),replace=TRUE),]))
R> STD=sd(resultsBoot)
R> lowN=EST-qnorm(1-(1-0.95)/2)*STD
R> upN=EST+qnorm(1-(1-0.95)/2)*STD
R> CIn=c(lowN,upN)
R> lowP=as.numeric(quantile(resultsBoot,probs=0.025))
R> upP=as.numeric(quantile(resultsBoot,probs=0.975))
R> CIp=c(lowP,upP)
```

We print the resultant bootstrap normality-based and percentile-based confidence intervals, respectively, as follows.

```
R> CIn
[1] 0.1562355 0.2635969
```

```
R> CIp
[1] 0.1610038 0.2655065
```

To make sure the users can reproduce the results, here we call the function `set.seed` before `KnownErrorDR`. If `set.seed` is not used, then the variance estimates generated at different times can differ due to the inner randomness of the bootstrap method.

This example code can be easily modified to produce bootstrap confidence intervals for an estimate obtained from a different method; one needs only to replace `KnownErrorDR` with the function in **ipwErrorY** that corresponds to the method.

Package **ipwErrorY** requires the data be complete (i.e., no missing values). An error message is shown when NAs in the dataset are detected. For example, if we artificially introduce an NA in dataset `da` and call the developed function `KnownErrorDR`, an error message is displayed:

```
R> da[1,1]=NA
R> KnownErrorDR(data = da, indA = "A",  indYerror = "Yast", indXtrt = c("X", "xx"),
+              indXout = c("X", "xx"), sensitivity = 0.95, specificity = 0.85,
+              sharePara = FALSE, confidence=0.95)
Error in KnownErrorDR(data = da, indA = "A", indYerror = "Yast", indXtrt = c("X",  :
  invalid dataset with NAs (missing data detected)
```

Once seeing this error message, users need to check their dataset to see if the NAs can be replaced with suitable values. If missing values do occur, the easiest way is to take the subsample of complete observations to conduct analysis. The resulting point estimates can be reasonable if the missing data mechanism is missing completely at random (MCAR) (Little and Rubin, 2002); in this instance, efficiency loss can occur. However, when missing data are not MCAR, this procedure often yields biased results.

Our implementation uses a logistic regression model with a linear function of covariates for both the treatment and the outcome processes, as it is perhaps the most widely-used parametric tool to model a binary variable. Such a logistic regression model can be generalized to include additional terms, such as higher order terms, nonlinear functions, or interactions of the covariates. In this case, the users need only to first create an expanded dataset with those terms included as additional columns of new "covariates", and then use the **ipwErrorY** package to analyze the expanded dataset.

Acknowledgments

The authors thank the editor and two reviewers for their helpful comments and suggestions which improved the manuscript. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and was partially supported by a Collaborative Research Team Project of Canadian Statistical Sciences Institute (CANSSI).

Bibliography

M. Babanezhad, S. Vansteelandt, and E. Goetghebeur. Comparison of causal effect estimators under exposure misclassification. *Journal of Statistical Planning and Inference*, 140:1306–1319, 2010. URL https://doi.org/10.1016/j.jspi.2009.11.015. [p337]

H. Bang and J. M. Robins. Doubly robust estimation in missing data and causal inference models. *Biometrics*, 61:962–973, 2005. URL https://doi.org/10.1111/j.1541-0420.2005.00377.x. [p337]

D. Braun, C. Zigler, F. Dominici, and M. Gorfine. Using validation data to adjust the inverse probability weighting estimator for misclassified treatment. *Harvard University Biostatistics Working Paper Series. Working Paper 201*, pages 1–19, 2016. [p337]

J. P. Buonaccorsi. *Measurement Error: Models, Methods, and Applications.* Chapman & Hall/CRC, London, 2010. [p337]

R. J. Carroll, D. Ruppert, L. A. Stefanski, and C. M. Crainiceanu. *Measurement Error in Nonlinear Models: A Modern Perspective, 2nd Edition.* Chapman & Hall/CRC, Boca Raton, 2006. [p337]

B. Efron. *The Jackknife, the Bootstrap and Other Resampling Plans*, volume 38. SIAM, Philadelphia, 1982. [p348]

B. Efron and R. Tibshirani. *An Introduction to the Bootstrap.* London: Chapman & Hall/CRC, 1993. [p348]

W. A. Fuller. *Measurement Error Models.* John Wiley & Sons, 1987. [p337]

P. Gustafson. *Measurement Error and Misclassification in Statistics and Epidemiology.* Chapman & Hall/CRC, London, 2003. [p337]

B. Hasselman. *Nleqslv: Solve Systems of Nonlinear Equations*, 2016. URL https://CRAN.R-project.org/package=nleqslv. R package version 3.0. [p344]

M. A. Hernán and J. M. Robins. *Causal Inference.* Chapman & Hall/CRC, Boca Raton, forthcoming, 2019. [p337]

C. C. Heyde. *Quasi-Likelihood and Its Application: A General Approach to Optimal Parameter Estimation.* Springer-Verlag, 1997. [p338, 343]

G. W. Imbens and D. B. Rubin. *Causal Inference in Statistics, Social, and Biomedical Sciences.* Cambridge University Press, New York, 2015. [p337]

R. J. Little and D. B. Rubin. *Statistical Analysis with Missing Data, 2nd Edition.* John Wiley & Sons, 2002. [p349]

J. K. Lunceford and M. Davidian. Stratification and weighting via the propensity score in estimation of causal treatment effects: A comparative study. *Statistics in Medicine*, 23:2937–2960, 2004. URL https://doi.org/10.1002/sim.1903. [p337, 338]

D. F. McCaffrey, J. Lockwood, and C. M. Setodji. Inverse probability weighting with error-prone covariates. *Biometrika*, 100:671–680, 2013. URL https://doi.org/10.1093/biomet/ast022. [p337]

W. K. Newey and D. McFadden. Large sample estimation and hypothesis testing. *Handbook of Econometrics*, 4:2111–2245, 1994. [p338, 343]

J. M. Robins, A. Rotnitzky, and L. P. Zhao. Estimation of regression coefficients when some regressors are not always observed. *Journal of the American Statistical Association*, 89:846–866, 1994. URL https://doi.org/10.2307/2290910. [p337]

J. M. Robins, M. A. Hernán, and B. Brumback. Marginal structural models and causal inference in epidemiology. *Epidemiology*, 11:550–560, 2000. [p337]

P. R. Rosenbaum. Model-based direct adjustment. *Journal of the American Statistical Association*, 82:387–394, 1987. URL https://doi.org/10.2307/2289440. [p337]

P. R. Rosenbaum. Propensity score. *In Encyclopedia of Biostatistics*, 5:3551–3555, 1998. [p337]

P. R. Rosenbaum and D. B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70:41–55, 1983. URL https://doi.org/10.1093/biomet/70.1.41. [p337, 338]

K. J. Rothman, S. Greenland, and T. L. Lash. *Modern Epidemiology, 3rd Edition.* Lippincott Williams & Wilkins, Philadelphia, 2008. [p337]

D. O. Scharfstein, A. Rotnitzky, and J. M. Robins. Adjusting for nonignorable drop-out using semiparametric nonresponse models. *Journal of the American Statistical Association*, 94:1096–1120, 1999. URL https://doi.org/10.2307/2669930. [p337]

D. Shu and G. Y. Yi. Causal inference with measurement error in outcomes: Bias analysis and estimation methods. *Statistical Methods in Medical Research*, 2017. URL https://doi.org/10.1177/0962280217743777. [p337, 338, 339, 340, 341, 344, 348]

D. Shu and G. Y. Yi. *ipwErrorY: Inverse Probability Weighted Estimation of Average Treatment Effect with Misclassified Binary Outcome*, 2019. URL https://CRAN.R-project.org/package=ipwErrorY. R package version 2.1. [p337]

L. A. Stefanski and D. D. Boos. The calculus of m-estimation. *The American Statistician*, 56:29–38, 2002. URL https://doi.org/10.1198/000313002753631330. [p343]

I. White, C. Frost, and S. Tokunaga. Correcting for measurement error in binary and continuous variables using replicates. *Statistics in Medicine*, 20:3441–3457, 2001. URL https://doi.org/10.1002/sim.908. [p341]

G. Y. Yi. *Statistical Analysis with Measurement Error or Misclassification: Strategy, Method and Application.* Springer-Verlag, 2017. [p337, 338, 343]

G. Y. Yi and W. He. Analysis of case-ontrol data with interacting misclassified covariates. *Journal of Statistical Distributions and Applications*, 4:1–16, 2017. URL https://doi.org/10.1186/s40488-017-0069-0. [p341]

Di Shu
Department of Statistics and Actuarial Science
University of Waterloo
Waterloo N2L 3G1, Ontario, Canada
shudi1991@gmail.com

Grace Y. Yi
Department of Statistics and Actuarial Science
University of Waterloo
Waterloo N2L 3G1, Ontario, Canada
yyi@uwaterloo.ca

optimParallel: An R Package Providing a Parallel Version of the L-BFGS-B Optimization Method

by Florian Gerber and Reinhard Furrer

Abstract The R package **optimParallel** provides a parallel version of the L-BFGS-B optimization method of `optim()`. The main function of the package is `optimParallel()`, which has the same usage and output as `optim()`. Using `optimParallel()` can significantly reduce the optimization time, especially when the evaluation time of the objective function is large and no analytical gradient is available. We introduce the R package and illustrate its implementation, which takes advantage of the lexical scoping mechanism of R.

Introduction

Many statistical tools involve optimization algorithms, which aim to find the minima or maxima of an objective function $fn : \mathbb{R}^p \rightarrow \mathbb{R}$, where $p \in \mathbb{N}$ denotes the number of parameters. Depending on the specific application different optimization algorithms may be preferred; see the book by Nash (2014), the special issue of the Journal of Statistical Software (Varadhan, 2014), and the CRAN Task View *Optimization* for overviews of the optimization software available for R. A widely used algorithm is the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm with box constraints (L-BFGS-B, Byrd et al., 1996), which is available through the general-purpose optimizer `optim()` of the R package **stats** and the more recent R packages **lbfgsb3** (Nash et al., 2015) and **lbfgsb3c** (Fidler et al., 2018). The L-BFGS-B algorithm has proven to work well in numerous applications. However, long optimization times of computationally intensive functions sometimes hinder its application; see Gerber et al. (2017) for an example of such a function from our research in spatial statistics. For this reason we present a parallel version of the `optim()` L-BFGS-B algorithm, denoted with `optimParallel()`, and explore its potential to reduce optimization times.

To illustrate the possible speed gains of a parallel L-BFGS-B implementation let $gr : \mathbb{R}^p \rightarrow \mathbb{R}^p$ denote the gradient of $fn()$. L-BFGS-B always first evaluates $fn()$ and then $gr()$ at the same parameter value and we call one such sequential evaluation one *step*. Note that this step should not be confused with the *iteration* as defined in Byrd et al. (1996) and used in the help page of `optim()`, because the latter may involve several steps. Let T_{fn} and T_{gr} denote the evaluation times of $fn()$ and $gr()$, respectively. In the case where $gr()$ is specified in the `optim()` call, one step of the L-BFGS-B algorithm evaluates $fn()$ and $gr()$ sequentially, and hence, the evaluation time is little more than $T_{fn} + T_{gr}$ per step. In contrast, `optimParallel()` evaluates both functions in parallel, which reduces the evaluation time to little more than $\max\{T_{fn}, T_{gr}\}$ per step. In the case where no gradient is provided, `optim()` calculates a numeric central difference gradient approximation (CGA). For $p = 1$ the CGA is defined as

$$\widetilde{gr}(x) = (fn(x+\epsilon) - fn(x-\epsilon))\,/\,2\,\epsilon, \qquad \text{(X.1.1)}$$

and hence, requires two evaluations of $fn()$. Similarly, calculating $\widetilde{gr}()$ requires $2p$ evaluations of $fn()$ if $fn()$ has p parameters. In total, `optim()` sequentially evaluates $fn()$ $1 + 2p$ times per step, resulting in an elapsed time of little more than $(1 + 2p)T_{fn}$ per step. Given $1 + 2p$ available processor cores, `optimParallel()` evaluates all calls of $fn()$ in parallel, which reduces the elapsed time to little more than T_{fn} per step.

`optimParallel()` by examples

The main function of the R package **optimParallel** (Gerber, 2019) is `optimParallel()`, which has the same usage and output as `optim()`, but evaluates $fn()$ and $gr()$ in parallel. For illustration, consider 10^7 samples from a normal distribution with mean $\mu = 5$ and standard deviation $\sigma = 2$. Then, we define the following negative log-likelihood and use `optim()` to estimate the parameters μ and σ:[1]

[1] It is obvious that simpler ways exists to estimate μ and σ. Moreover, a computationally more efficient version of `negll()` can be constructed based on `mean()` and `sd()`.

```
> x <- rnorm(n = 1e7, mean = 5, sd = 2)
> negll <- function(par, x) -sum(dnorm(x = x, mean = par[1], sd = par[2], log = TRUE))
> o1 <- optim(par = c(1, 1), fn = negll, x = x, method = "L-BFGS-B",
+             lower = c(-Inf, 0.0001))
> o1$par
[1] 5.000597 2.000038
```

Using optimParallel(), we can do the same in parallel. The functions makeCluster(), detectCores(), and setDefaultCluster() from the R package **parallel** are used to set up a default cluster for the parallel execution.

```
> install.packages("optimParallel")
> library("optimParallel")
Loading required package: parallel
> cl <- makeCluster(detectCores()); setDefaultCluster(cl = cl)
> o2 <- optimParallel(par = c(1, 1), fn = negll, x = x, lower = c(-Inf, 0.0001))
> identical(o1, o2)
[1] TRUE
```

On our computer with 12 Intel Xeon E5-2640 @ 2.50GHz processors one evaluation of negll() took 0.9 seconds. optim() run with 6.2 seconds per step, whereas optimParallel() run with 1.8 seconds per step. Thus, the optimization time of optimParallel() is reduced by 71% compared to that of optim(). Note that for $p = 2$, 71% is below the maximal possible reduction of $1 - 1/(2p + 1) = 80\%$ because of the overhead associated with the parallel execution and the time needed to run the L-BFGS-B algorithm, which are both not taken into account in this calculation. In general, the reduction of the optimization time is large if the parallel overhead is small relative to the execution time of $fn()$. Hence, for this example, the reduction of the optimization time approaches 80% when the evaluation time of negll() is increased, e. g., by increasing the number of data points in x.

In addition to the arguments of optim(), optimParallel() has the argument parallel, which takes a list of arguments. For example, we can set parallel = list(loginfo = TRUE) to store all evaluated parameters and the corresponding gradients.

```
> o3 <- optimParallel(par = c(1, 1), fn = negll, x = x, lower = c(-Inf, 0.0001),
+                     parallel = list(loginfo = TRUE))
> head(o3$loginfo, n = 3)
     step     par1     par2        fn       gr1        gr2
[1,]    1 1.000000 1.000000 109213991 -40005963 -190049608
[2,]    2 1.205988 1.978554  39513324  -9693283  -18700810
[3,]    3 1.265626 2.086455  37160791  -8579638  -14969646
> tail(o3$loginfo, n = 3)
      step     par1     par2       fn         gr1       gr2
[16,]   16 5.000840 2.000140 21121045 609.9480540 507.56421
[17,]   17 5.000586 2.000041 21121045 -26.8237162  15.17266
[18,]   18 5.000597 2.000038 21121045   0.6494038  -1.67717
```

This can be used to visualize the optimization path as shown in Figure 1 and simplifies the following study, which illustrates the impact of using different (approximate) gradient specifications.

In the optimParallel() calls above the argument gr was not specified, and hence, the CGA was used. Another way of using optimParallel() is with a gradient function given to the argument gr. If the computation of the analytical gradient is tractable and does not take more time than evaluating $fn()$, this usage is preferred.

```
> negll_gr <- function(par, x){
+     sm <- mean(x); n <- length(x)
+     c(-n*(sm-par[1])/par[2]^2,
+       n/par[2] - (sum((x-sm)^2) + n*(sm-par[1])^2)/par[2]^3)
+ }
> o4 <- optimParallel(par = c(1, 1), fn = negll, gr = negll_gr, x = x,
+                     lower = c(-Inf, 0.0001), parallel = list(loginfo = TRUE))
> tail(o4$loginfo, n = 3)
      step     par1     par2       fn         gr1        gr2
[16,]   16 5.000840 2.000139 21121045 609.9651113 507.625076
[17,]   17 5.000586 2.000041 21121045 -26.8233339  15.172072
[18,]   18 5.000597 2.000037 21121045   0.6494452  -1.677113
```

We see that the resulting optimization path is very similar to that based on the CGA (o3 above).

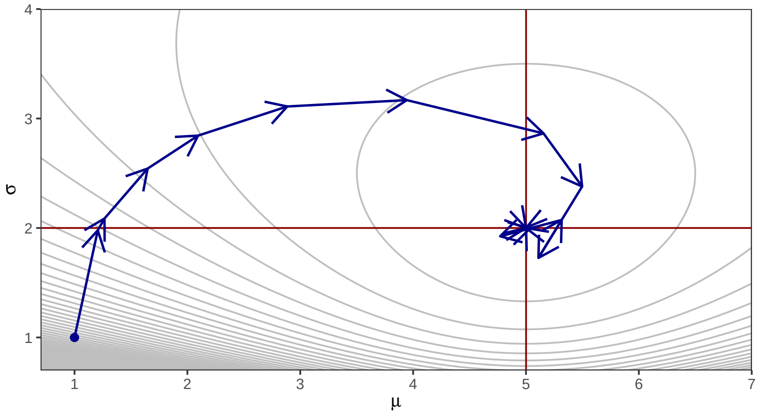


Figure 1: Visualization of the optimization path based on the log information obtained with `optimParallel(..., parallel = list(loginfo = TRUE))`. The red lines mark the estimates $\hat{\mu}$ and $\hat{\sigma}$.

Besides the CGA and the option to explicitly specify a gradient function, `optimParallel()` provides a third option, namely the numeric forward difference gradient approximation (FGA) defined as $\widetilde{gr}(x) = (fn(x+\epsilon) - fn(x))/\epsilon$ for $p = 1$ and x sufficiently away from the boundaries. Using the FGA, the return value of $fn(x)$ can be reused for the computation of the gradient, and hence, the number of evaluations of $fn()$ is reduced to $1 + p$ per step. This can be helpful if the number of available cores is less than $1 + 2p$ or insufficient memory is available to run $1 + 2p$ evaluations of $fn()$ in parallel.

```
> o5 <- optimParallel(par = c(1, 1), fn = negll, x = x, lower = c(-Inf, 0.0001),
+                      parallel = list(loginfo = TRUE, forward = TRUE))
> o5$loginfo[17:19, ]
     step     par1     par2       fn       gr1       gr2
[1,]   17 5.000086 1.999541 21121046 -26.33029  14.35781
[2,]   18 5.000331 1.999645 21121045 586.89953 534.85303
[3,]   19 5.000346 1.999651 21121045 625.10421 567.27441
```

```
> tail(o5$loginfo, n = 3)
      step     par1     par2       fn      gr1      gr2
[31,]   31 5.000347 1.999652 21121045 627.8436 569.5991
[32,]   32 5.000347 1.999652 21121045 627.8436 569.5991
[33,]   33 5.000347 1.999652 21121045 627.8436 569.5991
```

We see that the optimizer only stopped after 33 steps, whereas all previous optimization calls stopped after 18 steps. Hence, it is obvious that the choice of the gradient approximation affects the optimization. But what happened exactly?—It should be noted that the FGA is less accurate than the CGA; see, e. g., the numerical study in Nash (2014), Section 10.7. Hence, small differences in the optimization path are expected, but this does hardly explain the additional 15 steps used by the FGA based optimization. A closer inspection of the optimization path reveals that up to step 18 the path is very similar to those of the previous optimization calls and the remaining steps 19–33 only marginally change `par1` and `par2`. This suggests that using the FGA prevents the algorithm from stopping. One way to handle this issue is to set a less restrictive stopping criterion by increasing the value of `factr`.

```
> o6 <- optimParallel(par = c(1, 1), fn = negll, x = x, lower = c(-Inf, 0.0001),
+                      parallel = list(loginfo = TRUE, forward = TRUE),
+                      control = list(factr = 1e-6/.Machine$double.eps))
> tail(o6$loginfo, n = 3)
      step     par1     par2       fn        gr1        gr2
[14,]   14 4.996680 2.001974 21121074 -8524.7678 12125.5022
[15,]   15 4.999743 1.998478 21121052  -884.4955 -5305.2516
[16,]   16 5.000347 1.999652 21121045   627.8436   569.5991
```

Now the resulting optimization path and the evaluation counts are similar to those resulting from the optimization using the analytic gradient (`o4` above). The take-home message of this study is that the choice of the (approximate) gradient can affect the optimization path and it may be necessary to adjust control parameters such as `factr`, `ndeps`, and `parscale` to obtain satisfactory results. A more detailed discussion of the use of (approximate) gradients in optimization can be found in Nash (2014), Chapter 10.

Implementation

`optimParallel()` is a wrapper to `optim()` and enables the parallel evaluation of all function calls involved in one step of the L-BFGS-B optimization method. It is implemented in R and interfaces compiled C code only through `optim()`. The reuse of the stable and well-tested C code of `optim()` has the advantage that `optimParallel()` leads to the exact same results as `optim()`. To ensure that `optimParallel()` and `optim()` indeed return the same results **optimParallel** contains systematic unit tests implemented with the R package **testthat** (Wickham, 2017, 2011).

The basic idea of the implementation is that calling `fn()` (or `gr()`) triggers the evaluation of both `fn()` and `gr()`. Their return values are stored in a local environment. The next time `fn()` (or `gr()`) is called with the same parameters the results are read from the local environment without evaluating `fn()` and `gr()` again. The following R code illustrates how `optimParallel()` takes advantage of the lexical scoping mechanism of R to store the return values of `fn()` and `gr()`.

```
> demo_generator <- function(fn, gr) {
+     par_last <- value <- grad <- NA
+     eval <- function(par) {
+         if(!identical(par, par_last)) {
+             message("--> evaluate fn() and gr()")
+             par_last <<- par
+             value <<- fn(par)
+             grad <<- gr(par)
+         } else
+             message("--> read stored value")
+     }
+     f <- function(par) {
+         eval(par = par)
+         value
+     }
+     g <- function(par) {
+         eval(par = par)
+         grad
+     }
+     list(fn = f, gr = g)
+ }
> demo <- demo_generator(fn = sum, gr = prod)
```

Calling `demo$fn()` triggers the evaluation of both `fn()` and `gr()`.

```
> demo$fn(1:5)
--> evaluate fn() and gr()
[1] 15
```

The subsequent call of `demo$gr()` with the same parameters returns the stored value `grad` without evaluating `gr()` again.

```
> demo$gr(1:5)
--> read stored value
[1] 120
```

A similar construct allows `optimParallel()` to evaluate $fn()$ and $gr()$ at the same occasion. It is then straightforward to parallelize the evaluations of the functions using the R package **parallel**.

Speed gains and performance test

To illustrate the speed gains that can be achieved with `optimParallel()` we measure the elapsed times per step when optimizing the following test function and compare them to those of `optim()`.

```
> fn <- function(par, sleep) {
+     Sys.sleep(sleep)
+     sum(par^2)
```

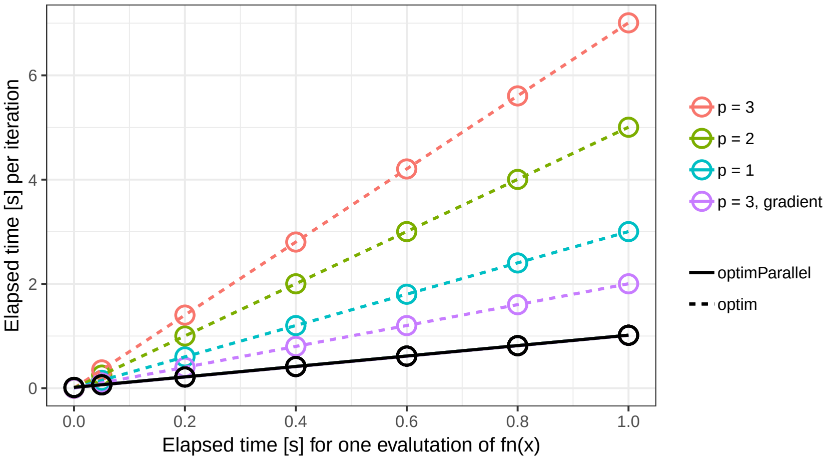


Figure 2: Benchmark experiment comparing the L-BFGS-B method from `optimParallel()` and `optim()`. Plotted are the elapsed times per step (y-axis) and the evaluation time of $fn()$ (x-axis) for $p = 1$, 2, and 3 using an approximate gradient and $p = 3$ using an analytic gradient. The elapsed times of `optimParallel()` (solid line) are independent of p and the specification of an analytic gradient.

```
+ }
> gr <- function(par, sleep) {
+     Sys.sleep(sleep)
+     2*par
+ }
```

In both functions the argument `par` can be a numeric vector with one or more elements and the argument `sleep` controls the evaluation time of the functions. We measure the elapsed time per step using all combinations of $p = 1, 2, 3$, `sleep`= 0, 0.05, 0.2, 0.4, 0.6, 0.8, 1 seconds with and without analytic gradient `gr()`. All measurements are taken on a computer with 12 Intel Xeon E5-2640 @ 2.50GHz processors. However, because of the experimental design a maximum of 7 processors are used in parallel. We repeat each measurement 5 times using the R package **microbenchmark** (Mersmann et al., 2018). The complete R script of the benchmark experiment is contained in **optimParallel**.

The results of the benchmark experiment are summarized in Figure 2. They show that for `optimParallel()` the elapsed time per step is only marginally larger than T_{fn} (black circles in Figure 2). Conversely, the elapsed time for `optim()` is $T_{fn} + T_{gr}$ if a gradient function is specified (violet circles) and $(1 + 2p)T_{fn}$ if no gradient function is specified (red, green, and blue circles). Moreover, `optimParallel()` adds a small overhead, and hence, it is only faster than `optim()` for T_{fn} larger than 0.05 seconds.

The use of `Sys.sleep()` in this illustration is useful to characterize the implementation and its overhead. However, it does not represent a practical use case of `optimParallel()` and the speed gains might be less pronounced for other examples. One factor that reduces the speed of `optimParallel()` is the specification of large objects in its `"..."` argument. All those objects are copied to the running R sessions in the cluster, which increases the elapsed time. Related to that is the increased memory usage, which may slowdown the optimization when not enough memory is available.

Summary

The R package **optimParallel** provides a parallel version of the L-BFGS-B optimization method of `optim()`. After a brief theoretical illustration of the possible speed improvement based on parallel processing, we illustrate `optimParallel()` by examples. The examples demonstrate that one can replace `optim()` by `optimParallel()` to execute the optimization in parallel and illustrate additional features like capturing log information and using different (approximate) gradients. Moreover, we briefly sketch the basic idea of the implementation, which is based on the lexical scoping mechanism of R. Finally, a performance test shows that using `optimParallel()` reduces the elapsed time to optimize computationally demanding functions significantly. For functions with evaluation times of more than 0.1 seconds we measured speed gains of about factor 2 in the case where an analytic gradient was specified and about factor $1 + 2p$ otherwise (p is the number of parameters). Our results suggest that using `optimParallel()` is most beneficial when (i) the evaluation time of $fn()$ is large (more than 0.1 seconds), (ii) no analytical gradient is available, and (iii) p or more processors as well as enough memory are available.

Bibliography

R. H. Byrd, L. Peihuang, and J. Nocedal. A limited-memory algorithm for bound-constrained optimization. *Technical Report*, 1996. URL https://doi.org/10.2172/204262. [p352]

M. L. Fidler, J. C. Nash, C. Zhu, R. Byrd, J. Nocedal, and J. L. Morales. *Lbfgsb3c: Limited Memory BFGS Minimizer with Bounds on Parameters with Optim() 'C' Interface*, 2018. URL https://CRAN.R-project.org/package=lbfgsb3c. R package version 2018-2.13-1. [p352]

F. Gerber. *optimParallel: A Parallel Version of the L-BFGS-B Method Optim()*, 2019. URL https://CRAN.R-project.org/package=optimParallel. R package version 0.8. [p352]

F. Gerber, K. Mösinger, and R. Furrer. Extending R packages to support 64-Bit compiled code: An illustration with spam64 and GIMMS NDVI$_{3g}$ data. *Computers & Geosciences*, 104:109–119, 2017. URL https://doi.org/10.1016/j.cageo.2016.11.015. [p352]

O. Mersmann, C. Beleites, R. Hurling, A. Friedman, and J. M. Ulrich. *Microbenchmark: Accurate Timing Functions*, 2018. URL https://CRAN.R-project.org/package=microbenchmark. R package version 1.4-4. [p356]

J. C. Nash. *Nonlinear Parameter Optimization Using R Tools*. John Wiley & Sons, 2014. ISBN 9781118883969. URL https://doi.org/10.1002/9781118884003. [p352, 354, 355]

J. C. Nash, C. Zhu, R. Byrd, J. Nocedal, and J. L. Morales. *Lbfgsb3: Limited Memory BFGS Minimizer with Bounds on Parameters*, 2015. URL https://CRAN.R-project.org/package=lbfgsb3. R package version 2015-2.13. [p352]

R. Varadhan. Special volume: Numerical optimization in R: Beyond optim. *Journal of Statistical Software*, 6, 2014. ISSN 1548-7660. URL https://www.jstatsoft.org/issue/view/v060. [p352]

H. Wickham. Testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL http://journal.R-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf. [p355]

H. Wickham. *Testthat: Unit Testing for R*, 2017. URL https://CRAN.R-project.org/package=testthat. R package version 2.0.0. [p355]

Dr. Florian Gerber
Department of Applied Mathematics and Statistics
Colorado School of Mines, Colorado, USA
gerber@mines.edu, *https://orcid.org/0000-0001-8545-5263*

Prof. Dr. Reinhard Furrer
Department of Mathematics & Department of Computational Science
University of Zurich, Switzerland
reinhard.furrer@math.uzh.ch, *https://orcid.org/0000-0002-6319-2332*

Fixed Point Acceleration in R

by Stuart Baumann, Margaryta Klymak

Abstract A fixed point problem is one where we seek a vector, X, for a function, f, such that f(X) = X. The solution of many such problems can be accelerated by using a fixed point acceleration algorithm. With the release of the **FixedPoint** package there is now a number of algorithms available in **R** that can be used for accelerating the finding of a fixed point of a function. These algorithms include Newton acceleration, Aitken acceleration and Anderson acceleration as well as epsilon extrapolation methods and minimal polynomial methods. This paper demonstrates the use of fixed point accelerators in solving numerical mathematics problems using the algorithms of the **FixedPoint** package as well as the squarem method of the **SQUAREM** package.

Introduction

R has had a number of packages providing optimisation algorithms for many years. These include traditional optimisers through the `optim()` function, genetic algorithms through the **rgenoud** package (Mebane, Jr. and Sekhon, 2011) and response surface global optimisers through packages like **DiceKriging** (Roustant et al., 2012). It also has several rootfinders like the `uniroot()` method and the methods of the **BB** package (Varadhan and Gilbert, 2009).

Fixed point accelerators are conceptually similar to both optimisation and root finding algorithms but thus far implementations of fixed point finders have been rare in **R**. Prior to **FixedPoint**'s (Baumann and Klymak, 2018) release the squarem method of the **SQUAREM** package[1] (Varadhan, 2010) was the only effective fixed point acceleration algorithm available in **R**.[2] In some part this is likely because there is often an obvious method to find a fixed point by merely feeding a guessed fixed point into a function, taking the result and feeding it back into the function. By doing this repeatedly a fixed point is often found. This method (that we will call the "Simple" method) is often convergent but it is also often slow which can be prohibitive when the function itself is expensive.

This paper shows how the finding of a fixed point of a function can be accelerated using fixed point accelerators in **R**. The next section starts by with a brief explanation of fixed points before a number of fixed point acceleration algorithms are discussed. The algorithms examined include the Newton, Aitken and Scalar Epsilon Algorithm (SEA) methods that are designed for accelerating the convergence of scalar sequences. Five algorithms for accelerating vector sequences are also discussed including the Vector Epsilon Algorithm (VEA), Anderson acceleration and three minimal polynomial algorithms (MPE, RRE and the squarem method provided in the **SQUAREM** package). The **FixedPoint** package is then introduced with applications of how it can be used to find fixed points. In total five problems are described which show how fixed point accelerators can be used in solving problems in asset pricing, machine learning and economics. Here the intent is not only to showcase the capabilities of **FixedPoint** and **SQUAREM** but also to demonstrate how various problems may be able to be recast in an iterate way in order to be able to exploit fixed point accelerators. Finally this paper uses the presented numerical problems to perform a speed of convergence test on all of the algorithms presented in this paper.

Fixed point acceleration

Fixed point problems

A fixed point problem is one where we look for a vector, $X \in \Re^N$, so that for a given real valued function $f : \Re^N \rightarrow \Re^N$ we have:

$$f(X) = X \tag{Z.2.1}$$

If $f : \Re^1 \rightarrow \Re^1$ and thus any solution X will be a scalar then one way to solve this problem would be to use a rootfinder on the function $g(x) = f(x) - x$ or to use an optimiser to minimise a function like $h(x) = (f(x) - x)^2$. These techniques will not generally work however if $f : \Re^N \rightarrow \Re^N$ where N is large. Consider for instance using a multidimensional Newtonian optimiser to minimise $h(x) = \sum_{i=1}^{N}(f_i(x) - x_i)^2$ where $f_i(x)$ is the i'th element output by $f(x)$. The estimation of gradients for each individual dimension may take an unfeasibly long time. In addition this method may not

[1] The squarem method has also been implemented in the **turboEM** package (Bobb and Varadhan, 2014).

[2] The Anderson method has since been implemented in the **daarem** package (Henderson and Varadhan, 2018).

make use all of the available information. Consider for instance that we know that the solution for x will be an increasing vector (so $x_i > x_j$ for any entries of x with $i > j$) with many elements. This information can be preserved and used in the fixed point acceleration algorithms that we present but would be more difficult to exploit in a standard optimisation algorithm.

Much of the intuition behind the use of optimisers and rootfinders carries over to the use of fixed point acceleration algorithms. Like a function may have multiple roots and multiple local optima, a function may have multiple fixed points. The extreme case of this is the identity mapping $f(x) = x$ for which every x is a fixed point. Some functions have no roots or optima and likewise some functions do not possess fixed points. This is the case for the function $f(x) = \frac{-1}{x}$. From a practical standpoint, it is often useful to have access to multiple optimisers and rootfinders as different algorithms are better suited to different types of functions. This is also the case for finding fixed points and the **FixedPoint** package is useful in this regard, offering eight fixed point algorithms.

The first algorithm implemented in **FixedPoint** is the "simple" method which merely takes the output of a function and feeds it back into the function. For instance starting with a guess of x_0, the next guess will be $x_1 = f(x_0)$. The guess after that will be $x_2 = f(x_1)$ and so on. Under some conditions f will be a contraction mapping and so the simple method will be guaranteed to converge to a unique fixed point (Stokey et al., 1989). Even when this is the case however the simple method may only converge slowly which can be inconvenient. The other seven methods implemented in **FixedPoint** and the squarem method of **SQUAREM** are designed to be faster than the simple method but may not be convergent for every problem.

Fixed point acceleration algorithms

Newton acceleration

Here we will define $g(x) = f(x) - x$. The general approach is to solve $g(x)$ with a rootfinder. The x that provides this root will be a fixed point. Thus after two iterates we can approximate the fixed point with:

$$\text{Next guess} = x_i - \frac{g(x_i)}{g'(x_i)} \tag{Z.2.2}$$

FixedPoint approximates the derivative $g'(x_i)$ such that we use to get an estimated fixed point of:

$$\text{Next guess} = x_i - \frac{g(x_i)}{\frac{g(x_i)-g(x_{i-1})}{x_i-x_{i-1}}} \tag{Z.2.3}$$

The implementation of the Newton method in **FixedPoint** uses this formula to predict the fixed point given two previous function iterates. This method is designed for use with scalar functions. If it is used with higher dimensional functions that take and return vectors then it will be used elementwise.

Aitken acceleration

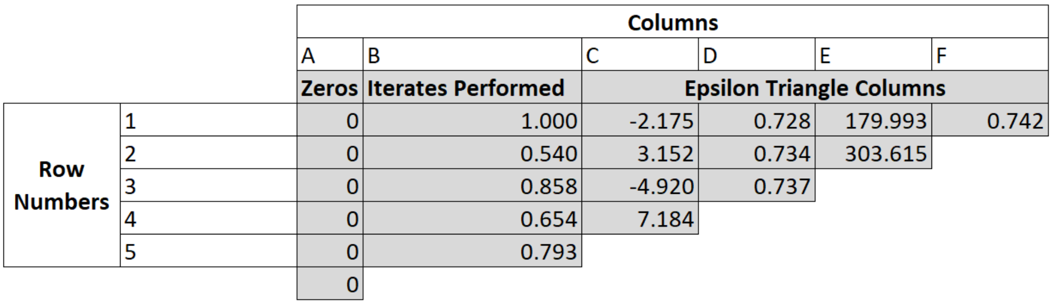
Consider that a sequence of scalars $\{x_i\}_{i=0}^{\infty}$ that converges linearly to its fixed point of $\hat{x}$. This implies that for a some i:

$$\frac{\hat{x} - x_{i+1}}{\hat{x} - x_i} \approx \frac{\hat{x} - x_{i+2}}{\hat{x} - x_{i+1}} \tag{Z.2.4}$$

For a concrete example consider that every iteration halves the distance between the current value of x_i and the fixed point. In this case the left hand side will be one half which will equal the right hand side which will also be one half. Equation Z.2.4 can be simply rearranged to give a formula predicting the fixed point that is used as the subsequent iterate. This is:

$$\text{Next guess} = x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2x_{i+1} + x_i} \tag{Z.2.5}$$

The implementation of the Aitken method in **FixedPoint** uses this formula to predict the fixed point given two previous iterates. This method is designed for use with scalar functions. If it is used with higher dimensional functions that take and return vectors then it will be used elementwise.



		Columns					
		A	B	C	D	E	F
		Zeros	Iterates Performed	Epsilon Triangle Columns			
Row Numbers	1	0	1.000	-2.175	0.728	179.993	0.742
	2	0	0.540	3.152	0.734	303.615	
	3	0	0.858	-4.920	0.737		
	4	0	0.654	7.184			
	5	0	0.793				
		0					

Figure 1: The Epsilon Algorithm applied to the cos(x) function

Epsilon algorithms

The epsilon algorithms introduced by Wynn (1962) provides an alternate method to extrapolate to a fixed point. This paper will present a brief numerical example and refer readers to Wynn (1962) or Smith et al. (1987) for a mathematical explanation of why it works. The basic epsilon algorithm starts with a column of simple function iterates. If i iterates have been performed then this column will have a length of $i + 1$ (the initial starting guess and the results of the i iterations). Then a series of columns are generated by means of the below equation:

$$\epsilon_{c+1}^{r} = \epsilon_{c-1}^{r+1} + (\epsilon_{c}^{r+1} - \epsilon_{c}^{r})^{-1} \tag{Z.2.6}$$

Where c is a column index and r is a row index. The algorithm starts with the ϵ_0 column being all zeros and ϵ_1 being the column of the sequence iterates. The value in the furthest right column ends up being the extrapolated value.

This can be seen in the figure 1 which uses an epsilon method to find the fixed point of $\cos(x)$ with an initial guess of a fixed point of 1. In this figure B1 is the initial guess of the fixed point. Then we have the iterates $B2 = \cos(B1)$, $B3 = \cos(B2)$ and so on. Moving to the next column we have $C1 = A2 + 1/(B2 - B1)$ and $C2 = A3 + 1/(B3 - B2)$ and so on before finally we get $F1 = D2 + 1/(E2 - E1)$. As this is the last entry in the triangle it is also the extrapolated value.

Note that the values in columns C and E are poor extrapolations. Only the even columns D,F provide reasonable extrapolation values. For this reason an even number of iterates (an odd number of values including the starting guess) should be used for extrapolation. **FixedPoint** will enforce this by throwing away the first iterate provided if necessary to get an even number of iterates.

In the vector case this algorithm can be visualised by considering each entry in the above table to contain a vector going into the page. In this case the complication emerges from the inverse term in equation Z.2.6: there is no clear interpretation of $(\epsilon_c^{r+1} - \epsilon_c^{r})^{-1}$ when $(\epsilon_c^{r+1} - \epsilon_c^{r})$ represents a vector. The Scalar Epsilon Algorithm (SEA) uses elementwise inverses to solve this problem which ignores the vectorised nature of the function. The Vector Epsilon Algorithm (VEA) uses the Samuelson inverse of each vector $(\epsilon_c^{r+1} - \epsilon_c^{r})$ as described in Smith et al. (1987).

Minimal polynomial algorithms

FixedPoint implements two minimal polynomial algorithms, Minimal Polynomial Extrapolation (MPE) and Reduced Rank Extrapolation (RRE). The key intuition for these methods is that a linear combination of previous iterates is taken to generate a new guess vector. The coefficients of the previous iterates are taken so that this new guess vector is expected to not be changed much by the function.[3]

To first define notation, each vector (the initial guess and subsequent iterates) is defined by $x_0, x_1, \ldots$. The first differences are denoted $u_j = x_{j+1} - x_j$ and the second differences are denoted $v_j = u_{j+1} - u_j$. If we have already completed $k - 1$ iterations (and so we have k terms) then we will use matrices of first and second differences with $U = [u_0, u_1, \ldots, u_{k-1}]$ and $V = [v_0, v_1, \ldots, v_{k-1}]$.

[3] For more details an interested reader is directed to Cabay and Jackson (1976) or Smith et al. (1987) for a detailed explanation.

For the MPE method the extrapolated vector, is found by:

$$\text{Next guess} = \frac{\sum_{j=0}^{k} c_j x_j}{\sum_{j=0}^{k} c_j} \tag{Z.2.7}$$

Where the coefficient vector is found by $c = -U^{+} u_k$ where U^{+} is the Moore-Penrose generalised inverse of the U matrix. In the case of the RRE method the extrapolated vector, is found by:

$$\text{Next guess} = x_0 - UV^{+} u_0 \tag{Z.2.8}$$

The only effective fixed point accelerator that was available in R prior to the release of the **FixedPoint** package was the squarem method provided in the **SQUAREM** packages. This method modifies the minimal polynomial algorithms with higher order terms and can thus be considered as a variant of the MPE algorithms. The squarem method is primarily intended to accelerate convergence in the solution of expectation maximisation problems but can be used more generally with any function that is a contraction mapping (Varadhan and Roland, 2008).

Anderson acceleration

Anderson (1965) acceleration is an acceleration algorithm that is well suited to functions of vectors. Similarly to the minimal polynomial algorithms it takes a weighted average of previous iterates. It is different however to all previous algorithms in that the previous iterates used to generate a guess vector need not be sequential but any previous iterates can be used. Thus it is well suited to parallelising the finding of a fixed point.[4]

Consider that we have previously run an N-dimensional function M times. We can define a matrix $G_i = [g_{i-M}, g_{i-M+1}, ..., g_i]$ where $g(x_j) = f(x_j) - x_j$. Each column of this matrix can be interpreted as giving the amount of "movement" that occurred in a run of the function.

In Anderson acceleration we assign a weight to apply to each column of the matrix. This weight vector is M-dimensional and can be denoted $\alpha = \{\alpha_0, \alpha_1, ..., \alpha_M\}$. These weights are determined by means of the following optimisation:

$$\begin{aligned} &\min_{\alpha} ||G_i \alpha||_2 \\ &s.t. \sum_{j=0}^{M} \alpha_j = 1 \end{aligned} \tag{Z.2.9}$$

Thus we choose the weights that will be predicted to create the lowest "movement" in an iteration.

With these weights we can then create the expression for the next iterate as:

$$\text{Next guess} = \sum_{j=0}^{M} \alpha_j f(x_{i-M+j}) \tag{Z.2.10}$$

Robustness of fixed point algorithms

Functions with restricted input spaces

Some functions have a restricted input space. Acceleration schemes can perform badly in these settings by proposing vectors that sit outside of the required input space. As an example consider the following $\Re^2 \rightarrow \Re^2$ function, that we try to find the fixed point for with the Anderson method:

$$\text{Output} = \left(\frac{\sqrt{\text{Input}[1] + \text{Input}[2]}}{2} \quad , \quad \left| \frac{3\text{Input}[1]}{2} + \frac{\text{Input}[2]}{2} \right| \right) \tag{Z.3.1}$$

```
library(FixedPoint)
SimpleVectorFunction = function(x){c(0.5*sqrt(x[1] + x[2]), abs(1.5*x[1] + 0.5*x[2]))}
FPSolution = FixedPoint(Function = SimpleVectorFunction, Inputs = c(0.3,900),
                        Method = "Anderson")
```

[4] An example of this is shown in the appendix for the consumption smoothing problem described later in this paper.

Unfortunately an error will occur here. After four iterates the Anderson method decides to try the vector $(-1.085113, -3.255338)$. This results in the square root of a negative number and hence the output is undefined.

In cases like this there are a few things a user can try. The first is to change the function to another function that retains the same fixed points. In the above case we could change the function to take the absolute value of the sum of the two inputs before taking the square root. Then after finding a fixedpoint we can verify if the sum of the two entries is positive and hence it is also a solution to the original function. Another measure that could be tried is to change the initial guess. Finally we could change the acceleration method. The simple method will be robust in this case as the function will never return an Output vector that sums to a negative number. It is still likely to be slow however.[5] A special feature of the FixedPoint package is that it allows methods to be changed while retaining previous iterates. So in this case we can run the preceding code until an error causes the acceleration to stop, switch to the simple method for a few iterates and then switch back to the anderson method. No error will result as we are close enough to the fixedpoint that each new guess sums to be positive:

```
FPSolution = FixedPoint(Function = SimpleVectorFunction, Inputs = FPSolution$Inputs,
                        Outputs = FPSolution$Outputs, Method = "Simple", MaxIter = 5)
# Now we switch to the Anderson Method again. No error results because we are
# close to fixed point.
FPSolution = FixedPoint(Function = SimpleVectorFunction, Inputs = FPSolution$Inputs,
                        Outputs = FPSolution$Outputs, Method = "Anderson")
```

Another example of a restricted input space is shown in the consumption smoothing example presented later in this paper. In this example the input vector must reproduce a monotonic and concave function. All of the vectorised methods presented in this paper take a combination of previous iterates all of which take and return vectors representing monotonic and concave functions. As a result these methods will only propose vectors representing monotonic and concave functions. By contrast the Newton, SEA and Aitken methods do not take into account the entire vector when proposing the fixedpoint value for each element of the vector and as a result some of the input vectors proposed by these methods may not be valid. Ensuring that a vectorised method is chosen is thus sufficient in this case to ensure that each vector tried is within the input space of the function for which a fixedpoint is sought.

Convergence by constant increments

Most fixed point acceleration algorithms will fail in finding the fixed point of a function that converges by a fixed increment. For instance we may have a function that takes x and returns x shifted 1 unit (in Euclidean norm) in a straight line towards its fixed point. A realistic example of this type of convergence is the training of a perceptron classifier which is explored later in this paper.

This type of convergence is problematic for all algorithms presented except for the simple method. The basic problem can be illustrated simply by looking at the Newton and Aitken methods. For the Newton method consider the derivative in equation Z.2.3 which is approximated by $\frac{g(x_i)-g(x_{i-1})}{x_i-x_{i-1}}$. When there is convergence by constant increments then $g(x_i) = g(x_{i-1})$ and the derivative is zero which means calculating the Newton method's recommended new guess of the fixed point involves division by zero. Now considering the Aitken method of equation Z.2.5 the new guess is given by $x_i - \frac{(x_{i+1}-x_i)^2}{x_{i+2}-2x_{i+1}+x_i}$. When there is convergence by constant increments then $x_i - x_{i+1} = x_{i+1} - x_{i+2}$ and so we have $x_{i+2} - 2x_{i+1} + x_i = (x_i - x_{i+1}) - (x_{i+1} - x_{i+2}) = 0$. It is not possible to calculate the new guess.

More generally, when there is convergence by constant increments then then the fixed point method receives information about what direction to go in but no information about how far to go. This is a complication that is common to all fixed point acceleration methods. In these cases it may be possible to change the function to make it converge by varying increments while retaining the same set of fixed points. An example of this is shown in the perceptron example presented later in this paper. In other cases where it is not possible to modify the function, it is advisable to use the simple method.

[5] As the simple method is so often monotonic and convergent the **FixedPoint** package has a "dampening" parameter which allows users to create guesses by linearly combining the guesses of their desired acceleration method with the simple iterates. This allows users to combine the robustness advantages of the simple method with the speed advantages of another method.

Applications of fixed point acceleration with the FixedPoint package

Simple examples with analytical functions

For the simplest possible example we will use the **FixedPoint** package to accelerate the solution for a square root. Consider we want to estimate a square root using the Babylonian method. To find the square root of a number x, given an initial guess t_0, the following sequence converges to the square root:

$$t_{n+1} = \frac{1}{2}\left[t_n + \frac{x}{t_n}\right] \tag{Z.4.1}$$

This is a fast converging and inexpensive sequence which probably makes an acceleration algorithm overkill but for sake of exposition we can implement this in **FixedPoint**. In the next code block we find the square root of 100 with the SEA method and an initial guess of six:

```
library(FixedPoint)
SequenceFunction = function(tn){0.5*(tn + 100/tn)}
FP = FixedPoint(Function = SequenceFunction, Inputs = 6, Method = "SEA")
```

The benefit of fixed point accelerators is more apparent when applied to vectorised functions. For a simple example consider the below function where each element of the returned vector depends on both elements of the input vector:

```
Vec_Function = function(x){c(0.5*sqrt(abs(x[1] + x[2])), 1.5*x[1] + 0.5*x[2])}
FP_Simple   = FixedPoint(Function = Vec_Function, Inputs = c(0.3,900),
                         Method = "Simple")
FP_Anderson = FixedPoint(Function = Vec_Function, Inputs = c(0.3,900),
                         Method = "Anderson")
```

Here it takes 105 iterates to find a fixed point with the simple method but only 14 with the Anderson acceleration method.

Gas diffusion

For a more complex example consider we want to model the diffusion of gas in a two dimensional space. We set up a two dimensional grid split into ϕ divisions along the side so there are ϕ^2 grid squares in total. Pure nitrogen is being released at location $(1, 1)$ and pure oxygen is being released at location (ϕ, ϕ). We are interested in determining the steady state gas concentrations in each square of the grid. We will model equilibrium as occurring when each square has a gas concentration equal to the average of itself with its contiguous squares.

```
phi = 10
Numbering = matrix(seq(1,phi^2,1), phi)           # Numbering scheme for squares

NeighbourSquares = function(n,phi){
  SurroundingIndexes = c(n)
  if (n %% phi != 1){SurroundingIndexes = c(SurroundingIndexes, n-1)}    # above
  if (n %% phi != 0){SurroundingIndexes = c(SurroundingIndexes, n+1)}    # below
  if (n > phi){SurroundingIndexes = c(SurroundingIndexes, n-phi)}        # right
  if (n <= phi^2-phi){SurroundingIndexes = c(SurroundingIndexes, n+phi)} # left
  return(SurroundingIndexes)
}

TwoDimensionalDiffusionIteration = function(x, phi){
  xnew = x
  for (i in 1:(phi^2)){
    Subset = NeighbourSquares(i, phi)
    xnew[i] = mean(x[Subset])
  }
  xnew[1] = 0
  xnew[phi^2] = 1
  return(xnew)
}
```

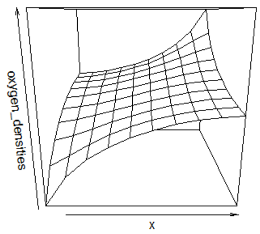


Figure 2: Equilibrium concentrations of Oxygen found by the `fixedpoint()` function

```
FP = FixedPoint(Function = function(x) TwoDimensionalDiffusionIteration(x,phi),
                Inputs =  c(rep(0,50), rep(1,50)), Method = "RRE")
```

The fixed point found here can then be used to plot the density of oxygen over the space. The code for this is below while the plot can be found in figure 2.

```
x = 1:phi
y = 1:phi
oxygen_densities = matrix(FP$FixedPoint, phi)
persp(x, y, oxygen_densities)
```

Finding equilibrium prices in a pure exchange economy

Consider now we are modeling a pure exchange economy and want to determine the equilibrium prices given household preferences and endowments. We have N households. Every household has preferences over G types of good. Household $n \in N$ has a utility function of

$$U_n = \sum_{i=1}^{G} \gamma_{n,i} \log(c_{n,i}) \tag{Z.4.2}$$

Where $\gamma_{n,i}$ is a parameter describing household n's taste for good i, $c_{n,i}$ is household n's consumption of good i. Each household is endowed with an amount of each good. They can then trade goods before consumption. We have data on each household's endowment and preferences for each good and want to determine the equilibrium prices for this pure exchange economy.

We will choose good 1 as the numeraire, so we will have $P_1 = 1$. First we will find an expression for demand given a price vector. Setting up the lagrangian for household n:

$$L_n = \sum_{i=1}^{G} \gamma_{n,i} \log(c_{n,i}) + \lambda_n [\sum_{i=1}^{G} P_i(e_{n,i} - c_{n,i})] \tag{Z.4.3}$$

Where λ_n is household n's shadow price, $e_{n,i}$ is this household's endowment of good i and P_i is the price of good i. Taking the first order condition with respect to c_i of this lagrangian yields:

$$c_{n,i} = \frac{\gamma_{n,i}}{P_i \lambda_n} \tag{Z.4.4}$$

and taking the first order condition with respect to λ_n yields the budget constraint. Subbing the above equation into the budget constraint and rearranging yields:

$$\lambda_n = \frac{\sum_{i=1}^{G} \gamma_{n,i}}{\sum_{i=1}^{G} P_i e_{n,i}} \tag{Z.4.5}$$

We can also sum over households to find total demand for each good as:

$$D_i = \frac{1}{P_i} \sum_{n=1}^{G} \frac{\gamma_{n,i}}{\lambda_n} \tag{Z.4.6}$$

We will find the equilibrium price vector by using an approximate price vector to estimate the λs using equation Z.4.5. We can then find an estimate of the equilibrium price P_i which solves clears the market, $D_i = \sum_{n=1}^{G} e_{n,i}$:

$$P_i = \frac{\sum_{n=1}^{G} \frac{\gamma_{n,i}}{\lambda_n}}{\sum_{n=1}^{G} e_{n,i}} \tag{Z.4.7}$$

We use this approach in the code below for the case of 10 goods with 8 households. For exposition sake we generate some data below before proceeding to find the equilibrium price vector.

```
# Generating data
set.seed(3112)
N = 8
G = 10
Endowments = matrix(rlnorm(N*G), nrow = G)
Gamma      = matrix(runif(N*G), nrow = G)
# Every column here represents a household and every row is a good.
# So Endowments[1,2] is the second household's endowment of good 1.

# We now start solving for equilibrium prices:
TotalEndowmentsPerGood  = apply(Endowments, 1, sum)
TotalGammasPerHousehold = apply(Gamma, 2, sum)
LambdasGivenPriceVector = function(Price){
  ValueOfEndowmentsPerHousehold = Price * Endowments
  TotalValueOfEndowmentsPerHousehold = apply(ValueOfEndowmentsPerHousehold, 2, sum)
  return(TotalGammasPerHousehold /TotalValueOfEndowmentsPerHousehold)
}

IterateOnce = function(Price){
  Lambdas = LambdasGivenPriceVector(Price)                      # eqn 16
  GammaOverLambdas = t(apply(Gamma, 1, function(x) x / Lambdas))
  SumGammaOverLambdas = apply(GammaOverLambdas,1,sum)
  NewPrices = SumGammaOverLambdas/ TotalEndowmentsPerGood       # eqn 18
  NewPrices = NewPrices/NewPrices[1]        # normalising with numeraire
  return(NewPrices)
}

InitialGuess = rep(1,10)
FP = FixedPoint(Function = IterateOnce, Inputs =  InitialGuess, Method = "VEA")
```

The fixed point contained in the FP object is the vector of equilibrium prices.

The training of a perceptron classifier

The perceptron is one of the oldest and simplest machine learning algorithms (Rosenblatt, 1958). In its simplest form, for each observation it is applied it uses an N-dimensional vector of features x together with N+1 weights $\mathbf{w}$ to classify the observation as being of type one or type zero. It classifies observation j as a type one if $w_0 + \sum_{i=1}^{N} w_i x_{i,j} > 0$ and as a type zero otherwise.

The innovation of the perceptron was its method for training its weights, $\mathbf{w}$. This is done by looping over a set of observations that can be used for training (the "training set") and for which the true category information is available. The perceptron classifies each observation. When it classifies an observation correctly no action is taken. On the other hand when the perceptron makes an error then it updates its weights with the following expressions.

$$w_0' = w_0 + (d_j - y_j) \tag{Z.4.8}$$

$$w_i' = w_i + (d_j - y_j)x_{j,i} \qquad \text{for } i > 0 \tag{Z.4.9}$$

Where w_i is the old weight for the i'th feature and w_i' is the updated weight. $x_{j,i}$ is the feature value for observation j's feature i, d_j is the category label for observation j and y_j is the perceptron's prediction for this observation's category.

This training algorithm can be rewritten as fixed point problem. We can write a function that takes perceptron weights, loops over the data updating these weights and then returns the updated weight vector. If the perceptron classifies every observation correctly then the weights will not update and we are at a fixed point.[6]

Most acceleration algorithms perform poorly in accelerating the convergence of this perceptron training algorithm. This is due to the perceptron often converging by a fixed increment. This occurs because multiple iterates can result in the same observations being misclassified and hence the same change in the weights. As a result we will use the simple method which is guaranteed to be convergent for this problem (Novikoff, 1963).

```
# Generating linearly separable data
set.seed(10)
data = data.frame(x1 = rnorm(100,4,2), x2 = rnorm(100,8,2), y = -1)
data = rbind(data,data.frame(x1 = rnorm(100,-4,2), x2 = rnorm(100,12), y = 1))

# Iterating training of Perceptron
IteratePerceptronWeights = function(w, LearningRate = 1){
  intSeq = 1:length(data[,"y"])
  for (i in intSeq){
    target = data[i,c("y")]
    score  =  w[1] + (w[2]*data[i, "x1"]) + (w[3]*data[i, "x2"])
    ypred  = 2*(as.numeric( score > 0 )-0.5)
    update = LearningRate * 0.5*(target-ypred)
    w[1]   = w[1] + update
    w[2]   = w[2] + update*data[i, "x1"]
    w[3]   = w[3] + update*data[i, "x2"]
  }
  return(w)
}

InitialGuess = c(1,1,1)
FP = FixedPoint(Function = IteratePerceptronWeights, Inputs =  InitialGuess,
                Method = "Simple", MaxIter = 1200)
```

The result of this algorithm can be seen in figure 3. It can be seen that the classification line perfectly separates the two groups of observations.

Only the simple method is convergent here and it is relatively slow taking 1121 iterations. We can still get a benefit from accelerators however if we can modify the training algorithm to give training increments that change depending on distance from the fixed point. This can be done by updating the weights by an amount proportional to a concave function of the norm of $w_0 + \sum_{i=1}^{N} w_i x_{i,j}$. Note that the instances in which the weights are not updated stay the same and hence the modified training function will result in the same set of fixed points as the basic function. This is done in the next piece of code where the MPE method is used. It can be seen that there is a substantial increase in speed with only 54 iterations required by the MPE method.

```
IteratePerceptronWeights = function(w, LearningRate = 1){
  intSeq = 1:length(data[,"y"])
  for (i in intSeq){
    target = data[i,c("y")]
    score  =  w[1] + (w[2]*data[i, "x1"]) + (w[3]*data[i, "x2"])
    ypred  = 2*(as.numeric( score > 0 )-0.5)
    if ((target-ypred) != 0){
      update = LearningRate * -sign(score) * sqrt(abs(score))
      w[1] = w[1] + update
      w[2] = w[2] + update*data[i, "x1"]
      w[3] = w[3] + update*data[i, "x2"]
    }
  }
  return(w)
}
FP = FixedPoint(Function = IteratePerceptronWeights, Inputs =  InitialGuess,
                Method = "MPE")
```

[6] Note that when a perceptron has one fixed point then there are uncountably many such fixed points where the perceptron correctly classifies the entire training set and will not further update. This is because a scalar multiple of any set of weights will generate the same classification line and the new set of weights will also be a fixed point. There may also be multiple linearly independent hyperplanes that correctly classify every observation. On the other hand it is possible that the data is not linearly separable in which case there may be no fixed point and the weights will continue to update forever.

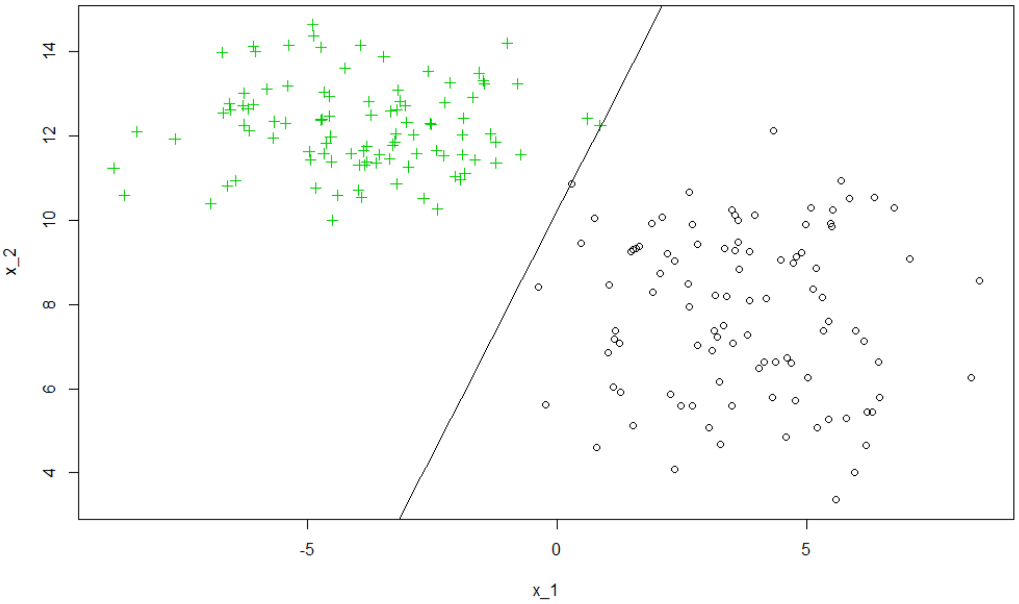


Figure 3: The perceptron linear classifier

Valuation of a perpetual American put option

For an application in finance consider the pricing of a perpetual American put option on a stock. It never expires unless it is exercised. Its value goes to zero however if the spot price rises to become α times as much as the strike price, denoted S.[7] We will denote x to be the current spot price, σ is the market volatility, d is the risk free rate. In each period the underlying price either increases by a multiple of e^{σ} (which happens with probability p) or decreases by a multiple of $e^{-\sigma}$ (which happens with probability $1-p$) in each unit of time. We have $-\sigma < d < \sigma$.

Given the risk neutral pricing principle the returns from holding the stock must equal the risk-free rate. Hence we must have $pe^{\sigma} + (1-p)e^{-\sigma} = e^{d}$. This implies that:

$$p = \frac{e^{d} - e^{-\sigma}}{e^{\sigma} - e^{-\sigma}} \tag{Z.4.10}$$

The price of this option at any given spot price of the stock can be solved by means of a fixed point algorithm as shown below:[8]

```
d = 0.05
sigma = 0.1
alpha = 2
S = 10
chi = 0
p = (exp(d) - exp(-sigma) ) / (exp(sigma) - exp(-sigma))

# Initially we guess that the option value decreases linearly from S
# (when the spot price is 0) to 0 (when the spot price is \alpha S).
UnderlyingPrices = seq(0,alpha*S, length.out = 100)
OptionPrice = seq(S,chi, length.out = 100)

ValueOfExercise = function(spot){S-spot}
ValueOfHolding = function(spot, EstimatedValueOfOption){
```

[7] This is a common approximation when pricing American options with a finite difference method. While no option's price will ever become exactly zero, at a sufficiently high spot price the option will be low enough value for this to be a good approximation.

[8] In this case the **SQUAREM** package is used with the squarem method. To use the MPE method through the **SQUAREM** package we could add list(K = 2, method="mpe", square=FALSE) as the control argument to the squarem function call. RRE can be implemented analogously.

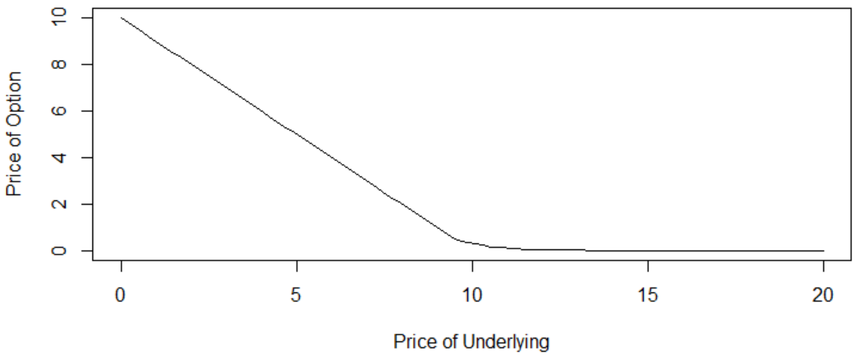


Figure 4: Price of Perpetual American put for each level of the spot price

```
  if (spot > alpha*S-1e-10){return(chi)}
  IncreasePrice = exp(sigma)*spot
  DecreasePrice = exp(-sigma)*spot
return((p*EstimatedValueOfOption(IncreasePrice) +
          (1-p)*EstimatedValueOfOption(DecreasePrice)))
}
ValueOfOption = function(spot, EstimatedValueOfOption){
  Holding = ValueOfHolding(spot, EstimatedValueOfOption)*exp(-d)
  Exercise = ValueOfExercise(spot)
  return(max(Holding, Exercise))
}
IterateOnce = function(OptionPrice){
  EstimatedValueOfOption = approxfun(UnderlyingPrices, OptionPrice, rule = 2)
  for (i in 1:length(OptionPrice)){
    OptionPrice[i] = ValueOfOption(UnderlyingPrices[i], EstimatedValueOfOption)
  }
  return(OptionPrice)
}

library(SQUAREM)
FP = squarem(par=OptionPrice, IterateOnce)

plot(UnderlyingPrices,FP$par, type = "l",
     xlab = "Price of Underlying", ylab = "Price of Option")
```

Here the fixed point gives the price of the option at any given level of the underlying asset's spot price. This can be visualized as seen in figure 4.

```
plot(UnderlyingPrices,FP$FixedPoint, type = "l",
     xlab = "Price of Underlying", ylab = "Price of Option")
```

A consumption smoothing problem

A common feature of macroeconomic models is the simulation of consumer spending patterns over time. These computations are not trivial, in order for a consumer to make a rational spending decision they need to know their future wellbeing as a function of their future wealth. Often models exhibit infinitely lived consumers without persistent shocks and in this setting the relationship between wealth and wellbeing can be found with a fixed point algorithm. Consider an infinitely lived consumer that has a budget of B_t at time t and a periodic income of 1. She has a periodic utility function given by $\epsilon_t x_t^\delta$, where x_t is spending in period t and ϵ_t is the shock in period t drawn from some stationary nonnegative shock process with pdf $f(\epsilon)$ defined on the interval $[y, z]$. The problem for the consumer in period t is to maximise her value function:

$$V(B_t|\epsilon_t) = \max_{0<x_t<B_t} \quad \epsilon_t x_t^\delta + \beta \int_y^z V(B_{t+1}|\epsilon) f(\epsilon) d\epsilon \tag{Z.4.11}$$

Where β is a discounting factor and $B_{t+1} = 1 + B_t - x_t$.

Our goal is to find a function that gives the optimal spending amount, $\hat{x}(B_t, \epsilon_t)$, in period t which is a function of the shock magnitude ϵ_t and the available budget B_t in this period. If we knew the function $\int_y^z V(B_{t+1}|\epsilon)f(\epsilon)d\epsilon$ then we could do this by remembering $B_{t+1} = 1 + B_t - x_t$ and using the optimisation:

$$\hat{x}(B_t, \epsilon_t) = \text{argmax}_{0<x_t<B_t} \quad \epsilon_t x_t^\delta + \beta \int_y^z V(B_{t+1}|\epsilon)f(\epsilon)d\epsilon \tag{Z.4.12}$$

So now we need to find the function $\int_y^z V(B_{t+1}|\epsilon)f(\epsilon)d\epsilon$. Note as the shock process is stationary, the consumer lives forever and income is always 1, this function will not vary with t. As a result we will rewrite it as simply $f(b)$, where b is the next period's budget.

Now we will construct a vector containing a grid of budget values, $\bar{b}$, for instance $\bar{b} = [0, 0.01, 0.02, ..., 5]$ (we will use bars to describe approximations gained from this grid). If we could then approximate a vector of the corresponding function values, $\bar{f}$, so we had for instance $\bar{f} = [f(0), f(0.01), f(0.02), ..., f(5)]$ then we could approximate the function by constructing a spline $\bar{f}(b)$ between these points. Then we can get the function:

$$\bar{x}(B_t, \epsilon_t) = \text{argmax}_{0<x<B_t} \quad \epsilon_t x_t^\delta + \bar{f}(B_t - x) \tag{Z.4.13}$$

So this problem reduces to finding the vector of function values at a discrete number of points, $\bar{f}$. This can be done as a fixed point problem. We can first note that this problem is a contraction mapping problem. In this particular example this means that if we define a sequence $\bar{f}_0 = f_0$ where f_0 is some initial guess and $f_{i+1} = g(f_i)$ where g is given by the `IterateOnce()` function below then this sequence will be convergent.[9] Convergence would be slow however so below we will actually use the Anderson method:

```
library(FixedPoint)
library(schumaker)
library(cubature)
delta = 0.2
beta = 0.99
BudgetStateSpace = c(seq(0,1, 0.015), seq(1.05,3,0.05))
InitialGuess = sqrt(BudgetStateSpace)

ValueGivenShock = function(Budget, epsilon, NextValueFunction){
  optimize(f = function(x) epsilon*(x^delta) + beta*NextValueFunction(Budget - x + 1),
           lower = 0, upper = Budget, maximum = TRUE)
}

ExpectedUtility = function(Budget, NextValueFunction){
  if (Budget > 0.001){
    adaptIntegrate(f = function(epsilon) ValueGivenShock(Budget,
        epsilon,NextValueFunction)$objective * dlnorm(epsilon),
        lowerLimit = qlnorm(0.0001), upperLimit = qlnorm(0.9999))$integral
  } else {
    beta*NextValueFunction(1)
  }
}

IterateOnce = function(BudgetValues){
  NextValueFunction = schumaker::Schumaker(BudgetStateSpace, BudgetValues,
                                             Extrapolation = "Linear")$Spline
  for (i in 1:length(BudgetStateSpace)){   # This is often a good loop to parallelise
    BudgetValues[i] = ExpectedUtility(BudgetStateSpace[i], NextValueFunction)
  }
  return(BudgetValues)
}
```

[9] We use two additional packages in solving this problem. The first is the **cubature** package (Narasimhan and Johnson, 2017) which is used for the integral in equation Z.4.12. The second is the **schumaker** package (Baumann and Klymak, 2017) which generates a spline representing $\bar{f}(B_t - x)$ in equation Z.4.13. It is necessary for this spline to be shape preserving to ensure there is a unique local maxima to be found for the optimiser used in evaluating this expression.

```
FP = FixedPoint(Function = IterateOnce, Inputs = InitialGuess,
                         Method = "Anderson")
```

This takes 71 iterates which is drastically better than the 2316 iterates it takes with the simple method. Now the optimal spending amount can be found for any given budget and any income shock. For instance with the following code we can work out what a consumer with a budget of 1.5 and a shock of 1.2 would spend:

```
NextValueFunction = Schumaker(BudgetStateSpace, FP$FixedPoint)$Spline
ValueGivenShock(1.5, 1.2, NextValueFunction)$maximum
```

Using parallelisation with the Anderson method

It takes 71 iterates for the Anderson method to find the fixed point, however we might want to get it going even faster though parallelisation. The easiest way to do this for this particular problem is to parallelise the for loop through the budgetspace. For exposition however we show how to do this by doing multiple iterates at the same time. We will do this by using six cores and using the parallel capabilities of the **foreach** and **doParallel** packages (Revolution Analytics and Weston, 2015; Microsoft Corporation and Weston, 2017). Each node will produce an different guess vector through the Anderson method. This will be done by giving each node a different subset of the previous iterates that have been completed. The first node will have all previous iterate information. For $i > 1$, the ith node will have all previous iterates except for the ith most recent iterate. The code for this approach is presented in the appendix.

This parallel method takes 102 iterates when using six cores which takes approximately the same time as running $6 + \frac{96}{6} = 22$ iterates sequentially. This is a significant speedup and is possible with the Anderson method as previous iterates do not need to be sequential. The simple parallel algorithm here may also be able to be modified for better performance, for instance different methods could be used in each core or the dampening parameter could be modified.

Speed of convergence comparison

All of the algorithms of the **FixedPoint** package as well as the squarem algorithm of the **SQUAREM** package were run for a variety of problems. In addition to all of the above problems fixed points were found for some basic analytical functions such as $cos(x)$, $x^{\frac{1}{3}}$ and the linear case of $95(18 - x)$.[10] The results are shown in table 1.

It can be seen that the Anderson algorithm performed well in almost all cases. The minimal polynomial methods tended to outperform the epsilon extrapolation methods. This is largely in agreement with previous benchmarking performed in Jbilou and Sadok (2000). The MPE tended to generally outperform the RRE and the VEA outperformed the SEA in all cases. The squarem method tended to be outperformed by the standard minimal polynomial methods. While it was generally amongst the slowest methods, the simple method was the most generally applicable, converging in all but one of the test cases studied.

Conclusion

R has had available a multitude of algorithms for rootfinding and multidimensional optimisation for a long time. Until recently however the range of fixed point accelerators available in **R** has been limited. Before the release of **FixedPoint**, only the squarem method of the **SQUAREM** package was available as a general use fixed point accelerator.

This paper examines the use of fixed point accelerators in **R**. The algorithms of the **FixedPoint** and **SQUAREM** packages are used to demonstrate the use of fixed point acceleration algorithms in the solution of numerical mathematics problems. A number of applications were shown. First the package was used to accelerate the finding of an equilibrium distribution of gas in a diffusion setting. The package was then used to accelerate the training of a perceptron classifier. The acceleration of this training was complicated by the training function converging in fixed increments however

[10] The starting guesses, convergence criteria, etc can also be found in the test files for **FixedPoint** which are included with the package's source files. The squarem method provided in the **SQUAREM** package checks for convergence in a different way to the **FixedPoint** package. To overcome this the convergence target was adjusted for this package so that in general the squarem achieves slightly less convergence than the **FixedPoint** methods in the convergence tests in this table which results in any bias being slightly in favor of the squarem method.

Case	Dimensions	Function
1	1	Babylonian Square Root
2	1	cos(x)
3	6	$x^{1/3}$
4	6	$95(18 - x)$
5	2	Simple Vector Function
6	100	Gas Diffusion
7	3	Perceptron
8	3	Modified Perceptron
9	10	Equilibrium Prices
10	100	Perpetual American Put
11	107	Consumption Smoothing

Case	Simple	Anderson	Aitken	Newton	VEA	SEA	MPE	RRE	squarem
1	6	7	7	7	6	6	6	6	6
2	58	7	11	9	13	13	19	25	55
3	22	12	9	9	13	13	9	10	12
4	*	5	3	3	25	*	19	7	*
5	105	14	67	239	20	25	31	31	44
6	221	26	323	*	150	221	44	50	159
7	1121	*	*	*	*	*	*	*	*
8	1156	*	*	20	75	158	54	129	638
9	11	9	14	24	11	11	11	12	11
10	103	37	203	*	108	103	43	52	103

it was possible to speed up the solution using a fixed point accelerator by changing the training algorithm while retaining the same set of fixed points. A number of problems in economics were then examined. First the equilibrium price vector was found for a pure exchange economy. Next a vector was found that gives the price of a perpetual American put option at various values of the underlying asset's spot price. Finally the future value function was found for an infinitely lived consumer facing a consumption smoothing problem.

In all of these example applications it can be noted that the solving for a fixed point was accelerated significantly by the use of a fixed point acceleration algorithm. In many cases an accelerator was available that was more than an order of magnitude faster than the simple method. The results indicate that large speedups are available to **R** programmers that are able to apply fixed point acceleration algorithms to their numerical problem of interest.

Bibliography

D. G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM*, 12(4): 547–560, 1965. URL `https://doi.org/10.1145/321296.321305`. [p362]

S. Baumann and M. Klymak. *Schumaker: Schumaker Shape-Preserving Spline*, 2017. URL `https://CRAN.R-project.org/package=schumaker`. R package version 1.0. [p370]

S. Baumann and M. Klymak. *FixedPoint: Algorithms for Finding Fixed Point Vectors of Functions*, 2018. URL `https://cran.r-project.org/package=FixedPoint`. [p359]

J. F. Bobb and R. Varadhan. *turboEM: A Suite of Convergence Acceleration Schemes for EM, MM and Other Fixed-Point Algorithms*, 2014. URL `https://CRAN.R-project.org/package=turboEM`. R package version 2014.8-1. [p359]

S. Cabay and L. W. Jackson. A polynomial extrapolation method for finding limits and antilimits of vector sequences. *Siam Journal of Numerical Analysis*, 13(5):734–752, 1976. URL `https://doi.org/10.1137/0713060`. [p361]

N. Henderson and R. Varadhan. *Daarem: Damped Anderson Acceleration with Epsilon Monotonicity for Accelerating EM-Like Monotone Algorithms*, 2018. URL `https://cran.r-project.org/package=daarem`. [p359]

K. Jbilou and H. Sadok. Vector extrapolation methods. applications and numerical comparison. *Journal of Computational and Applied Mathematics*, 122(1-2):149–165, 2000. URL `https://doi.org/10.1016/S0377-0427(00)00357-5`. [p371]

W. R. Mebane, Jr. and J. S. Sekhon. Genetic optimization using derivatives: The rgenoud package for R. *Journal of Statistical Software*, 42(11):1–26, 2011. URL `https://doi.org/10.18637/jss.v042.i11`. [p359]

Microsoft Corporation and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2017. URL `https://CRAN.R-project.org/package=doParallel`. R package version 1.0.11. [p371]

B. Narasimhan and S. G. Johnson. *Cubature: Adaptive Multivariate Integration over Hypercubes*, 2017. URL `https://CRAN.R-project.org/package=cubature`. R package version 1.3-11. [p370]

A. Novikoff. On convergence proofs for perceptrons. *Stanford Research Institute: Technical Report*, 298258, 1963. [p367]

Revolution Analytics and S. Weston. *Foreach: Provides Foreach Looping Construct for R*, 2015. URL `https://CRAN.R-project.org/package=foreach`. R package version 1.4.3. [p371]

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. URL `https://doi.org/10.1037/h0042519`. [p366]

O. Roustant, D. Ginsbourger, and Y. Deville. DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1):1–55, 2012. URL `http://doi.org/10.18637/jss.v051.i01`. [p359]

D. Smith, W. Ford, and A. Sidi. Extrapolation methods for vector sequences. *SIAM Review*, 29(2): 199–233, 1987. URL `https://doi.org/10.1137/1029042`. [p361]

N. Stokey, R. E. Lucas, and E. Prescott. *Recursive Methods in Economic Dynamics.* Harvard University Press, 1989. ISBN 9780674750968. [p360]

R. Varadhan. *SQUAREM: Squared Extrapolation Methods for Accelerating EM-Like Monotone Algorithms*, 2010. URL https://CRAN.R-project.org/package=SQUAREM. R package version 2017.10-1. [p359]

R. Varadhan and P. Gilbert. BB: An R package for solving a large system of nonlinear equations and for optimizing a high-dimensional nonlinear objective function. *Journal of Statistical Software*, 32(4):1–26, 2009. URL https://doi.org/10.18637/jss.v032.i04. [p359]

R. Varadhan and C. Roland. Simple and globally convergent methods for accelerating the convergence of any em algorithm. *Scandinavian Journal of Statistics*, 35(2):335–353, 2008. URL https://doi.org/10.1111/j.1467-9469.2007.00585.x. [p362]

P. Wynn. Acceleration techniques for iterated vector and matrix problems. *Mathematics of Computation*, 16(79):301–322, 1962. URL https://doi.org/10.2307/2004051. [p361]

Appendix: An algorithm for finding a fixed point while using parallelisation

```
library(foreach)
library(doParallel)
cores = 6

NodeTaskAssigner = function(Inputs, Outputs, i, Function){
  library(FixedPoint)
  library(schumaker)
  library(cubature)
  Iterates = dim(Inputs)[2]
  if (i > 1.5) {IterateToDrop = Iterates-i+1} else {IterateToDrop = 0}
  IteratesToUse = (1:Iterates)[ 1:Iterates != IterateToDrop]
  Inputs = matrix(Inputs[,IteratesToUse], ncol = length(IteratesToUse), byrow = FALSE)
  Outputs = matrix(Outputs[,IteratesToUse], ncol = length(IteratesToUse), byrow = FALSE)
  Guess = FixedPointNewInput(Inputs = Inputs, Outputs = Outputs, Method = "Anderson")
  Outputs = matrix(Function(Guess), ncol = 1, byrow = FALSE)
  Inputs = matrix(Guess, ncol = 1, byrow = FALSE)

  return(list(Inputs = Inputs, Outputs = Outputs))
}

# This combines the results returned by each node
CombineLists = function(List1, List2){
  width = dim(List1$Inputs)[2] + dim(List2$Inputs)[2]
  C = list()
  C$Inputs  = matrix(c(List1$Inputs , List2$Inputs ), ncol = width, byrow = FALSE)
  C$Outputs = matrix(c(List1$Outputs, List2$Outputs), ncol = width, byrow = FALSE)
  return(C)
}

# ReSortIterations
# This function takes the previous inputs and outputs from the function, removes
# duplicates and then sorts them in order of increasing convergence.
ReSortIterations = function(PreviousIterates,
                            ConvergenceMetric = function(Resids){max(abs(Resids))})
{
  # Removing any duplicates
  NotDuplicated   = (!(duplicated.matrix(PreviousIterates$Inputs, MARGIN = 2)))
  PreviousIterates$Inputs  = PreviousIterates$Inputs[,NotDuplicated]
  PreviousIterates$Outputs = PreviousIterates$Outputs[,NotDuplicated]
  # Resorting
  Resid       = PreviousIterates$Outputs - PreviousIterates$Inputs
  Convergence = ConvergenceVector = sapply(1:(dim(Resid)[2]), function(x)
    ConvergenceMetric(Resid[,x]) )
  Reordering  = order(Convergence, decreasing = TRUE)
  PreviousIterates$Inputs  = PreviousIterates$Inputs[,Reordering]
  PreviousIterates$Outputs = PreviousIterates$Outputs[,Reordering]
  return(PreviousIterates)
}


ConvergenceMetric = function(Resid){max(abs(Resid))}

# Preparing for clustering and getting a few runs to input to later functions:
PreviousRuns = FixedPoint(Function = IterateOnce, Inputs = InitialGuess,
                          Method = "Anderson", MaxIter = cores)
PreviousRuns$Residuals = PreviousRuns$Outputs - PreviousRuns$Inputs
PreviousRuns$Convergence = apply(PreviousRuns$Residuals, 2, ConvergenceMetric)
ConvergenceVal = min(PreviousRuns$Convergence)

registerDoParallel(cores=cores)

iter = cores
while (iter < 100 & ConvergenceVal > 1e-10){
  NewRuns = foreach(i = 1:cores, .combine=CombineLists) %dopar% {
    NodeTaskAssigner(PreviousRuns$Inputs, PreviousRuns$Outputs, i, IterateOnce)
  }
  # Appending to previous runs
  PreviousRuns$Inputs = matrix(c(PreviousRuns$Inputs, NewRuns$Inputs),
                     ncol = dim(PreviousRuns$Inputs)[2] + cores, byrow = FALSE)
  PreviousRuns$Outputs = matrix(c(PreviousRuns$Outputs, NewRuns$Outputs),
                     ncol = dim(PreviousRuns$Outputs)[2] + cores, byrow = FALSE)
  PreviousRuns = ReSortIterations(PreviousRuns)
  PreviousRuns$Residuals = PreviousRuns$Outputs - PreviousRuns$Inputs
  PreviousRuns$Convergence = apply(PreviousRuns$Residuals, 2, ConvergenceMetric)
  # Finding Convergence
  ConvergenceVal = min(PreviousRuns$Convergence)
  iter = iter + cores
}

stopImplicitCluster()
# And the fixed point comes out to be:
PreviousRuns$Outputs[, dim(PreviousRuns$Outputs)[2]]
```

Stuart Baumann
ORCiD: 0000-0002-9657-0969
stuart@stuartbaumann.com

Margaryta Klymak
University of Oxford
ORCiD: 0000-0003-4376-883X
margaryta.klymak@qeh.ox.ac.uk

SemiCompRisks: An R Package for the Analysis of Independent and Cluster-correlated Semi-competing Risks Data

by Danilo Alvares, Sebastien Haneuse, Catherine Lee, and Kyu Ha Lee

Abstract Semi-competing risks refer to the setting where primary scientific interest lies in estimation and inference with respect to a non-terminal event, the occurrence of which is subject to a terminal event. In this paper, we present the R package **SemiCompRisks** that provides functions to perform the analysis of independent/clustered semi-competing risks data under the illness-death multi-state model. The package allows the user to choose the specification for model components from a range of options giving users substantial flexibility, including: accelerated failure time or proportional hazards regression models; parametric or non-parametric specifications for baseline survival functions; parametric or non-parametric specifications for random effects distributions when the data are cluster-correlated; and, a Markov or semi-Markov specification for terminal event following non-terminal event. While estimation is mainly performed within the Bayesian paradigm, the package also provides the maximum likelihood estimation for select parametric models. The package also includes functions for univariate survival analysis as complementary analysis tools.

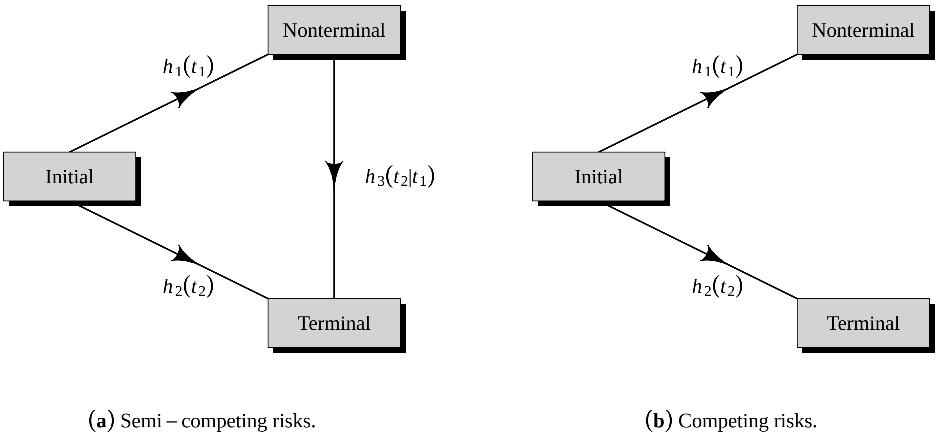
Introduction

Semi-competing risks refer to the general setting where primary scientific interest lies in estimation and inference with respect to a non-terminal event (e.g., disease diagnosis), the occurrence of which is subject to a terminal event (e.g., death) (Fine et al., 2001; Jazić et al., 2016). When there is a strong association between two event times, naïve application of a univariate survival model for non-terminal event time will result in overestimation of outcome rates as the analysis treats the terminal event as an independent censoring mechanism (Haneuse and Lee, 2016). The semi-competing risks analysis framework appropriately treats the terminal event as a competing event and considers the dependence between non-terminal and terminal events as part of the model specification.

Toward formally describing the structure of semi-competing risks data, let T_1 and T_2 denote the times to the non-terminal and terminal events, respectively. From the modeling perspective, the focus in the semi-competing risks setting is to characterize the distribution T_1 and its potential relationship with the distribution of T_2, i.e. the joint distribution of (T_1, T_2). For example, from an initial state (e.g., transplantation), as time progresses, a subject could make a transition into the non-terminal or terminal state (see Figure 1.a). In the case of a transition into the non-terminal state, the subject could subsequently transition into the terminal state even if these transitions cannot occur in the reverse order. The main disadvantage of the competing risks framework (see Figure 1.b) to the study of non-terminal event is that it does not utilize the information on the occurrence and timing of terminal event following the non-terminal event, which could be used to understand the dependence between the two events.

The current literature for the analysis of semi-competing risks data is composed of three approaches: methods that specify the dependence between non-terminal and terminal events via a copula (Fine et al., 2001; Wang, 2003; Jiang et al., 2005; Ghosh, 2006; Peng and Fine, 2007; Lakhal et al., 2008; Hsieh et al., 2008; Fu et al., 2013); methods based on multi-state models, specifically the so-called *illness-death* model (Liu et al., 2004; Putter et al., 2007; Ye et al., 2007; Kneib and Hennerfeind, 2008; Zeng and Lin, 2009; Xu et al., 2010; Zeng et al., 2012; Han et al., 2014; Zhang et al., 2014; Lee et al., 2015, 2016); and methods built upon the principles of causal inference (Zhang and Rubin, 2003; Egleston et al., 2007; Tchetgen Tchetgen, 2014; Varadhan et al., 2014).

The **SemiCompRisks** package is designed to provide a comprehensive suite of functions for the analysis of semi-competing risks data based on the illness-death model, together with, as a complementary suite of tools, functions for the analysis of univariate time-to-event data. While Bayesian methods are used for estimation and inference for all available models, maximum likelihood estimation is also provided for select parametric models. Furthermore, **SemiCompRisks** offers flexible parametric and non-parametric specifications for baseline survival functions and cluster-specific random effects distributions under accelerated failure time and proportional hazards models. The functionality of the package covers methods proposed in a series of recent papers on the analysis of



(a) Semi – competing risks.

(b) Competing risks.

Figure 1: Graphical representation of **(a)** semi-competing risks and **(b)** competing risks.

semi-competing risks data (Lee et al., 2015, 2016, 2017c).

The remainder of the paper is organized as follows. Section Other packages and their features summarizes existing R packages that provide methods for multi-state modeling, and explains the key contributions of the **SemiCompRisks** package. Section Datasets introduces an on-going study of stem cell transplantation and provides a description of the data available in the package. Section The illness-death models for semi-competing risks data presents different specifications of models and estimation methods implemented in our package. Section Package description summarizes the core components of the **SemiCompRisks** package, including datasets, functions for fitting models, functions, the structure of output provided to analysts. Section Illustration: Stem cell transplantation data illustrates the usage of the main functions in the package through three semi-competing risks analyses of the stem cell transplantation data. Finally, Section Discussion concludes with discussion and an overview of the extensions we are working on.

Other packages and their features

As we elaborate upon below, the illness-death model for semi-competing risks, that is the focus on the **SemiCompRisks** package, is a special case of the broader class of multi-state models. Currently, there are numerous R packages that permit estimation and inference for a multi-state model and that could conceivably be used to analyze semi-competing risks data.

The **mvna** package computes the Nelson-Aalen estimator of the cumulative transition hazard for arbitrary Markov multi-state models with right-censored and left-truncated data, but it does not compute transition probability matrices (Allignol et al., 2008). The **TPmsm** implements non-parametric and semi-parametric estimators for the transition probabilities in 3-state models, including the Aalen-Johansen estimator and estimators that are consistent even without Markov assumption or in case of dependent censoring (Araújo et al., 2014). The **p3state.msm** package performs inference in an illness-death model (Meira-Machado and Roca-Pardiñas, 2011). Its main feature is the ability for obtaining non-Markov estimates for the transition probabilities. The **etm** package calculates the empirical transition probability matrices and corresponding variance estimates for any time-inhomogeneous multi-state model with finite state space and data subject to right-censoring and left-truncation, but it does not account for the influence of covariates (Allignol et al., 2011). The **msm** package is able to fit time-homogeneous Markov models to panel count data and hidden Markov models in continuous time (Jackson, 2011). The time-homogeneous Markov approach could be a particular case of the illness-death model, where interval-censored data can be considered. The **tdc.msm** package may be used to fit the time-dependent proportional hazards model and multi-state regression models in continuous time, such as Cox Markov model, Cox semi-Markov model, homogeneous Markov model, non-homogeneous piecewise model, and non-parametric Markov model (Meira-Machado et al., 2007). The **SemiMarkov** package performs parametric (Weibull or exponentiated Weibull specification) estimation in a homogeneous semi-Markov model (Król and Saint-Pierre, 2015). Moreover, the effects of covariates on the process evolution can be studied using a semi-parametric Cox model for the distributions of sojourn times. The **flexsurv** package provides functions for fitting and predicting from fully-parametric multi-state models with Markov or semi-Markov specification (Jackson, 2016). In addition, the multi-state models implemented in

flexsurv give the possibility to include interval-censoring and some of them also left-truncation. The **msSurv** calculates non-parametric estimation of general multi-state models subject to independent right-censoring and possibly left-truncation (Ferguson et al., 2012). This package also computes the marginal state occupation probabilities along with the corresponding variance estimates, and lower and upper confidence intervals. The **mstate** package can be applied to right-censored and left-truncated data in semi-parametric or non-parametric multi-state models with or without covariates and it may also be used to competing risk models (Wreede et al., 2011). Specifically for Cox-type illness-death models to interval-censored data, we highlight the packages **coxinterval** (Boruvka and Cook, 2015) and **SmoothHazard** (Touraine et al., 2017), where the latter also allows that the event times to be left-truncated. Finally, **frailtypack** package permits the analysis of correlated data under select clusterings, as well as the analysis of left-truncated data, through a focus on frailty models using penalized likelihood estimation or parametric estimation (Rondeau et al., 2012).

While these packages collectively provide broad functionality, each of them is either non-specific to semi-competing risks or only permits consideration of a narrow model specifications. In developing the **SemiCompRisks** package, the goal was to provide a single package within which a broad range of models and model specifications could be entertained. The **frailtypack** package, for example, can also be used to analyze cluster-correlated semi-competing risks data but it is restricted to the proportional hazards model with either patient-specific or cluster-specific random effects but not both (Liquet et al., 2012). Furthermore, estimation/inference is within the frequentist framework so that estimation of hospital-specific random effects, of particular interest in health policy applications (Lee et al., 2016), together with the quantification of uncertainty is incredibly challenging. This, however, is (relatively) easily achieved through the functionality of **SemiCompRisks** package. Given the breadth of the functionality of the package, in addition to the usual help files, we have developed a series of model-specific vignettes which can be accessed through the CRAN (Lee et al., 2017b) or R command `vignette("SemiCompRisks")`, covering a total of 12 distinct model specifications.

CIBMTR data

The example dataset used throughout this paper was obtained from the Center for International Blood and Marrow Transplant Research (CIBMTR), a collaboration between the National Marrow Donor Program and the Medical College of Wisconsin representing a worldwide network of transplant centers (Lee et al., 2017a). For illustrative purposes, we consider a hypothetical study in which the goal is to investigate risk factors for grade III or IV acute graft-versus-host disease (GVHD) among $9,651$ patients who underwent the first allogeneic hematopoietic cell transplant (HCT) between January 1999 and December 2011.

As summarized in Table 1, after administratively censoring follow-up at 365 days post-transplant, each patient can be categorized according to their observed outcome information into four groups: (i) acute GVHD and death; (ii) acute GVHD and censored for death; (iii) death without acute GVHD; and (iv) censored for both. Furthermore, for each patient, the following covariates are available:gender (Male, Female); age (<10, 10-19, 20-29, 30-39, 40-49, 50-59, 60+); disease type (AML, ALL, CML, MDS); disease stage (Early, Intermediate, Advanced); and HLA compatibility (Identical sibling, 8/8, 7/8).

We note that due to confidentiality considerations the original study outcomes (`time1`, `time2`, `event1`, `event2`: times and censoring indicators to the non-terminal and terminal events) are not available in **SemiCompRisks** package. As such we provide the five original covariates together with estimates of parameters from the analysis of CIBMTR data, so that one could simulate semi-competing risks outcomes (see the simulation procedure in Appendix Simulating outcomes using CIBMTR covariates). Based on this, the data shown in Table 1 reflects simulated outcome data using 1405 as the seed.

The illness-death models for semi-competing risks data

We offer three flexible multi-state illness-death models for the analysis of semi-competing risks data: accelerated failure time (AFT) models for independent data; proportional hazards regression (PHR) models for independent data; and PHR models for cluster-correlated data. These models accommodate parametric or non-parametric specifications for baseline survival functions as well as a Markov or semi-Markov assumptions for terminal event following non-terminal event.

AFT models for independent semi-competing risks data

In the AFT model specification, we directly model the connection between event times and covariates (Wei, 1992). For the analysis of semi-competing risks data, we consider the following AFT model specifications under the illness-death modeling framework (Lee et al., 2017c):

$$\log(T_{i1}) = \boldsymbol{x}_{i1}^\top \boldsymbol{\beta}_1 + \gamma_i + \epsilon_{i1}, \quad T_{i1} > 0, \tag{B.4.1}$$

$$\log(T_{i2}) = \boldsymbol{x}_{i2}^\top \boldsymbol{\beta}_2 + \gamma_i + \epsilon_{i2}, \quad T_{i2} > 0, \tag{B.4.2}$$

$$\log(T_{i2} - T_{i1}) = \boldsymbol{x}_{i3}^\top \boldsymbol{\beta}_3 + \gamma_i + \epsilon_{i3}, \quad T_{i2} > T_{i1}, \tag{B.4.3}$$

where T_{i1} and T_{i2} denote the times to the non-terminal and terminal events, respectively, from subject $i = 1, \ldots, n$, $\boldsymbol{x}_{ig}$ is a vector of transition-specific covariates, $\boldsymbol{\beta}_g$ is a corresponding vector of transition-specific regression parameters, and ϵ_{ig} is a transition-specific random variable whose distribution determines that of the corresponding transition time, $g \in \{1, 2, 3\}$. Finally, in each of (B.4.1)-(B.4.3), γ_i is a study subject-specific random effect that induces positive dependence between the two event times. We assume that γ_i follows a Normal(0, θ) distribution and adopt a conjugate inverse Gamma distribution, denoted by IG$(a^{(\theta)}, b^{(\theta)})$ for the variance component θ. For regression parameters $\boldsymbol{\beta}_g$, we adopt non-informative flat prior on the real line.

From models (B.4.1)-(B.4.3), we can adopt either a fully parametric or a semi-parametric approach depending on the specification of the distributions for ϵ_{i1}, ϵ_{i2}, ϵ_{i3}. We build a parametric modeling based on the log-Normal formulation, where ϵ_{ig} follows a Normal(μ_g, σ_g^2) distribution. We adopt non-informative flat priors on the real line for μ_g and independent IG$(a_g^{(\sigma)}, b_g^{(\sigma)})$ for σ_g^2. As an alternative, a semi-parametric framework can be considered by adopting independent non-parametric Dirichlet process mixtures (DPM) of M_g Normal$(\mu_{gr}, \sigma_{gr}^2)$ distributions, $r \in \{1, \ldots, M_g\}$, for each ϵ_{ig}. Following convention in the literature, we refer to each component Normal distribution as being specific to some "class" (Neal, 2000). Since the class-specific $(\mu_{gr}, \sigma_{gr}^2)$ are unknown, they are assumed to be draws from a so-called the *centering distribution*. Specifically, we take a Normal distribution centered at μ_{g0} with a variance σ_{g0}^2 for μ_{gr} and an IG$(a_g^{(\sigma_{gr})}, b_g^{(\sigma_{gr})})$ for σ_{gr}^2. Furthermore, since the "true" class membership for any given study subject is unknown, we let p_{gr} denote the probability of belonging to the rth class for transition g and $\boldsymbol{p}_g = (p_{g1}, \ldots, p_{gM_g})^\top$ the collection of such probabilities. In the absence of prior knowledge regarding the distribution of class memberships for the n subjects across the M_g classes, $\boldsymbol{p}_g$ is assumed to follow a conjugate symmetric Dirichlet$(\tau_g/M_g, \ldots, \tau_g/M_g)$ distribution, where τ_g is referred to as the *precision parameter* (for more details, see Lee et al., 2017c).

Our AFT modeling framework can also handle interval-censored and/or left-truncated semi-competing risks data. Suppose that subject i was observed at follow-up times $\{c_{i1}, \ldots, c_{im_i}\}$ and let c_i^* and L_i denote the time to the end of study (or administrative right-censoring) and the time at study entry (i.e., the left-truncation time), respectively. Considering interval-censoring for both events, T_{i1} and T_{i2}, for $i = 1, \ldots, n$, satisfy $c_{ij} \leqslant T_{i1} < c_{ij+1}$ for some j and $c_{ik} \leqslant T_{i2} < c_{ik+1}$ for some k, respectively. Therefore, the observed outcome information for interval-censored and left-truncated semi-competing risks data for the subject i can be represented by $\{L_i, c_{ij}, c_{ij+1}, c_{ik}, c_{ik+1}\}$.

PHR models for independent semi-competing risks data

We consider an illness-death multi-state model with proportional hazards assumptions characterized by three hazard functions (see Figure 1.a) that govern the rates at which subjects transition between the states: a cause-specific hazard for non-terminal event, $h_1(t_{i1})$; a cause-specific hazard for terminal event, $h_2(t_{i2})$; and a hazard for terminal event conditional on a time for non-terminal event, $h_3(t_{i2} \mid t_{i1})$. We consider the following specification for hazard functions (Xu et al., 2010; Lee et al., 2015):

$$h_1(t_{i1} \mid \gamma_i, \boldsymbol{x}_{i1}) = \gamma_i \, h_{01}(t_{i1}) \exp(\boldsymbol{x}_{i1}^\top \boldsymbol{\beta}_1), \quad t_{i1} > 0, \tag{B.4.4}$$

$$h_2(t_{i2} \mid \gamma_i, \boldsymbol{x}_{i2}) = \gamma_i \, h_{02}(t_{i2}) \exp(\boldsymbol{x}_{i2}^\top \boldsymbol{\beta}_2), \quad t_{i2} > 0, \tag{B.4.5}$$

$$h_3(t_{i2} \mid t_{i1}, \gamma_i, \boldsymbol{x}_{i3}) = \gamma_i \, h_{03}(z(t_{i1}, t_{i2})) \exp(\boldsymbol{x}_{i3}^\top \boldsymbol{\beta}_3), \quad t_{i2} > t_{i1}, \tag{B.4.6}$$

where h_{0g} is an unspecified baseline hazard function and $\boldsymbol{\beta}_g$ is a vector of log-hazard ratio regression parameters associated with the covariates $\boldsymbol{x}_{ig}$. Finally, in each of (B.4.4)-(B.4.6), γ_i is a study subject-specific shared frailty following a Gamma$(\theta^{-1}, \theta^{-1})$ distribution, parametrized so that $E[\gamma_i] = 1$ and $V[\gamma_i] = \theta$. The model (B.4.6) is referred to as being Markov or semi-Markov depending on whether we assume $z(t_{i1}, t_{i2}) = t_{i2}$ or $z(t_{i1}, t_{i2}) = t_{i2} - t_{i1}$, respectively.

The Bayesian approach for models (B.4.4)-(B.4.6) requires the specification of prior distributions for unknown parameters. For the regression parameters $\boldsymbol{\beta}_g$, we adopt a non-informative flat prior distribution on the real line. For the variance in the subject-specific frailties, θ, we adopt a Gamma$(a^{(\theta)}, b^{(\theta)})$ for the precision θ^{-1}. For the parametric specification for baseline hazard functions, we consider a Weibull model: $h_{0g}(t) = \alpha_g\, \kappa_g\, t^{\alpha_g - 1}$. We assign a Gamma$(a_g^{(\alpha)}, b_g^{(\alpha)})$ for α_g and a Gamma$(c_g^{(\kappa)}, d_g^{(\kappa)})$ for κ_g. As an alternative, a non-parametric piecewise exponential model (PEM) is considered for baseline hazard functions based on taking each of the log-baseline hazard functions to be a flexible mixture of piecewise constant function. Let $s_{g,\text{max}}$ denote the largest observed event time for each transition and construct a finite partition of the time axis, $0 = s_{g,0} < s_{g,1} < s_{g,2} < \ldots < s_{g,K_g+1} = s_{g,\text{max}}$. Letting $\boldsymbol{\lambda}_g = (\lambda_{g,1}, \ldots, \lambda_{g,K_g}, \lambda_{g,K_g+1})^\top$ denote the heights of the log-baseline hazard function on the disjoint intervals based on the time splits $\boldsymbol{s}_g = (s_{g,1}, \ldots, s_{g,K_g+1})^\top$, we assume that $\boldsymbol{\lambda}_g$ follows a multivariate Normal distribution (MVN), MVN$(\mu_{\lambda_g}\mathbf{1}, \sigma^2_{\lambda_g}\boldsymbol{\Sigma}_{\lambda_g})$, where μ_{λ_g} is the overall mean, $\sigma^2_{\lambda_g}$ represents a common variance component for the $K_g + 1$ elements, and $\boldsymbol{\Sigma}_{\lambda_g}$ specifies the covariance structure these elements. We adopt a flat prior on the real line for μ_{λ_g} and a conjugate Gamma$(a_g^{(\sigma)}, b_g^{(\sigma)})$ distribution for the precision $\sigma^{-2}_{\lambda_g}$. In order to relax the assumption of fixed partition of the time scales, we adopt a Poisson$(\alpha_g^{(K)})$ prior for the number of splits, K_g, and conditioned on the number of splits, we consider locations, $\boldsymbol{s}_g$, to be *a priori* distributed as the even-numbered order statistics:

$$\pi(\boldsymbol{s}_g \mid K_g) \propto \frac{(2K_g+1)! \prod_{k=1}^{K_g+1}(s_{g,k} - s_{g,k-1})}{(s_{g,K_g+1})^{2K_g+1}}. \tag{B.4.7}$$

Note that the prior distributions of K_g and $\boldsymbol{s}_g$ jointly form a time-homogeneous Poisson process prior for the partition $(K_g, \boldsymbol{s}_g)$. For more details, see Lee et al. (2015).

PHR models for cluster-correlated semi-competing risks data

Lee et al. (2016) proposed hierarchical models that accommodate correlation in the joint distribution of the non-terminal and terminal events across patients for the setting where patients are clustered within hospitals. The hierarchical models for cluster-correlated semi-competing risks data build upon the illness-death model given in (B.4.4)-(B.4.6). Let T_{ji1} and T_{ji2} denote the times to the non-terminal and terminal events for the ith subject in the jth cluster, respectively, for $i = 1, \ldots, n_j$ and $j = 1, \ldots, J$. The general modeling specification is given by:

$$h_1(t_{ji1} \mid \gamma_{ji}, \boldsymbol{x}_{ji1}, V_{j1}) = \gamma_{ji}\, h_{01}(t_{ji1}) \exp(\boldsymbol{x}_{ji1}^\top \boldsymbol{\beta}_1 + V_{j1}), \quad t_{ji1} > 0, \tag{B.4.8}$$

$$h_2(t_{ji2} \mid \gamma_{ji}, \boldsymbol{x}_{ji2}, V_{j2}) = \gamma_{ji}\, h_{02}(t_{ji2}) \exp(\boldsymbol{x}_{ji2}^\top \boldsymbol{\beta}_2 + V_{j2}), \quad t_{ji2} > 0, \tag{B.4.9}$$

$$h_3(t_{ji2} \mid t_{ji1}, \gamma_{ji}, \boldsymbol{x}_{ji3}, V_{j3}) = \gamma_{ji}\, h_{03}(z(t_{ji1}, t_{ji2})) \exp(\boldsymbol{x}_{ji3}^\top \boldsymbol{\beta}_3 + V_{j3}), \quad t_{ji2} > t_{ji1}, \tag{B.4.10}$$

where h_{0g} is an unspecified baseline hazard function and $\boldsymbol{\beta}_g$ is a vector of log-hazard ratio regression parameters associated with the covariates $\boldsymbol{x}_{jig}$. A study subject-specific shared frailty γ_{ji} is assumed to follow a Gamma$(\theta^{-1}, \theta^{-1})$ distribution and $\boldsymbol{V}_j = (V_{j1}, V_{j2}, V_{j3})^\top$ is a vector of cluster-specific random effects, each specific to one of the three possible transitions.

From a Bayesian perspective for models (B.4.8)-(B.4.10), we can adopt either a parametric Weibull or non-parametric PEM specification for baseline hazard functions h_{0g} with their respective configurations of prior distributions analogous to those outlined in Section PHR models for independent semi-competing risks data. For the parametric specification of cluster-specific random effects, we assume that $\boldsymbol{V}_j$ follows MVN$_3(\mathbf{0}, \boldsymbol{\Sigma}_V)$ distribution. We adopt a conjugate inverse-Wishart$(\boldsymbol{\Psi}_v, \rho_v)$ prior for the variance-covariance matrix $\boldsymbol{\Sigma}_V$. For the non-parametric specification, we adopt a DPM of MVN distributions with a centering distribution, G_0, and a precision parameter, τ. Here we take G_0 to be a multivariate Normal/inverse-Wishart (NIW) distribution for which the probability density function can be expressed as the product:

$$f_{\text{NIW}}(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \boldsymbol{\Psi}_0, \rho_0) = f_{\text{MVN}}(\boldsymbol{\mu} \mid \mathbf{0}, \boldsymbol{\Sigma}) \times f_{\text{inverse-Wishart}}(\boldsymbol{\Sigma} \mid \boldsymbol{\Psi}_0, \rho_0), \tag{B.4.11}$$

where $\boldsymbol{\Psi}_0$ and ρ_0 are the hyperparameters of $f_{\text{NIW}}(\cdot)$. We assign a Gamma(a_τ, b_τ) prior distribution for τ. Finally, for $\boldsymbol{\beta}_g$ and θ, we adopt the same priors as those adopted for the model in Section PHR models for independent semi-competing risks data. For more details, see Lee et al. (2016).

Estimation and inference

Bayesian estimation and inference is available for all models in the **SemiCompRisks**. Additionally, one may also choose to use maximum likelihood estimation for the parametric Weibull PHR model described in Section PHR models for independent semi-competing risks data.

To perform Bayesian estimation and inference, we use a random scan Gibbs sampling algorithm to generate samples from the full posterior distribution. Depending on the complexity of the model adopted, the Markov chain Monte Carlo (MCMC) scheme may also include additional strategies, such as Metropolis-Hastings and reversible jump MCMC (Metropolis-Hastings-Green) steps. Specific details of each implementation can be seen in the online supplemental materials of Lee et al. (2015, 2016, 2017c).

Package description

The **SemiCompRisks** package contains three key functions, `FreqID_HReg`, `BayesID_HReg` and `BayesID_AFT`, focused on models for semi-competing risks data as well as the analogous univariate survival models, `FreqSurv_HReg`, `BayesSurv_HReg` and `BayesSurv_AFT`. It also provides two auxiliary functions, `initiate.startValues_HReg` and `initiate.startValues_AFT`, that can be used to generate initial values for Bayesian estimation; `simID` and `simSurv` functions for simulating semi-competing risks and univariate survival data, respectively; five covariates and parameter estimates from `CIBMTR` data; and the `BMT` dataset referring to 137 bone marrow transplant patients.

Summary of functionality

Table 2 shows the modeling options implemented in the **SemiCompRisks** package for both semi-competing risks and univariate analysis. Specifically, we categorize the approaches based on the analysis type (semi-competing risks or univariate), the survival model (AFT or PHR), data type (independent or clustered), accommodation to left-truncation and/or interval-censoring in addition to right-censoring, and also statistical paradigms (frequentist or Bayesian).

The full description of functionality of the **SemiCompRisks** package can be accessed through the R command `help("SemiCompRisks")` or `vignette("SemiCompRisks")` which provides in detail the specification of all models implemented in the package. Below we describe the input data format and some crucial arguments for defining and fitting a model for semi-competing risks data using the **SemiCompRisks** package.

Model specification

From a semi-competing risks dataset, we jointly define the outcomes and covariates in a `Formula` object. Here we use the `simCIBMTR` dataset, obtained from the simulation procedure presented in Appendix Simulating outcomes using CIBMTR covariates:

```
R> form <- Formula(time1 + event1 | time2 + event2 ~ dTypeALL + dTypeCML +
+     dTypeMDS + sexP | dTypeALL + dTypeCML + dTypeMDS | dTypeALL +
+     dTypeCML + dTypeMDS)
```

The outcomes `time1`, `time2`, `event1` and `event2` denote the times and censoring indicators to the non-terminal and terminal events, respectively, and the covariates of each hazard function are separated by | (vertical bar).

The specification of the `Formula` object varies slightly if the semi-competing risks model accommodates left-truncated and/or interval-censored data (see vignette documentation Lee et al. (2017b)).

Critical arguments

Most functions for semi-competing risks analysis in the **SemiCompRisks** package take common arguments. These arguments and their descriptions are shown as follows:

- `id`: a vector of cluster information for n subjects, where cluster membership corresponds to one of the positive integers $1, \ldots, J$.
- `model`: a character vector that specifies the type of components in a model. It can have up to three elements depending on the model specification. The first element is for the assumption on h_3: "semi-Markov" or "Markov". The second element is for the specification of baseline hazard functions for PHR models - "Weibull" or "PEM" - or baseline survival distribution for AFT models - "LN" (log-Normal) or "DPM". The third element needs to be set only for clustered semi-competing risks data and is for the specification of cluster-specific random effects distribution: "MVN" or "DPM".

- `hyperParams`: a list containing vectors for hyperparameter values in hierarchical models.
- `startValues`: a list containing vectors of starting values for model parameters.
- `mcmcParams`: a list containing variables required for MCMC sampling.

Hyperparameter values, starting values for model parameters, and MCMC arguments depend on the specified Bayesian model and the assigned prior distributions. For a list of illustrations, see vignette documentation Lee et al. (2017b).

FreqID_HReg

The function `FreqID_HReg` fits Weibull PHR models for independent semi-competing risks data, as in (B.4.4)-(B.4.6), based on maximum likelihood estimation. Its default structure is given by:

```
FreqID_HReg(Formula, data, model="semi-Markov", frailty=TRUE),
```

where `Formula` represents the outcomes and the linear predictors jointly, as presented in Section Summary of functionality; `data` is a data frame containing the variables named in `Formula`; `model` is one of the critical arguments of the **SemiCompRisks** package (see Section Summary of functionality), in which it specifies the type of model based on the assumption on $h_3(t_{i2} \mid t_{i1}, \cdot)$ in (B.4.6). Here, `model` can be `"Markov"` or `"semi-Markov"`. Finally, `frailty` is a logical value (`TRUE` or `FALSE`) to determine whether to include the subject-specific shared frailty term γ into the illness-death model.

BayesID_HReg

The function `BayesID_HReg` fits parametric and semi-parametric PHR models for independent or cluster-correlated semi-competing risks data, as in (B.4.4)-(B.4.6) or (B.4.8)-(B.4.10), based on Bayesian inference. Its default structure is given by:

```
BayesID_HReg(Formula, data, id=NULL, model=c("semi-Markov","Weibull"), hyperParams,
startValues, mcmcParams, path=NULL).
```

`Formula` and `data` are analogous to the previous case; `id`, `model`, `hyperParams`, `startValues`, and `mcmcParams` are all critical arguments of the **SemiCompRisks** package (see Section Summary of functionality), where `id` indicates the cluster that each subject belongs to (for independent data, `id=NULL`); `model` allows us to specify either `"Markov"` or `"semi-Markov"` assumption, whether the priors for baseline hazard functions are parametric (`"Weibull"`) or non-parametric (`"PEM"`), and whether the cluster-specific random effects distribution is parametric (`"MVN"`) or non-parametric (`"DPM"`). The third element of `model` is only required for models for clustered-correlated data given in (B.4.8)-(B.4.10).

The `hyperParams` argument defines all model hyperparameters: `theta` (a numeric vector for hyperparameters, $a^{(\theta)}$ and $b^{(\theta)}$, in the prior of subject-specific frailty variance component), `WB` (a list containing numeric vectors for Weibull hyperparameters $(a_g^{(\alpha)}, b_g^{(\alpha)})$ and $(c_g^{(\kappa)}, d_g^{(\kappa)})$ for $g \in \{1, 2, 3\}$: `WB.ab1`, `WB.ab2`, `WB.ab3`, `WB.cd1`, `WB.cd2`, `WB.cd3`), `PEM` (a list containing numeric vectors for PEM hyperparameters $(a_g^{(\sigma)}, b_g^{(\sigma)})$, and $\alpha_g^{(K)}$ for $g \in \{1, 2, 3\}$: `PEM.ab1`, `PEM.ab2`, `PEM.ab3`, `PEM.alpha1`, `PEM.alpha2`, `PEM.alpha3`); and for the analysis of clustered semi-competing risks data, additional components are required: `MVN` (a list containing numeric vectors for MVN hyperparameters $\mathbf{\Psi}_v$ and ρ_v: `Psi_v`, `rho_v`), `DPM` (a list containing numeric vectors for DPM hyperparameters $\mathbf{\Psi}_0$, ρ_0, a_τ, and b_τ: `Psi0`, `rho0`, `aTau`, `bTau`).

The `startValues` argument specifies initial values for model parameters. This specification can be done manually or through the auxiliary function `initiate.startValues_HReg`. The `mcmcParams` argument sets the information for MCMC sampling: `run` (a list containing numeric values for setting for the overall run: `numReps`, total number of scans; `thin`, extent of thinning; `burninPerc`, the proportion of burn-in), `storage` (a list containing numeric values for storing posterior samples for subject- and cluster-specific random effects: `nGam_save`, the number of γ to be stored; `storeV`, a vector of three logical values to determine whether all the posterior samples of $\boldsymbol{V}_j$, for $j = 1, \ldots, J$ are to be stored), `tuning` (a list containing numeric values relevant to tuning parameters for specific updates in Metropolis-Hastings-Green (MHG) algorithm: `mhProp_theta_var`, the variance of proposal density for θ; `mhProp_Vg_var`, the variance of proposal density for $\boldsymbol{V}_j$ in DPM models; `mhProp_alphag_var`, the variance of proposal density for α_g in Weibull models; `Cg`, a vector of three proportions that determine the sum of probabilities of choosing the birth and the death moves in PEM models (the sum of the three elements should not exceed 0.6); `delPertg`, the perturbation parameters in the birth update in PEM models (the values must be between 0 and 0.5); `rj.scheme`: if `rj.scheme`=1, the birth update will draw the proposal time split from 1:`sg_max` and if `rj.scheme`=2, the birth update will draw the proposal time split from uniquely ordered failure times in the data. For PEM models, additional components are required: `Kg_max`, the maximum number of splits allowed at each iteration in MHG algorithm for PEM models; `time_lambda1`, `time_lambda2`, `time_lambda3`, time points at which the posterior distribution of log-hazard functions are calculated. Finally, `path` indicates the name of directory where the results are saved. For more details and examples, see Lee et al. (2017b).

BayesID_AFT

The function `BayesID_AFT` fits parametric and semi-parametric AFT models for independent semi-competing risks data, given in (B.4.1)-(B.4.3), based on Bayesian inference. Its default structure is given by:

```
BayesID_AFT(Formula, data, model="LN", hyperParams, startValues, mcmcParams, path=NULL),
```

where `data`, `startValues` (auxiliary function `initiate.startValues_AFT`), and `path` are analogous to functions described in previous sections. Here, `Formula` has a different structure of outcomes, since the AFT model accommodates more complex censoring, such as interval-censoring and/or left-truncation (see Section AFT models for independent semi-competing risks data). It takes the generic form `Formula(LT | y1L + y1U | y2L + y2U   cov1 | cov2 | cov3)`, where `LT` represents the left-truncation time, (`y1L`, `y1U`) and (`y2L`, `y2U`) are the interval-censored times to the non-terminal and terminal events, respectively, and `cov1`, `cov2` and `cov3` are covariates of each linear regression. The `model` argument specifies whether the baseline survival distribution is parametric (`"LN"`) or non-parametric (`"DPM"`). The `hyperParams` argument defines all model hyperparameters: `theta` is for hyperparameters ($a^{(\theta)}$ and $b^{(\theta)}$)); `LN` is a list containing numeric vectors, `LN.ab1`, `LN.ab2`, `LN.ab3`, for log-Normal hyperparameters ($a_g^{(\sigma)}$, $b_g^{(\sigma)}$) with $g \in \{1,2,3\}$; `DPM` is a list containing numeric vectors, `DPM.mu1`, `DPM.mu2`, `DPM.mu3`, `DPM.sigSq1`, `DPM.sigSq2`, `DPM.sigSq3`, `DPM.ab1`, `DPM.ab2`, `DPM.ab3`, `Tau.ab1`, `Tau.ab2`, `Tau.ab3` for DPM hyperparameters (μ_{g0}, σ_{g0}^2), ($a_g^{(\sigma_{gr})}$, $b_g^{(\sigma_{gr})}$), and τ_g with $g \in \{1,2,3\}$. The `mcmcParams` argument sets the information for MCMC sampling: `run` (see Section BayesID_HReg), `storage` (`nGam_save`; `nY1_save`, the number of `y1` to be stored; `nY2_save`, the number of `y2` to be stored; `nY1.NA_save`, the number of `y1==NA` to be stored), `tuning` (`betag.prop.var`, the variance of proposal density for $\boldsymbol{\beta}_g$; `mug.prop.var`, the variance of proposal density for μ_g; `zetag.prop.var`, the variance of proposal density for $1/\sigma_g^2$; `gamma.prop.var`, the variance of proposal density for γ).

Univariate survival data analysis

The functions `FreqSurv_HReg`, `BayesSurv_HReg` and `BayesSurv_AFT` provide the same flexibility as functions `FreqID_HReg`, `BayesID_HReg` and `BayesID_AFT`, respectively, but in a univariate context (i.e., a single outcome).

The function `FreqSurv_HReg` fits a Weibull PHR model based on maximum likelihood estimation. This model is described by:

$$h(t_i \mid \boldsymbol{x}_i) = \alpha\, \kappa\, t_i^{\alpha-1} \exp(\boldsymbol{x}_i^\top \boldsymbol{\beta}), \quad t_i > 0. \tag{B.5.1}$$

The function `BayesSurv_HReg` implements Bayesian PHR models given by:

$$h(t_{ji} \mid \boldsymbol{x}_{ji}) = h_0(t_{ji}) \exp(\boldsymbol{x}_{ji}^\top \boldsymbol{\beta} + V_j), \quad t_i > 0. \tag{B.5.2}$$

We can adopt either a parametric Weibull or a non-parametric PEM specification for h_0. Cluster-specific random effects V_j, $j = 1, \ldots, J$, can be assumed to follow a parametric Normal distribution or a non-parametric DPM of Normal distributions.

Finally, the function `BayesSurv_AFT` implements Bayesian AFT models expressed by:

$$\log(T_i) = \boldsymbol{x}_i^\top \boldsymbol{\beta} + \epsilon_i, \quad T_i > 0, \tag{B.5.3}$$

where we can adopt either a fully parametric log-Normal or a non-parametric DPM specification for ϵ_i.

Summary output

The functions presented in Sections FreqID_HReg, BayesID_HReg and BayesID_AFT return objects of classes `Freq_HReg`, `Bayes_HReg` and `Bayes_AFT`, respectively. Each of these objects represents results from its respective semi-competing risks analysis. These results can be visualized using several R methods, such as `print`, `summary`, `predict`, `plot`, `coef`, and `vcov`.

The function `print` shows the estimated parameters and, in the Bayesian case, also the MCMC description (number of chains, scans, thinning, and burn-in) and the potential scale reduction factor (PSRF) convergence diagnostic for each model parameter (Gelman and Rubin, 1992; Brooks and Gelman, 1998). If the PSRF is close to 1, a group of chains have mixed well and have converged to a stable distribution. The function `summary` presents the regression parameters in exponential format (hazard ratios) and the estimated baseline hazard function components. Along with a summary of analysis results, the output from `summary` includes two model diagnostics and performance metrics, log-pseudo marginal likelihood (LPML) (Geisser and Eddy, 1979; Gelfand and Mallick, 1995) and deviance information criterion (DIC) (Spiegelhalter et al., 2002; Celeux et al., 2006), for Bayesian illness-death models.

Functions `predict` and `plot` complement each other. The former uses the fitted model to predict an output of interest (survival or hazard) at a given time interval from new covariates. From the object created by `predict`, `plot` displays survival (`plot.est="Surv"`) or hazard (`plot.est="Haz"`) functions with their respective credibility/confidence intervals. In order to predict the joint probability involving two event times for a new covariate profile, one can use the function `PPD`, which is calculated from the joint posterior predictive distribution of (T_1, T_2) (Lee et al., 2015).

SemiCompRisks also provides the standard functions `coef` (model coefficients) and `vcov` (variance-covariance matrix for a fitted frequentist model). For examples with more details, see Lee et al. (2017b).

Simulation of semi-competing risks data

The function `simID` simulates semi-competing risks outcomes from independent or cluster-correlated data (for more details of the simulation algorithm, see Appendix Simulation algorithm for semi-competing risks data). The simulation is based on a semi-Markov Weibull PHR modeling and, in the case of the cluster-correlated approach, the cluster-specific random effects follow a MVN distribution. We provide a simulation example of independent semi-competing risks data in Appendix Simulating outcomes using CIBMTR covariates.

Analogously, the function `simSurv` simulates univariate independent/cluster-correlated survival data under a Weibull PHR model with cluster-specific random effects following a Normal distribution.

Datasets

CIBMTR data. It is composed of 5 covariates that come from a study of acute GVHD with 9,651 patients who underwent the first allogeneic hematopoietic cell transplant between January 1999 and December 2011 (see Section Datasets).

BMT data. It refers to a well-known study of bone marrow transplantation for acute leukemia (Klein and Moeschberger, 2003). This data frame contains 137 patients with 22 variables and its description can be viewed from the R command `help(BMT)`.

Illustration: Stem cell transplantation data

To illustrate the usage of the **SemiCompRisks** package, we present two PHR models (one parametric model with maximum likelihood estimation and another semi-parametric model based on Bayesian inference) and one Bayesian AFT model using stem cell transplantation data, described in Section Datasets.

Frequentist analysis

Independent semi-Markov PHR model with Weibull baseline hazards

In our first example we employ the modeling (B.4.4)-(B.4.6) for independent data, semi-Markov assumption and Weibull baseline hazards. Here, `Formula` (`form`) is defined as in Section Summary of functionality. We fit the model using the function `FreqID_HReg`, described in Section FreqID_HReg, and visualize the results through the function `summary`:

```
R> fitFreqPHR <- FreqID_HReg(form, data=simCIBMTR, model="semi-Markov")
R> summary(fitFreqPHR)
```

```
Analysis of independent semi-competing risks data
semi-Markov assumption for h3
Confidence level: 0.05

Hazard ratios:
         beta1   LL  UL beta2   LL  UL beta3   LL  UL
dTypeALL  1.49 1.20 1.8  1.37 1.09 1.7  0.99 0.78 1.3
dTypeCML  1.78 1.41 2.3  0.83 0.64 1.1  1.30 0.99 1.7
dTypeMDS  1.64 1.26 2.1  1.39 1.04 1.9  1.49 1.09 2.0
sexP      0.89 0.79 1.0    NA   NA  NA    NA   NA  NA

Variance of frailties:
      Estimate  LL  UL
theta      7.8 7.3 8.4

Baseline hazard function components:
                   h1-PM   LL    UL  h2-PM     LL     UL  h3-PM     LL     UL
Weibull: log-kappa -6.14 -6.4 -5.90 -11.33 -11.74 -10.93 -6.873 -7.189 -6.557
Weibull: log-alpha  0.15  0.1  0.21   0.86   0.82   0.91  0.022 -0.033  0.077
```

As shown in Section Summary output, `summary` provides estimates of all model parameters. Using the auxiliary functions `predict` (default option `x1new=x2new=x3new=NULL` which corresponds to the baseline specification) and `plot`, we can graphically visualize the results:

```
R> pred <- predict(fitFreqPHR, time=seq(0,365,1), tseq=seq(from=0,to=365,by=30))
R> plot(pred, plot.est="Surv")
R> plot(pred, plot.est="Haz")
```

Figure 2 displays estimated baseline survival and hazard functions (solid line) with their corresponding 95% confidence intervals (dotted line).

Bayesian analysis

Independent semi-Markov PHR model with PEM baseline hazards

Our second example is also based on the models (B.4.4)-(B.4.6) adopting a semi-Markov assumption for h_3, but now we use the non-parametric PEM specification for baseline hazard functions. Again, `Formula` is defined as in Section Summary of functionality. Here we employ the Bayesian estimation by means of the function `BayesID_HReg`, described in Section BayesID_HReg. The first step is to specify initial values for model parameters through the `startValues` argument using the auxiliary function `initiate.startValues_HReg`:

```
R> startValues <- initiate.startValues_HReg(form, data=simCIBMTR,
+     model=c("semi-Markov","PEM"), nChain=3)
```

The `nChain` argument indicates the number of Markov chains that will be used in the MCMC algorithm. Next step is to define all model hyperparameters using the `hyperParams` argument:

```
R> hyperParams <- list(theta=c(0.5,0.05), PEM=list(PEM.ab1=c(0.5,0.05),
+     PEM.ab2=c(0.5,0.05), PEM.ab3=c(0.5,0.05), PEM.alpha1=10,
+     PEM.alpha2=10, PEM.alpha3=10))
```

To recall what prior distributions are related to these hyperparameters, see Section PHR models for cluster-correlated semi-competing risks data. Now we set the MCMC configuration for the `mcmcParams` argument, more specifically defining the overall run, storage, and tuning parameters for specific updates:

```
R> sg_max <- c(max(simCIBMTR$time1[simCIBMTR$event1==1]),
+     max(simCIBMTR$time2[simCIBMTR$event1==0 & simCIBMTR$event2==1]),
+     max(simCIBMTR$time2[simCIBMTR$event1==1 & simCIBMTR$event2==1]))

R> mcmcParams <- list(run=list(numReps=5e6, thin=1e3, burninPerc=0.5),
+     storage=list(nGam_save=0, storeV=rep(FALSE,3)),
+     tuning=list(mhProp_theta_var=0.05, Cg=rep(0.2,3), delPertg=rep(0.5,3),
+     rj.scheme=1, Kg_max=rep(50,3), sg_max=sg_max, time_lambda1=seq(1,sg_max[1],1),
+     time_lambda2=seq(1,sg_max[2],1), time_lambda3=seq(1,sg_max[3],1)))
```

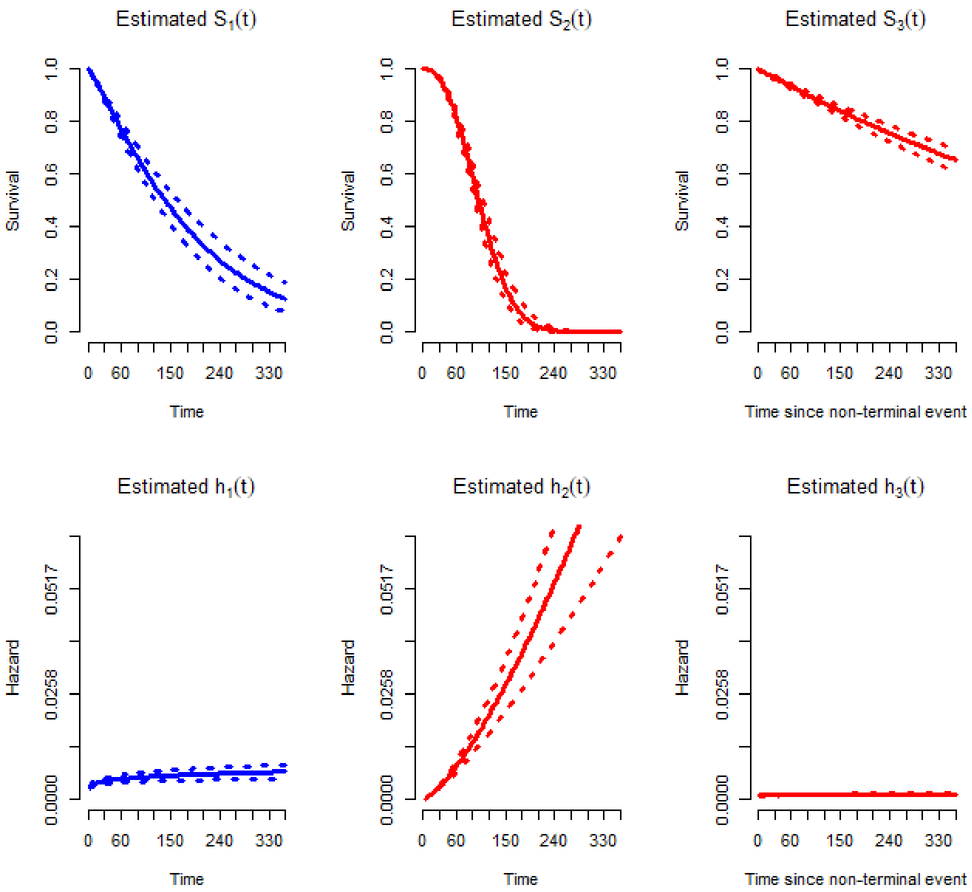


Figure 2: Estimated baseline survival (top) and hazard (bottom) functions from the above analysis.

As shown above, we set `sg_max` to the largest observed failure times for $g \in \{1, 2, 3\}$. For more details of each item of `mcmcParams`, see Section BayesID_HReg.

Given this setup, we fit the PHR model using the function `BayesID_HReg`:

```
R> fitBayesPHR <- BayesID_HReg(form, data=simCIBMTR, model=c("semi-Markov","PEM"),
+     startValues=startValues, hyperParams=hyperParams, mcmcParams=mcmcParams)
```

We note that, depending on the complexity of the model specification (e.g. if PEM baseline hazards are adopted) and the size of the dataset, despite the functions having been written in C and compiled for R, the MCMC scheme may require a large number of MCMC scans to ensure convergence. As such, some models may take a relatively long time to converge. The example we present below, for example, took 45 hours on a Windows laptop with an Intel(R) Core(TM) i5-3337U 1.80GHz processor, 2 cores, 4 logical processors, 4GB of RAM and 3MB of cache memory to cycle through the 6 millions scans for 3 chains. In lieu of attempting to reproduce the exact results we present here, while readers are of course free to do, Appendix Code for illustrative Bayesian examples provides the code for this same semi-competing risks model and its respective posterior summary, but based on a reduced number of scans of the MCMC scheme (specifically 50,000 scans for 3 chains). Based on the full set of scans, the `print` method for object returned by `BayesID_HReg`, yields:

```
R> print(fitBayesPHR, digits=2)

Analysis of independent semi-competing risks data
semi-Markov assumption for h3

Number of chains:     3
Number of scans:      5e+06
Thinning:             1000
Percentage of burnin: 50%
```

```
######
Potential Scale Reduction Factor

Variance of frailties, theta:
 1

Regression coefficients:
         beta1 beta2 beta3
dTypeALL     1     1     1
dTypeCML     1     1     1
dTypeMDS     1     1     1
sexP         1    NA    NA

Baseline hazard function components:

lambda1: summary statistics
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.01    1.01    1.01    1.02    1.02

lambda2: summary statistics
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.00    1.00    1.00    1.00    1.02

lambda3: summary statistics
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.00    1.00    1.00    1.00    1.01

        h1 h2 h3
mu       1  1  1
sigmaSq  1  1  1
K        1  1  1

...
```

Note that all parameters obtained PSRF close to 1, indicating that the chains have converged well (see Section Summary output). Convergence can also be assessed graphically through a trace plot:

```
R> plot(fitBayesPHR$chain1$theta.p, type="l", col="red",
+     xlab="iteration", ylab=expression(theta))
R> lines(fitBayesPHR$chain2$theta.p, type="l", col="green")
R> lines(fitBayesPHR$chain3$theta.p, type="l", col="blue")
```

Figure 3 shows convergence diagnostic for θ (subject-specific frailty variance component), where the three chains have mixed and converged to a stable distribution. Any other model parameter could be similarly evaluated. Analogous to the frequentist example, we can also visualize the results through the function `summary`:

```
R> summary(fitBayesPHR)

Analysis of independent semi-competing risks data
semi-Markov assumption for h3

#####

DIC:  85722
LPML:  -42827
Credibility level: 0.05

#####

Hazard ratios:
         exp(beta1)  LL  UL exp(beta2)   LL  UL exp(beta3)   LL  UL
dTypeALL       1.44 1.2 1.8        1.3 1.06 1.6       0.98 0.77 1.2
```

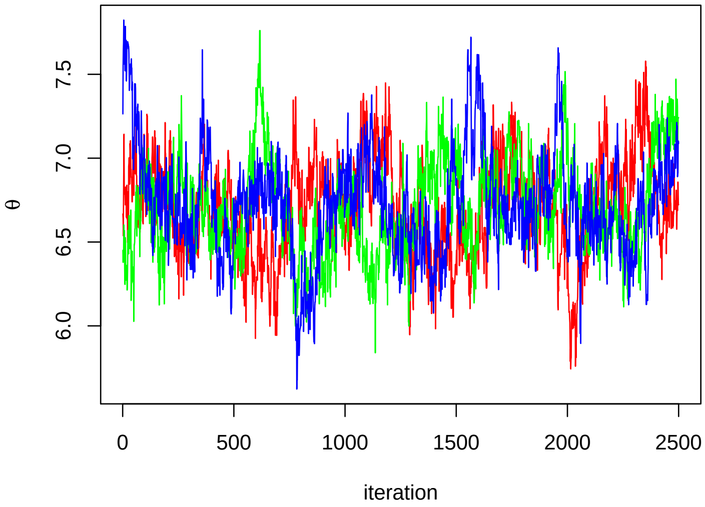


Figure 3: Convergence diagnostic via trace plot of multiple chains.

```
dTypeCML        1.71 1.4 2.1        0.8 0.63 1.0        1.25 0.96 1.6
dTypeMDS        1.61 1.3 2.1        1.4 1.04 1.8        1.44 1.07 2.0
sexP            0.89 0.8 1.0         NA   NA  NA          NA   NA  NA

Variance of frailties:
 theta  LL  UL
   6.7 6.1 7.4

Baseline hazard function components:
        h1-PM     LL   UL h2-PM   LL   UL h3-PM     LL   UL
mu      -5.60 -6.006 -5.0  -5.0 -9.5 -2.3 -6.74 -7.030 -6.5
sigmaSq  0.22  0.027  2.3   7.6  2.7 24.5  0.13  0.018  2.7
K       10.00  5.000 17.0  15.0 11.0 20.0 10.00  4.000 17.0
```

Here we provide two model assessment measures (DIC and LPML) and estimates of all model parameters with their respective 95% credible intervals.

Independent AFT model with log-Normal baseline survival distribution

Our last example is based on AFT models (B.4.1)-(B.4.3) adopting a semi-Markov assumption for h_3 and the parametric log-Normal specification for baseline survival distributions. Here we apply the Bayesian framework via function BayesID_AFT. As pointed out in Section BayesID_AFT, Formula argument for AFT models takes a specific form:

```
R> simCIBMTR$LT <- rep(0,dim(simCIBMTR)[1])
R> simCIBMTR$y1L <- simCIBMTR$y1U <- simCIBMTR[,1]
R> simCIBMTR$y1U[which(simCIBMTR[,2]==0)] <- Inf
R> simCIBMTR$y2L <- simCIBMTR$y2U <- simCIBMTR[,3]
R> simCIBMTR$y2U[which(simCIBMTR[,4]==0)] <- Inf

R> formAFT <- Formula(LT | y1L + y1U | y2L + y2U ~ dTypeALL + dTypeCML + dTypeMDS +
+     sexP | dTypeALL + dTypeCML + dTypeMDS | dTypeALL + dTypeCML + dTypeMDS)
```

Recall that LT represents the left-truncation time, and (y1L, y1U) and (y2L, y2U) are the interval-censored times to the non-terminal and terminal events, respectively. Next step is to set the initial values for model parameters through the startValues argument, but now using the auxiliary function initiate.startValues_AFT:

```
R> startValues <- initiate.startValues_AFT(formAFT, data=simCIBMTR,
+     model="LN", nChain=3)
```

Again, we considered three Markov chains (nChain=3). Using the hyperParams argument we specify all model hyperparameters:

```
R> hyperParams <- list(theta=c(0.5,0.05), LN=list(LN.ab1=c(0.5,0.05),
+     LN.ab2=c(0.5,0.05), LN.ab3=c(0.5,0.05)))
```

Each pair of hyperparameters defines shape and scale of an inverse Gamma prior distribution (see Section AFT models for independent semi-competing risks data). Similar to the previous example, we must specify overall run, storage, and tuning parameters for specific updates through the mcmcParams argument:

```
R> mcmcParams <- list(run=list(numReps=5e6, thin=1e3, burninPerc=0.5),
+     storage=list(nGam_save=0, nY1_save=0, nY2_save=0, nY1.NA_save=0),
+     tuning=list(betag.prop.var=rep(0.01,3), mug.prop.var=rep(0.01,3),
+     zetag.prop.var=rep(0.01,3), gamma.prop.var=0.01))
```

Analogous to the previous Bayesian model, a large number of scans are also required here to achieve the convergence of the Markov chains. Again, for a quickly reproducible example, the code for the AFT model with simplified MCMC setting is provided in Appendix Code for illustrative Bayesian examples. For more details of each item of mcmcParams, see Section BayesID_AFT. Finally, we fit the AFT model using the function BayesID_AFT and analyze the convergence of each parameter through the function print:

```
R> fitBayesAFT <- BayesID_AFT(formAFT, data=simCIBMTR, model="LN",
+     startValues=startValues, hyperParams=hyperParams, mcmcParams=mcmcParams)
R> print(fitBayesAFT, digits=2)

Analysis of independent semi-competing risks data

Number of chains:     3
Number of scans:      5e+06
Thinning:             1000
Percentage of burnin: 50%

######
Potential Scale Reduction Factor

Variance of frailties, theta:  1

Regression coefficients:
         beta1 beta2 beta3
dTypeALL     1     1     1
dTypeCML     1     1     1
dTypeMDS     1     1     1
sexP         1    NA    NA

Baseline survival function components:
        g=1 g=2 g=3
mu        1 1.2   1
sigmaSq   1 1.1   1

...
```

Again, the PSRF for each parameter indicates the convergence. As a last step, we visualize the estimate of each parameter and their respective 95% credible intervals through the function summary:

```
R> summary(fitBayesAFT)

Analysis of independent semi-competing risks data

#####

DIC:  21400
LPML:  -12597
Credibility level: 0.05
```

```
#####

Acceleration factors:
         exp(beta1)   LL   UL exp(beta2)   LL  UL exp(beta3)   LL  UL
dTypeALL       0.68 0.54 0.84       0.94 0.86 1.0       1.08 0.85 1.4
dTypeCML       0.53 0.42 0.67       1.27 1.12 1.4       0.92 0.71 1.2
dTypeMDS       0.58 0.44 0.75       0.88 0.78 1.0       0.78 0.58 1.0
sexP           1.16 0.99 1.36         NA   NA  NA         NA   NA  NA

Variance of frailties:
 theta  LL  UL
   2.6 2.5 2.8

Baseline survival function components:
                    g=1: PM  LL  UL g=2: PM    LL    UL g=3: PM  LL  UL
log-Normal: mu          8.2 8.0 8.4   6.293 6.244 6.335     6.5 6.4 6.7
log-Normal: sigmaSq     7.2 6.4 8.0   0.013 0.005 0.033     1.7 1.5 2.0
```

Discussion

This paper discusses the implementation of a comprehensive R package **SemiCompRisks** for the analyses of independent/cluster-correlated semi-competing risks data. The package allows to fit parametric or semi-parametric models based on either accelerated failure time or proportional hazards regression approach. It is also flexible in that one can adopt either a Markov or semi-Markov specification for terminal event following non-terminal event. The estimation and inference are mostly based on the Bayesian paradigm, but parametric PHR models can also be fitted using the maximum likelihood estimation. Users can easily obtain numerical and graphical presentation of model fits using R methods, as illustrated in the stem cell transplantation example in Section Illustration: Stem cell transplantation data. In addition, the package provides functions for performing univariate survival analysis. We would also like to emphasize that the vignette documentation (Lee et al., 2017b) provides a list of detailed examples applying each of the implemented models in the package.

Given the complexity of some Bayesian models in the package, it may take relatively long time to implement the models for large datasets. We are currently looking into possibility to parallelize parts of the algorithm and to add support for OpenMP to the package, which can bring significant gains in computational time.

SemiCompRisks provides researchers with valid and practical analysis tools for semi-competing risks data. The application examples in this paper were run using version v3.30 of the package, available from the CRAN at `https://cran.r-project.org/package=SemiCompRisks`. We plan to constantly update the package to incorporate more functionality and flexibility to the models for semi-competing risks analysis.

Acknowledgments

Funding for this work was provided by National Institutes of Health grants R01 CA181360-01. The authors also gratefully acknowledge the CIBMTR (grant U24-CA076518) for providing the covariates of the illustrative example.

Bibliography

A. Allignol, J. Beyersmann, and M. Schumacher. Mvna: An R package for the Nelson-Aalen estimator in multistate models. *R News*, 8(2):48–50, 2008. URL `http://cran.r-project.org/doc/Rnews/Rnews_2008-2.pdf`. [p377]

A. Allignol, M. Schumacher, and J. Beyersmann. Empirical transition matrix of multi-state models: The etm package. *Journal of Statistical Software*, 38(4):1–15, 2011. URL `https://doi.org/10.18637/jss.v038.i04`. [p377]

A. Araújo, L. Meira-Machado, and J. Roca-Pardiñas. TPmsm: Estimation of the transition probabilities in 3-state models. *Journal of Statistical Software*, 62(4):1–29, 2014. URL `https://doi.org/10.18637/jss.v062.i04`. [p377]

A. Boruvka and R. J. Cook. *Coxinterval: Cox-Type Models for Interval-Censored Data*, 2015. URL https://cran.r-project.org/package=coxinterval. R package version 1.2. [p378]

S. P. Brooks and A. Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, 1998. URL https://doi.org/10.1080/10618600.1998.10474787. [p384]

G. Celeux, F. Forbes, C. P. Robert, and D. M. Titterington. Deviance information criteria for missing data models. *Bayesian Analysis*, 1(4):651–673, 2006. URL https://doi.org/10.1214/06-BA122. [p384]

B. L. Egleston, D. O. Scharfstein, E. E. Freeman, and S. K. West. Causal inference for non-mortality outcomes in the presence of death. *Biostatistics*, 8(3):526–545, 2007. URL https://doi.org/10.1093/biostatistics/kxl027. [p376]

N. Ferguson, S. Datta, and G. Brock. msSurv: An R package for nonparametric estimation of multistate models. *Journal of Statistical Software*, 50(14):1–24, 2012. URL https://doi.org/10.18637/jss.v050.i14. [p378]

J. P. Fine, H. Jiang, and R. Chappell. On semi-competing risks data. *Biometrika*, 88(4):907–919, 2001. URL https://doi.org/10.1093/biomet/88.4.907. [p376]

H. Fu, Y. Wang, J. Liu, P. M. Kulkarni, and A. S. Melemed. Joint modeling of progression-free survival and overall survival by a Bayesian normal induced copula estimation model. *Statistics in Medicine*, 32(2):240–254, 2013. URL https://doi.org/10.1002/sim.5487. [p376]

S. Geisser and W. F. Eddy. A predictive approach to model selection. *Journal of the American Statistical Association*, 74(365):153–160, 1979. URL https://doi.org/10.2307/2286745. [p384]

A. E. Gelfand and B. K. Mallick. Bayesian analysis of proportional hazards models built from monotone functions. *Biometrics*, 51(3):843–852, 1995. URL https://doi.org/10.2307/2532986. [p384]

A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992. URL https://doi.org/10.1214/ss/1177011136. [p384]

D. Ghosh. Semiparametric inferences for association with semi-competing risks data. *Statistics in Medicine*, 25(12):2059–2070, 2006. URL https://doi.org/10.1002/sim.2327. [p376]

B. Han, M. Yu, J. J. Dignam, and P. J. Rathouz. Bayesian approach for flexible modeling of semicompeting risks data. *Statistics in Medicine*, 33(29):5111–5125, 2014. URL https://doi.org/10.1002/sim.6313. [p376]

S. Haneuse and K. H. Lee. Semi-competing risks data analysis: Accounting for death as a competing risk when the outcome of interest is nonterminal. *Circulation: Cardiovascular Quality and Outcomes*, 9(3):322–331, 2016. URL https://doi.org/10.1161/CIRCOUTCOMES.115.001841. [p376]

J. J. Hsieh, W. Wang, and A. A. Ding. Regression analysis based on semicompeting risks data. *Journal of the Royal Statistical Society B*, 70(1):3–20, 2008. URL https://doi.org/10.1111/j.1467-9868.2007.00621.x. [p376]

C. H. Jackson. Multi-state models for panel data: The msm package for R. *Journal of Statistical Software*, 38(8):1–28, 2011. URL https://doi.org/10.18637/jss.v038.i08. [p377]

C. H. Jackson. Flexsurv: A platform for parametric survival modeling in R. *Journal of Statistical Software*, 70(8):1–33, 2016. URL https://doi.org/10.18637/jss.v070.i08. [p377]

I. Jazić, D. Schrag, D. J. Sargent, and S. Haneuse. Beyond composite endpoints analysis: Semicompeting risks as an underutilized framework for cancer research. *Journal of the National Cancer Institute*, 108(12):djw154, 2016. URL https://doi.org/10.1093/jnci/djw154. [p376]

H. Jiang, J. P. Fine, and R. Chappell. Semiparametric analysis of survival data with left truncation and dependent right censoring. *Biometrics*, 61(2):567–575, 2005. URL https://doi.org/10.1111/j.1541-0420.2005.00335.x. [p376]

J. P. Klein and M. L. Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data*. Springer-Verlag, 2nd edition, 2003. [p384]

T. Kneib and A. Hennerfeind. Bayesian semi parametric multi-state models. *Statistical Modelling*, 8 (2):169–198, 2008. URL https://doi.org/10.1177/1471082X0800800203. [p376]

A. Król and P. Saint-Pierre. SemiMarkov: An R package for parametric estimation in multi-state semi-Markov models. *Journal of Statistical Software*, 66(6):1–16, 2015. URL https://doi.org/10.18637/jss.v066.i06. [p377]

L. Lakhal, L. P. Rivest, and B. Abdous. Estimating survival and association in a semicompeting risks model. *Biometrics*, 64(1):180–188, 2008. URL https://doi.org/10.1111/j.1541-0420.2007.00872.x. [p376]

C. Lee, S. J. Lee, and S. Haneuse. Time-to-event analysis when the event is defined on a finite time interval. *Submitted*, 2017a. [p378]

K. H. Lee, S. Haneuse, D. Schrag, and F. Dominici. Bayesian semiparametric analysis of semicompeting risks data: Investigating hospital readmission after a pancreatic cancer diagnosis. *Journal of the Royal Statistical Society C*, 64(2):253–273, 2015. URL https://doi.org/10.1111/rssc.12078. [p376, 377, 379, 380, 381, 384]

K. H. Lee, F. Dominici, D. Schrag, and S. Haneuse. Hierarchical models for semicompeting risks data with application to quality of end-of-life care for pancreatic cancer. *Journal of the American Statistical Association*, 111(515):1075–1095, 2016. URL https://doi.org/10.1080/01621459.2016.1164052. [p376, 377, 378, 380, 381]

K. H. Lee, C. Lee, D. Alvares, and S. Haneuse. *SemiCompRisks: Hierarchical Models for Parametric and Semi-Parametric Analyses of Semi-Competing Risks Data*, 2017b. URL https://cran.r-project.org/web/packages/SemiCompRisks/vignettes/SemiCompRisks.pdf. R package version 3.30. [p378, 381, 382, 383, 384, 390]

K. H. Lee, V. Rondeau, and S. Haneuse. Accelerated failure time models for semi-competing risks data in the presence of complex censoring. *Biometrics*, 73(4):1401–1412, 2017c. URL https://doi.org/10.1111/biom.12696. [p377, 379, 381]

B. Liquet, J. F. Timsit, and V. Rondeau. Investigating hospital heterogeneity with a multi-state frailty model: Application to nosocomial pneumonia disease in intensive care units. *BMC Medical Research Methodology*, 12(1):1–14, 2012. URL https://doi.org/10.1186/1471-2288-12-79. [p378, 395]

L. Liu, R. A. Wolfe, and X. Huang. Shared frailty models for recurrent events and a terminal event. *Biometrics*, 60(3):747–756, 2004. URL https://doi.org/10.1111/j.0006-341X.2004.00225.x. [p376]

L. Meira-Machado and J. Roca-Pardiñas. P3state.msm: Analyzing survival data from an illness-death model. *Journal of Statistical Software*, 38(3):1–18, 2011. URL https://doi.org/10.18637/jss.v038.i03. [p377]

L. Meira-Machado, C. Cadarso-Suárez, and J. Uña-Álvarez. Tdc.msm: AnR library for the analysis of multi-state survival data. *Computer Methods and Programs in Biomedicine*, 86(2):131–140, 2007. URL https://doi.org/10.1016/j.cmpb.2007.01.010. [p377]

R. M. Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000. URL https://doi.org/10.1080/10618600.2000.10474879. [p379]

L. Peng and J. P. Fine. Regression modeling of semicompeting risks data. *Biometrics*, 63(1):96–108, 2007. URL https://doi.org/10.1111/j.1541-0420.2006.00621.x. [p376]

H. Putter, M. Fiocco, and R. B. Geskus. Tutorial in biostatistics: Competing risks and multi-state models. *Statistics in Medicine*, 26(11):2389–2430, 2007. URL https://doi.org/10.1002/sim.2712. [p376]

V. Rondeau, Y. Mazroui, and J. R. Gonzalez. Frailtypack: An R package for the analysis of correlated survival data with frailty models using penalized likelihood estimation or parametrical estimation. *Journal of Statistical Software*, 47(4):1–28, 2012. URL https://doi.org/10.18637/jss.v047.i04. [p378]

D. J. Spiegelhalter, N. G. Best, B. P. Carlin, and A. van der Linde. Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society B*, 64(4):583–639, 2002. URL https://doi.org/10.1111/1467-9868.00353. [p384]

E. J. Tchetgen Tchetgen. Identification and estimation of survivor average causal effects. *Statistics in Medicine*, 33(21):3601–3628, 2014. URL https://doi.org/10.1002/sim.6181. [p376]

C. Touraine, T. A. Gerds, and P. Joly. SmoothHazard: An R package for fitting regression models to interval-censored observations of illness-death models. *Journal of Statistical Software*, 79(7): 1–22, 2017. URL https://doi.org/10.18637/jss.v079.i07. [p378]

R. Varadhan, Q. L. Xue, and K. Bandeen-Roche. Semicompeting risks in aging research: Methods, issues and needs. *Lifetime Data Analysis*, 20(4):538–562, 2014. URL https://doi.org/10.1007/s10985-014-9295-7. [p376]

W. Wang. Estimating the association parameter for copula models under dependent censoring. *Journal of the Royal Statistical Society B*, 65(1):257–273, 2003. URL https://doi.org/10.1111/1467-9868.00385. [p376]

L. J. Wei. The accelerated failure time model: A useful alternative to the Cox regression model in survival analysis. *Statistics in Medicine*, 11(14-15):1871–1879, 1992. URL https://doi.org/10.1002/sim.4780111409. [p379]

L. C. Wreede, M. Fiocco, and H. Putter. Mstate: An R package for the analysis of competing risks and multi-state models. *Journal of Statistical Software*, 38(7):1–30, 2011. URL https://doi.org/10.18637/jss.v038.i07. [p378]

J. Xu, J. D. Kalbfleisch, and B. Tai. Statistical analysis of illness-death processes and semicompeting risks data. *Biometrics*, 66(3):716–725, 2010. URL https://doi.org/10.1111/j.1541-0420.2009.01340.x. [p376, 379]

Y. Ye, J. D. Kalbfleisch, and D. E. Schaubel. Semiparametric analysis of correlated recurrent and terminal events. *Biometrics*, 63(1):78–87, 2007. URL https://doi.org/10.1111/j.1541-0420.2006.00677.x. [p376]

D. Zeng and D. Y. Lin. Semiparametric transformation models with random effects for joint analysis of recurrent and terminal events. *Biometrics*, 65(3):746–752, 2009. URL https://doi.org/10.1111/j.1541-0420.2008.01126.x. [p376]

D. Zeng, Q. Chen, M. H. Chen, and J. G. Ibrahim. Estimating treatment effects with treatment switching via semicompeting risks models: An application to a colorectal cancer study. *Biometrika*, 99(1):167–184, 2012. URL https://doi.org/10.1093/biomet/asr062. [p376]

J. L. Zhang and D. B. Rubin. Estimation of causal effects via principal stratification when some outcomes are truncated by “death”. *Journal of Educational and Behavioral Statistics*, 28(4): 353–368, 2003. URL https://doi.org/10.3102/10769986028004353. [p376]

Y. Zhang, M. H. Chen, J. G. Ibrahim, D. Zeng, Q. Chen, Z. Pan, and X. Xue. Bayesian gamma frailty models for survival data with semi-competing risks and treatment switching. *Lifetime Data Analysis*, 20(1):76–105, 2014. URL https://doi.org/10.1007/s10985-013-9254-8. [p376]

Danilo Alvares
Department of Statistics
Pontificia Universidad Católica de Chile
Macul, Santiago, Chile
dalvares@mat.uc.cl

Sebastien Haneuse
Department of Biostatistics
Harvard T. H. Chan School of Public Health
02115 Boston, MA, USA
shaneuse@hsph.harvard.edu

Catherine Lee
Division of Research
Kaiser Permanente Northern California
94612 Oakland, CA, USA
catherine.lee@kp.org

Kyu Ha Lee
Epidemiology and Biostatistics Core

The Forsyth Institute
02142 Cambridge, MA, USA
klee@forsyth.org

Appendix

Simulation algorithm for semi-competing risks data

The **SemiCompRisks** package contains a function, `simID`, for simulating independent or cluster-correlated semi-competing risks data. In this section, we provide the details on the simulation algorithm used in `simID` for generating cluster-correlated semi-competing risks data based on a parametric Weibull-MVN semi-Markov illness-death model, as presented in Section PHR models for cluster-correlated semi-competing risks data, where the baseline hazard functions are defined as $h_{0g}(t) = \alpha_g \, \kappa_g \, t^{\alpha_g - 1}$, for $g \in \{1, 2, 3\}$. The step by step algorithm is given as follows:

1. Generate $\boldsymbol{V}_j = (V_{j1}, V_{j2}, V_{j3})^\top$ from a MVN($\mathbf{0}$, Σ_V), for $j = 1, \ldots, J$.
2. For each j, repeat the following steps for $i = 1, \ldots, n_j$.
    (a) Generate γ_{ji} from a Gamma(θ^{-1}, θ^{-1}).
    (b) Calculate $\eta_{jig} = \log(\gamma_{ji}) + \boldsymbol{x}_{jig}^\top \boldsymbol{\beta}_g + V_{jg}$, for $g \in \{1, 2, 3\}$.
    (c) Generate t_1^* from a Weibull(α_1, $\kappa_1 \, e^{\eta_{ji1}}$) and t_2^* from a Weibull(α_2, $\kappa_2 \, e^{\eta_{ji2}}$).
        - If $t_1^* \leqslant t_2^*$, generate t^* from a Weibull(α_3, $\kappa_3 \, e^{\eta_{ji3}}$) and set $t_{ji1} = t_1^*$, $t_{ji2} = t_1^* + t^*$.
        - Otherwise, set $t_{ji1} = \infty$, $t_{ji2} = t_2^*$.
    (d) Generate a censoring time c_{ji} from Uniform(c_L, c_U).
    (e) Set the observed outcome information (`time1`, `time2`, `event1`, `event2`) as follows:
        - (t_{ji1}, t_{ji2}, 1, 1), if $t_{ji1} < t_{ji2} < c_{ji}$.
        - (t_{ji1}, c_{ji}, 1, 0), if $t_{ji1} < c_{ji} < t_{ji2}$.
        - (t_{ji2}, t_{ji2}, 0, 1), if $t_{ji1} = \infty$ and $t_{ji2} < c_{ji}$.
        - (c_{ji}, c_{ji}, 0, 0), if $t_{ji1} > c_{ji}$ and $t_{ji2} > c_{ji}$.

We note that the function `simID` is flexible in that one can set the θ argument as zero (`theta.true`=0) to simulate the data under the model without the subject-specific shared frailty term (γ_{ji}), which is analogous to the model proposed by Liquet et al. (2012). One can generate independent semi-competing risks data outlined in Section PHR models for independent semi-competing risks data by setting the `id` and Σ_V arguments as nulls (`id`=`NULL` and `SimgaV.true`=`NULL`).

Simulating outcomes using CIBMTR covariates

The true values of model parameters are set to estimates obtained by fitting a semi-Markov Weibull PHR model to the original CIBMTR data.

```
R> data(CIBMTR_Params)
R> beta1.true <- CIBMTR_Params$beta1.true
R> beta2.true <- CIBMTR_Params$beta2.true
R> beta3.true <- CIBMTR_Params$beta3.true
R> alpha1.true <- CIBMTR_Params$alpha1.true
R> alpha2.true <- CIBMTR_Params$alpha2.true
R> alpha3.true <- CIBMTR_Params$alpha3.true
R> kappa1.true <- CIBMTR_Params$kappa1.true
R> kappa2.true <- CIBMTR_Params$kappa2.true
R> kappa3.true <- CIBMTR_Params$kappa3.true
R> theta.true <- CIBMTR_Params$theta.true
R> cens <- c(365, 365)
```

The next step is to define the covariates matrices and then simulate outcomes using the `simID` function, available in the **SemiCompRisks** package.

```
R> data(CIBMTR)
# Sex (M: reference category)
R> CIBMTR$sexP <- as.numeric(CIBMTR$sexP)-1

# Age (LessThan10: reference category)
R> CIBMTR$ageP20to29 <- as.numeric(CIBMTR$ageP=="20to29")
R> CIBMTR$ageP30to39 <- as.numeric(CIBMTR$ageP=="30to39")
R> CIBMTR$ageP40to49 <- as.numeric(CIBMTR$ageP=="40to49")
```

```
R> CIBMTR$ageP50to59 <- as.numeric(CIBMTR$ageP=="50to59")
R> CIBMTR$ageP60plus <- as.numeric(CIBMTR$ageP=="60plus")

# Disease type (AML: reference category)
R> CIBMTR$dTypeALL <- as.numeric(CIBMTR$dType=="ALL")
R> CIBMTR$dTypeCML <- as.numeric(CIBMTR$dType=="CML")
R> CIBMTR$dTypeMDS <- as.numeric(CIBMTR$dType=="MDS")

# Disease status (Early: reference category)
R> CIBMTR$dStatusInt <- as.numeric(CIBMTR$dStatus=="Int")
R> CIBMTR$dStatusAdv <- as.numeric(CIBMTR$dStatus=="Adv")

# HLA compatibility (HLA_Id_Sib: reference category)
R> CIBMTR$donorGrp8_8 <- as.numeric(CIBMTR$donorGrp=="8_8")
R> CIBMTR$donorGrp7_8 <- as.numeric(CIBMTR$donorGrp=="7_8")

# Covariate matrix
R> x1 <- CIBMTR[,c("sexP", "ageP20to29", "ageP30to39", "ageP40to49",
+     "ageP50to59", "ageP60plus", "dTypeALL", "dTypeCML", "dTypeMDS",
+     "dStatusInt", "dStatusAdv", "donorGrp8_8", "donorGrp7_8")]

R> x2 <- CIBMTR[,c("sexP", "ageP20to29", "ageP30to39", "ageP40to49",
+     "ageP50to59", "ageP60plus", "dTypeALL", "dTypeCML", "dTypeMDS",
+     "dStatusInt", "dStatusAdv", "donorGrp8_8", "donorGrp7_8")]

R> x3 <- CIBMTR[,c("sexP", "ageP20to29", "ageP30to39", "ageP40to49",
+     "ageP50to59", "ageP60plus", "dTypeALL", "dTypeCML", "dTypeMDS",
+     "dStatusInt", "dStatusAdv", "donorGrp8_8", "donorGrp7_8")]

R> set.seed(1405)
R> simOutcomes <- simID(id=NULL, x1=x1, x2=x2, x3=x3,
+     beta1.true, beta2.true, beta3.true, alpha1.true, alpha2.true, alpha3.true,
+     kappa1.true, kappa2.true, kappa3.true, theta.true, SigmaV.true=NULL, cens)

R> names(simOutcomes) <- c("time1", "event1", "time2", "event2")
R> simCIBMTR <- cbind(simOutcomes, CIBMTR[,c("sexP", "ageP20to29", "ageP30to39",
+    "ageP40to49", "ageP50to59", "ageP60plus", "dTypeALL", "dTypeCML", "dTypeMDS",
+    "dStatusInt", "dStatusAdv", "donorGrp8_8", "donorGrp7_8")])
```

Code for illustrative Bayesian examples

In order to encourage the reproducibility of the results obtained through our R package in a reasonable computational time, Bayesian analyses contained in Section Bayesian analysis are illustrated below using a reduced number of scans (**numReps**), extent of thinning (**thin**), and simplifying the design matrix. Given the complexity of these Bayesian models, the reduction of scans/thinning results in non-convergence of the Markov chains, but at least it is possible to reproduce the results quickly.

Independent semi-Markov PHR model with PEM baseline hazards

```
R> form <- Formula(time1 + event1 | time2 + event2 ~ sexP | sexP | sexP)

R> startValues <- initiate.startValues_HReg(form, data=simCIBMTR,
+     model=c("semi-Markov","PEM"), nChain=3)

R> hyperParams <- list(theta=c(0.5,0.05), PEM=list(PEM.ab1=c(0.5,0.05),
+     PEM.ab2=c(0.5,0.05), PEM.ab3=c(0.5,0.05), PEM.alpha1=10,
+     PEM.alpha2=10, PEM.alpha3=10))

R> sg_max <- c(max(simCIBMTR$time1[simCIBMTR$event1==1]),
+     max(simCIBMTR$time2[simCIBMTR$event1==0 & simCIBMTR$event2==1]),
+     max(simCIBMTR$time2[simCIBMTR$event1==1 & simCIBMTR$event2==1]))

R> mcmcParams <- list(run=list(numReps=5e4, thin=5e1, burninPerc=0.5),
+     storage=list(nGam_save=0, storeV=rep(FALSE,3)),
+     tuning=list(mhProp_theta_var=0.05, Cg=rep(0.2,3), delPertg=rep(0.5,3),
+     rj.scheme=1, Kg_max=rep(50,3), sg_max=sg_max, time_lambda1=seq(1,sg_max[1],1),
+     time_lambda2=seq(1,sg_max[2],1), time_lambda3=seq(1,sg_max[3],1)))

R> fitBayesPHR <- BayesID_HReg(form, data=simCIBMTR, model=c("semi-Markov","PEM"),
+     startValues=startValues, hyperParams=hyperParams, mcmcParams=mcmcParams)
R> print(fitBayesPHR, digits=2)

Analysis of independent semi-competing risks data
semi-Markov assumption for h3

Number of chains:     3
Number of scans:      50000
Thinning:             50
Percentage of burnin: 50%

######
Potential Scale Reduction Factor

Variance of frailties, theta:
 5.4

Regression coefficients:
     beta1 beta2 beta3
sexP   1.3   1.4   1.3

Baseline hazard function components:

lambda1: summary statistics
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.1     2.7     3.0     3.0     3.3     4.0

lambda2: summary statistics
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0     2.5     3.6     3.3     4.1     5.2

lambda3: summary statistics
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.12    1.42    1.60    1.59    1.70    2.17

         h1  h2  h3
mu      1.2 1.0 1.1
sigmaSq 1.2 1.1 1.0
K       1.0 1.4 1.0

######
Estimates
Credibility level: 0.05

Variance of frailties, theta:
 Estimate   SD  LL UL
      9.4 0.71 8.9 11

Regression coefficients:
     Estimate   SD   LL   UL
sexP    -0.19 0.09 0.68 0.99
sexP    -0.04 0.10 0.78 1.16
sexP    -0.08 0.11 0.74 1.14

Note: Covariates are arranged in order of transition number, 1->3.
```

The joint posterior predictive probability involving two event times can be obtained with the PPD function:

```
# Prediction for a female patient (x1=x2=x3=1)
R> predF <- PPD(fitBayesPHR, x1=1, x2=1, x3=1, t1=120, t2=300)
R> predF$F_u
0.076
R> predF$F_l
0.26
```

predF$F_u represents the joint posterior predictive probability of dying within 300 days and being diagnosed with acute GVHD within 120 days for a female patient (the joint probability from the upper wedge support, 0<t1<t2). On the other hand, predF$F_l is the joint posterior predictive probability of dying within 300 days without acute GVHD for a female patient (the joint probability from the domain, t1=∞, t2>0).

Independent AFT model with log-Normal baseline survival distribution

```
R> simCIBMTR$LT <- rep(0,dim(simCIBMTR)[1])
R> simCIBMTR$y1L <- simCIBMTR$y1U <- simCIBMTR[,1]
R> simCIBMTR$y1U[which(simCIBMTR[,2]==0)] <- Inf
R> simCIBMTR$y2L <- simCIBMTR$y2U <- simCIBMTR[,3]
R> simCIBMTR$y2U[which(simCIBMTR[,4]==0)] <- Inf

R> formAFT <- Formula(LT | y1L + y1U | y2L + y2U ~ sexP | sexP | sexP)

R> startValues <- initiate.startValues_AFT(formAFT, data=simCIBMTR,
+     model="LN", nChain=3)

R> hyperParams <- list(theta=c(0.5,0.05), LN=list(LN.ab1=c(0.5,0.05),
+     LN.ab2=c(0.5,0.05), LN.ab3=c(0.5,0.05)))

R> mcmcParams <- list(run=list(numReps=5e4, thin=5e1, burninPerc=0.5),
+     storage=list(nGam_save=0, nY1_save=0, nY2_save=0, nY1.NA_save=0),
+     tuning=list(betag.prop.var=rep(0.01,3), mug.prop.var=rep(0.01,3),
+     zetag.prop.var=rep(0.01,3), gamma.prop.var=0.01))

R> fitBayesAFT <- BayesID_AFT(formAFT, data=simCIBMTR, model="LN",
+     startValues=startValues, hyperParams=hyperParams, mcmcParams=mcmcParams)
R> summary(fitBayesAFT, digits=2)

Analysis of independent semi-competing risks data

#####

DIC:  55244
LPML:  -25839
Credibility level: 0.05

#####

Acceleration factors:
     exp(beta1)   LL  UL exp(beta2)   LL   UL exp(beta3)  LL  UL
sexP        1.2 0.95 1.4       0.92 0.86 0.99       0.93 0.8 1.1

Variance of frailties:
 theta   LL  UL
   1.5 0.96 1.8

Baseline survival function components:
                    g=1: PM  LL   UL g=2: PM   LL  UL g=3: PM  LL  UL
log-Normal: mu          8.3 8.2  8.6     6.4 6.38 6.5     6.1 5.9 6.2
log-Normal: sigmaSq    10.1 9.2 11.8     1.1 0.82 1.7     1.9 1.6 2.5
```

			Outcome category (%)			
	N	%	Both acute GVHD & death	Acute GVHD & censored for death	Death without acute GVHD	Censored for both
Total subjects	9,651	100.0	9.5	8.9	28.8	52.8
Gender						
Male	5,366	55.6	9.7	9.5	28.1	52.7
Female	4,285	44.4	9.1	8.3	29.7	52.9
Age, years						
<10	653	6.8	5.0	11.9	23.4	59.7
10-19	1,162	12.0	8.0	11.4	24.0	56.6
20-29	1,572	16.3	9.7	9.9	27.4	53.0
30-39	1,581	16.4	9.8	10.7	28.5	51.0
40-49	2,095	21.7	11.0	9.6	29.7	49.7
50-59	2,008	20.8	9.8	5.1	32.3	52.8
60+	580	6.0	9.9	4.8	33.1	52.2
Disease type						
AML	4,919	51.0	8.2	8.0	30.3	53.5
ALL	2,071	21.5	9.9	9.0	29.3	51.8
CML	1,525	15.8	12.1	11.3	22.2	54.4
MDS	1,136	11.8	11.0	10.0	30.0	49.0
Disease status						
Early	4,873	50.5	8.4	11.0	23.6	57.0
Intermediate	2,316	24.0	9.7	8.5	30.1	51.7
Advanced	2,462	25.5	11.5	5.4	37.7	45.4

Analysis	Model	Data type	L-T and/or I-C	Statistical paradigm
Semi-competing risks	AFT	Independent	No	B
			Yes	B
		Clustered	No	x
			Yes	x
	PHR	Independent	No	B & F
			Yes	x
		Clustered	No	B
			Yes	x
Univariate	AFT	Independent	No	B
			Yes	B
		Clustered	No	x
			Yes	x
	PHR	Independent	No	B & F
			Yes	x
		Clustered	No	B
			Yes	x

L-T: left-truncation; I-C: interval-censoring; B: Bayesian; F: frequentist; x: not available

Table 2: Models implemented in the **SemiCompRisks** package.

RSSampling: A Pioneering Package for Ranked Set Sampling

by Busra Sevinc, Bekir Cetintav, Melek Esemen, and Selma Gurler

Abstract Ranked set sampling (RSS) is an advanced data collection method when the exact measurement of an observation is difficult and/or expensive used in a number of research areas, e.g., environment, bioinformatics, ecology, etc. In this method, random sets are drawn from a population and the units in sets are ranked with a ranking mechanism which is based on a visual inspection or a concomitant variable. Because of the importance of working with a good design and easy analysis, there is a need for a software tool which provides sampling designs and statistical inferences based on RSS and its modifications. This paper introduces an R package as a free and easy-to-use analysis tool for both sampling processes and statistical inferences based on RSS and its modified versions. For researchers, the **RSSampling** package provides a sample with RSS, extreme RSS, median RSS, percentile RSS, balanced groups RSS, double versions of RSS, L-RSS, truncation-based RSS, and robust extreme RSS when the judgment rankings are both perfect and imperfect. Researchers can also use this new package to make parametric inferences for the population mean and the variance where the sample is obtained via classical RSS. Moreover, this package includes applications of the nonparametric methods which are one sample sign test, Mann-Whitney-Wilcoxon test, and Wilcoxon signed-rank test procedures. The package is available as **RSSampling** on CRAN.

Introduction

Data collection is the crucial part in all types of scientific research. Ranked set sampling (RSS) is one of the advanced data collection methods, which provides representative sample data by using the ranking information of the sample units. It was firstly proposed by McIntyre (1952) and the term "ranked set sampling" was introduced in the study of Halls and Dell (1966) about the estimation of forage yields in a pine hardwood forest. Takahasi and Wakimoto (1968) theoretically studied the efficiency of the mean estimator based on RSS which is unbiased for the population mean. They found that its variance is always smaller than the variance of the mean estimator based on simple random sampling (SRS) with the same sample size when the ranking is perfect. Some other results on the efficiency of RSS can be found in Dell and Clutter (1972), David and Levine (1972), and Stokes (1980a). Stokes (1977) studied the use of concomitant variables for ranking of the sample units in the RSS procedure and found that the ranking procedure was allowed to be imperfect. In another study, she constructed the estimator for the population variance in the presence of the ranking error (Stokes, 1980b). For some examples and results on the regression estimation based on RSS, see, Yu and Lam (1997) and Chen (2001). The estimation of a distribution function with various settings of RSS can be found in Stokes and Sager (1988), Kvam and Samaniego (1993), and Chen (2000). Other results on distribution-free test procedures based on RSS can be found in Bohn and Wolfe (1992, 1994), and Hettmansperger (1995). Additional results for inferential procedures based on RSS can be found in the recent works of Zamanzade and Vock (2015), Zhang et al. (2016), and Ozturk (2018). For more details on RSS, we refer the review papers by Kaur et al. (1995), Chen et al. (2003), and Wolfe (2012).

The RSS method and its modified versions have come into prominence recently due to its efficiency and therefore new software tools or packages for a quick evaluation is required. A free software called Visual Sample Plan (VSP) created by Pacific Northwest National Labaratory has many sampling designs including classical RSS method for developing environmental sampling plans under balanced and unbalanced cases. It provides the calculation of the required sample size and cost information with the location to be sampled. Also, a package **NSM3** by Schneider (2015) in R has two functions related to classical RSS method. It only provides the Monte Carlo samples and computes a statistic for a nonparametric procedure. Both the VSP and **NSM3** package include only the classical RSS method as a sampling procedure and provide limited methods for inference. Therefore, there is no extensive package for sampling and statistical inference using both classical and modified RSS methods in any available software packages. In this study, we propose a pioneering package, named **RSSampling**, for sampling procedures based on the classical RSS and the modified RSS methods in both perfect and imperfect ranking cases. Also, the package provides the estimation of the mean and the variance of the population and allows the use of the one sample sign, Mann-Whitney-Wilcoxon, and Wilcoxon signed-rank test procedures under classical RSS. The organization of the paper is as follows: in the following section, we give some brief information about classical RSS and modified RSS methods. Then, we introduce the details of **RSSampling** package and further, we give some illustrative examples with a real data analysis. In the last section, we give the conclusion of the study.

The classical and modified RSS methods

RSS and its modifications are advanced sampling methods using the rank information of the sample units. The ranking of the units can be done by visual inspection of a human expert or a concomitant variable. The procedure for the RSS method is as follows:

1. Select m units at random from a specified population.
2. Rank these m units by judgment without actual measurement.
3. Keep the smallest judged unit from the ranked set.
4. Select second set of m units at random from a specified population, rank these units without measuring them, keep the second smallest judged unit.
5. Continue the process until m ranked units are measured.

The first five steps are referred to as a cycle. Then, the cycle repeats r times and a ranked set sample of size $n = mr$ is obtained. Figure 1 illustrates the RSS procedure with visual inspection for the case of $r = 1$ and $m = 3$, and in the following scheme, $X_{i(j:m)}$ represents the jth ranked unit in ith set where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, m$ and bold units represent the units which are chosen to ranked set sample.

$$\begin{pmatrix} \mathbf{X_{1(1:3)}} \leqslant X_{1(2:3)} \leqslant X_{1(3:3)} \\ \\ X_{2(1:3)} \leqslant \mathbf{X_{2(2:3)}} \leqslant X_{2(3:3)} \\ \\ X_{3(1:3)} \leqslant X_{3(2:3)} \leqslant \mathbf{X_{3(3:3)}} \end{pmatrix}$$

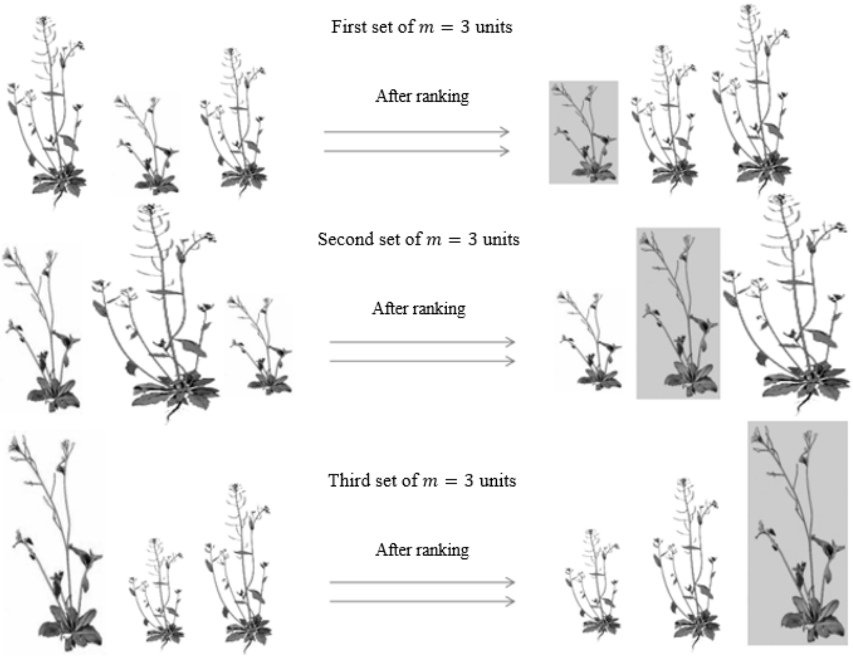

First set of m = 3 units
After ranking
Second set of m = 3 units
After ranking
Third set of m = 3 units
After ranking

Figure 1: Ranking with visual inspection for one cycle, Haq et al. (2013)

RSS design obtains more representative samples and gives more precise estimates of the population parameters relative to SRS (EPA, 2012). The main difference between the RSS method and the other modified methods is the selection procedure of the sample units from the ranked sets. For example, Samawi et al. (1996) suggested extreme RSS using the minimum or maximum units from each ranked set. Muttlak (1997) introduced median RSS using only median units of the random sets. Jemain et al. (2008) suggested balanced groups RSS which is defined as the combination of extreme

RSS and median RSS. For additional examples of modified methods, see Muttlak (2003a), Al-Saleh and Al-Kadiri (2000), and for robust methods see, Al-Nasser (2007), Al-Omari and Raqab (2013), and Al-Nasser and Mustafa (2009). In literature, the studies for modified RSS methods are generally interested in obtaining a sample more easily or making a more robust estimation for a population parameter. Such studies are made for the investigation of properties (for example, bias and mean squared error) of a proposed estimator and they have generally focused on the comparisons of SRS and RSS methods. Note that the true comparisons of the modified RSS methods to the others are difficult to present in general terms. Because the advantages of the sampling methods, when compared to each other, may vary according to the situations such as the parameter to be estimated, underlying distribution, the presence of ranking error, etc. For more detailed information on the modifications of RSS, see Al-Omari and Bouza (2014) and references therein. In the following, the modified RSS methods which are considered in **RSSampling** are introduced.

Extreme RSS

Extreme RSS (ERSS) is the first modification of RSS suggested by Samawi et al. (1996) to estimate the population mean only using the minimum or maximum ranked units from each set. The procedure for ERSS can be described as follows: select m random sets each of size m units from the population and rank the units within each set by a human expert or a concomitant variable. If the set size m is even, the lowest ranked units of each set are chosen from the first $m/2$ sets, and the largest ranked units of each set are chosen from the other $m/2$ sets. If the set size is odd, the lowest ranked units from the first $(m-1)/2$ sets, the largest ranked units from the other $(m-1)/2$ sets and median unit from the remaining last set are chosen. If we repeat the procedure r times, we have a sample of size $n = mr$. An example of the procedure for $r = 1$ and $m = 4$ is shown below.

$$\begin{pmatrix} \mathbf{X_{1(1:4)}} \leqslant X_{1(2:4)} \leqslant X_{1(3:4)} \leqslant X_{1(4:4)} \\ \\ \mathbf{X_{2(1:4)}} \leqslant X_{2(2:4)} \leqslant X_{2(3:4)} \leqslant X_{2(4:4)} \\ \\ X_{3(1:4)} \leqslant X_{3(2:4)} \leqslant X_{3(3:4)} \leqslant \mathbf{X_{3(4:4)}} \\ \\ X_{4(1:4)} \leqslant X_{4(2:4)} \leqslant X_{4(3:4)} \leqslant \mathbf{X_{4(4:4)}} \end{pmatrix}$$

Median RSS

Median RSS (MRSS) was suggested by Muttlak (1997). In this method, only median units of the random sets are chosen as the sample for estimation of population mean. For the odd set sizes, the $((m+1)/2)$th ranked units are chosen as the median of each set. For even set sizes, the $(m/2)$th ranked units are chosen from the first $m/2$ sets and the $((m+2)/2)$th ranked units are chosen from the remaining $m/2$ sets. If necessary, procedure can be repeated r times and we have $n = mr$ sample of size. An example of the procedure for $r = 1$ and $m = 3$ is shown below.

$$\begin{pmatrix} X_{1(1:3)} \leqslant \mathbf{X_{1(2:3)}} \leqslant X_{1(3:3)} \\ \\ X_{2(1:3)} \leqslant \mathbf{X_{2(2:3)}} \leqslant X_{2(3:3)} \\ \\ X_{3(1:3)} \leqslant \mathbf{X_{3(2:3)}} \leqslant X_{3(3:3)} \end{pmatrix}$$

Percentile RSS

Muttlak (2003a) suggested another modification for the RSS, percentile RSS (PRSS), where only the upper and lower percentiles of the random sets are chosen as the sample for selected value of p, where $0 \leqslant p \leqslant 1$. Suppose that m random sets with the size m are chosen from a specific population to sample m units and ranked visually or with a concomitant variable. If the set size is even, the $(p(m+1))$th smallest units from the first $m/2$ sets and the $((1-p)(m+1))$th smallest units from the other $m/2$ sets are chosen. If m is odd, the $(p(m+1))$th smallest units are chosen from the first $(m-1)/2$ sets, the $((1-p)(m+1))$th smallest units are chosen from the other $(m-1)/2$ sets and the median unit is chosen as the mth unit from the last set. An example of the procedure for $r=1$, $m=5$ and $p=0.3$ is as below.

$$\begin{pmatrix}
X_{1(1:5)} \leqslant \mathbf{X_{1(2:5)}} \leqslant X_{1(3:5)} \leqslant X_{1(4:5)} \leqslant X_{1(5:5)} \\
X_{2(1:5)} \leqslant \mathbf{X_{2(2:5)}} \leqslant X_{2(3:5)} \leqslant X_{2(4:5)} \leqslant X_{2(5:5)} \\
X_{3(1:5)} \leqslant X_{3(2:5)} \leqslant X_{3(3:5)} \leqslant \mathbf{X_{3(4:5)}} \leqslant X_{3(5:5)} \\
X_{4(1:5)} \leqslant X_{4(2:5)} \leqslant X_{4(3:5)} \leqslant \mathbf{X_{4(4:5)}} \leqslant X_{4(5:5)} \\
X_{5(1:5)} \leqslant X_{5(2:5)} \leqslant \mathbf{X_{5(3:5)}} \leqslant X_{5(4:5)} \leqslant X_{5(5:5)}
\end{pmatrix}$$

Balanced groups RSS

Balanced groups RSS (BGRSS) can be defined as the combination of ERSS and MRSS. Jemain et al. (2008) suggested to use BGRSS for estimating the population mean with a special sample size $m=3k$. In their study, BGRSS procedure can be described as follows: $m=3k$ (where $k=1,2,3,\ldots$) sets each size of m are selected randomly from a specific population. The sets are randomly allocated into three groups and units in each set are ranked. The smallest units from the first group, median units from the second group and the largest units from the third group of ranked sets are chosen. When the set size is odd, the median unit in the second group is defined as the $((m+1)/2)$th ranked unit in the set and when the set size is even, the median unit is defined as the mean of the $(m/2)$th and the $((m+2)/2)$th ranked units. BGRSS process for one cycle and $k=2$ can be described as below.

$$\begin{pmatrix}
\mathbf{X_{1(1:6)}} \leqslant X_{1(2:6)} \leqslant X_{1(3:6)} \leqslant X_{1(4:6)} \leqslant X_{1(5:6)} \leqslant X_{1(6:6)} \\
\mathbf{X_{2(1:6)}} \leqslant X_{2(2:6)} \leqslant X_{2(3:6)} \leqslant X_{2(4:6)} \leqslant X_{2(5:6)} \leqslant X_{2(6:6)} \\
X_{3(1:6)} \leqslant X_{3(2:6)} \leqslant \mathbf{X_{3(3:6)}} \leqslant \mathbf{X_{3(4:6)}} \leqslant X_{3(5:6)} \leqslant X_{3(6:6)} \\
X_{4(1:6)} \leqslant X_{4(2:6)} \leqslant \mathbf{X_{4(3:6)}} \leqslant \mathbf{X_{4(4:6)}} \leqslant X_{4(5:6)} \leqslant X_{4(6:6)} \\
X_{5(1:6)} \leqslant X_{5(2:6)} \leqslant X_{5(3:6)} \leqslant X_{5(4:6)} \leqslant X_{5(5:6)} \leqslant \mathbf{X_{5(6:6)}} \\
X_{6(1:6)} \leqslant X_{6(2:6)} \leqslant X_{6(3:6)} \leqslant X_{6(4:6)} \leqslant X_{6(5:6)} \leqslant \mathbf{X_{6(6:6)}}
\end{pmatrix}$$

Double RSS

Al-Saleh and Al-Kadiri (2000) introduced another modification of RSS, that is double RSS (DRSS) as a beginning of multistage procedure. Several researchers also extended the DRSS method to modified versions such as double extreme RSS (DERSS) by Samawi (2002), double median RSS (DMRSS) by Samawi and Tawalbeh (2002), and double percentile RSS (DPRSS) by Jemain and Al-Omari (2006). The DRSS procedure is described as follows: m^3 units are identified from the target population and divided randomly into m groups, the size of each is m^2. Then, the usual RSS procedure is used on each group to obtain m ranked set samples each of size m. Finally, RSS procedure is applied again on the obtained ranked set samples in the previous step to get a double ranked set sample of size m.

L-RSS

L-RSS, which is a robust RSS procedure, is based on the idea of L statistic and it was introduced by Al-Nasser (2007) as a generalization of different type of RSS methods. The first step for L-RSS procedure is selecting m random sets with m units and ranking the units in each set. Let k be the L-RSS coefficient, where $k = \lfloor m\alpha \rfloor$ for $0 \leqslant \alpha < 0.5$ and $\lfloor m\alpha \rfloor$ is the largest integer value less than or equal to $m\alpha$. Then, the $(k+1)$th ranked units from the first $k+1$ sets, $(m-k)$th ranked units from the last $k+1$ sets and ith ranked units from the remaining sets which are numbered with i, where $i = k+2, \ldots, m-k-1$ are selected. The L-RSS procedure for the case of $m = 6$ and $k = 1$ $(\alpha = 0.20)$ in a cycle can be shown as below:

$$\begin{pmatrix}
X_{1(1:6)} \leqslant \mathbf{X_{1(2:6)}} \leqslant X_{1(3:6)} \leqslant X_{1(4:6)} \leqslant X_{1(5:6)} \leqslant X_{1(6:6)} \\
X_{2(1:6)} \leqslant \mathbf{X_{2(2:6)}} \leqslant X_{2(3:6)} \leqslant X_{2(4:6)} \leqslant X_{2(5:6)} \leqslant X_{2(6:6)} \\
X_{3(1:6)} \leqslant X_{3(2:6)} \leqslant \mathbf{X_{3(3:6)}} \leqslant X_{3(4:6)} \leqslant X_{3(5:6)} \leqslant X_{3(6:6)} \\
X_{4(1:6)} \leqslant X_{4(2:6)} \leqslant X_{4(3:6)} \leqslant \mathbf{X_{4(4:6)}} \leqslant X_{4(5:6)} \leqslant X_{4(6:6)} \\
X_{5(1:6)} \leqslant X_{5(2:6)} \leqslant X_{5(3:6)} \leqslant X_{5(4:6)} \leqslant \mathbf{X_{5(5:6)}} \leqslant X_{5(6:6)} \\
X_{6(1:6)} \leqslant X_{6(2:6)} \leqslant X_{6(3:6)} \leqslant X_{6(4:6)} \leqslant \mathbf{X_{6(5:6)}} \leqslant X_{6(6:6)}
\end{pmatrix}$$

When $k = 0$, then this procedure leads to the classical RSS and when $k = \lfloor (m-1)/2 \rfloor$, then it leads to the MRSS method.

Truncation-based RSS

The truncation-based RSS (TBRSS) was presented by Al-Omari and Raqab (2013). This procedure can be summarized as follows: select randomly m sets each of size m units from the population and rank the units in each set. Then, determine TBRSS coefficient k as in the L-RSS method and select the minimums of the first k sets and the maximums of the last k sets. From the remaining $m - 2k$ samples, select the ith ranked unit of the ith sample $(k+1 \leqslant i \leqslant m-k)$. The one cycled TBRSS method for the case of $m = 8$ and $k = 2$ $(\alpha = 0.35)$ is shown below.

$$\begin{pmatrix}
\mathbf{X_{1(1:8)}} \leqslant X_{1(2:8)} \leqslant X_{1(3:8)} \leqslant X_{1(4:8)} \leqslant X_{1(5:8)} \leqslant X_{1(6:8)} \leqslant X_{1(7:8)} \leqslant X_{1(8:8)} \\
\mathbf{X_{2(1:8)}} \leqslant X_{2(2:8)} \leqslant X_{2(3:8)} \leqslant X_{2(4:8)} \leqslant X_{2(5:8)} \leqslant X_{2(6:8)} \leqslant X_{2(7:8)} \leqslant X_{2(8:8)} \\
X_{3(1:8)} \leqslant X_{3(2:8)} \leqslant \mathbf{X_{3(3:8)}} \leqslant X_{3(4:8)} \leqslant X_{3(5:8)} \leqslant X_{3(6:8)} \leqslant X_{3(7:8)} \leqslant X_{3(8:8)} \\
X_{4(1:8)} \leqslant X_{4(2:8)} \leqslant X_{4(3:8)} \leqslant \mathbf{X_{4(4:8)}} \leqslant X_{4(5:8)} \leqslant X_{4(6:8)} \leqslant X_{4(7:8)} \leqslant X_{4(8:8)} \\
X_{5(1:8)} \leqslant X_{5(2:8)} \leqslant X_{5(3:8)} \leqslant X_{5(4:8)} \leqslant \mathbf{X_{5(5:8)}} \leqslant X_{5(6:8)} \leqslant X_{5(7:8)} \leqslant X_{5(8:8)} \\
X_{6(1:8)} \leqslant X_{6(2:8)} \leqslant X_{6(3:8)} \leqslant X_{6(4:8)} \leqslant X_{6(5:8)} \leqslant \mathbf{X_{6(6:8)}} \leqslant X_{6(7:8)} \leqslant X_{6(8:8)} \\
X_{7(1:8)} \leqslant X_{7(2:8)} \leqslant X_{7(3:8)} \leqslant X_{7(4:8)} \leqslant X_{7(5:8)} \leqslant X_{7(6:8)} \leqslant X_{7(7:8)} \leqslant \mathbf{X_{7(8:8)}} \\
X_{8(1:8)} \leqslant X_{8(2:8)} \leqslant X_{8(3:8)} \leqslant X_{8(4:8)} \leqslant X_{8(5:8)} \leqslant X_{8(6:8)} \leqslant X_{8(7:8)} \leqslant \mathbf{X_{8(8:8)}}
\end{pmatrix}$$

Note that when $k = 0$ or $k = 1$, TBRSS scheme is equivalent to the classical RSS scheme.

Robust extreme RSS

Robust extreme RSS (RERSS) scheme was introduced by Al-Nasser and Mustafa (2009). This method can be described as follows: identify m random sets with m units and rank the units within each set. Select the $(k+1)$th ranked units from the first $m/2$ sets where $k = \lfloor m\alpha \rfloor$ for $0 < \alpha < 0.5$ and $\lfloor m\alpha \rfloor$ is the largest integer value less than or equal to $m\alpha$. Then, select the $(m-k)$th ranked units from the other $m/2$ sets. If the set size m is odd, $((m+1)/2)$th ranked unit is selected additionally from the last remaining set. The procedure for one cycle and the case of $m = 6$ and $k = 1$ ($\alpha = 0.20$) can be shown as below.

$$\begin{pmatrix}
X_{1(1:6)} \leqslant \mathbf{X_{1(2:6)}} \leqslant X_{1(3:6)} \leqslant X_{1(4:6)} \leqslant X_{1(5:6)} \leqslant X_{1(6:6)} \\
X_{2(1:6)} \leqslant \mathbf{X_{2(2:6)}} \leqslant X_{2(3:6)} \leqslant X_{2(4:6)} \leqslant X_{2(5:6)} \leqslant X_{2(6:6)} \\
X_{3(1:6)} \leqslant \mathbf{X_{3(2:6)}} \leqslant X_{3(3:6)} \leqslant X_{3(4:6)} \leqslant X_{3(5:6)} \leqslant X_{3(6:6)} \\
X_{4(1:6)} \leqslant X_{4(2:6)} \leqslant X_{4(3:6)} \leqslant X_{4(4:6)} \leqslant \mathbf{X_{4(5:6)}} \leqslant X_{4(6:6)} \\
X_{5(1:6)} \leqslant X_{5(2:6)} \leqslant X_{5(3:6)} \leqslant X_{5(4:6)} \leqslant \mathbf{X_{5(5:6)}} \leqslant X_{5(6:6)} \\
X_{6(1:6)} \leqslant X_{6(2:6)} \leqslant X_{6(3:6)} \leqslant X_{6(4:6)} \leqslant \mathbf{X_{6(5:6)}} \leqslant X_{6(6:6)}
\end{pmatrix}$$

If $k = 0$ and $k = (m/2)$, then this sampling procedure corresponds to ERSS and MRSS methods, respectively.

RSSampling package

The package **RSSampling** is available on CRAN and can be installed and loaded via the following commands:

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

The package depends on the **stats** package and uses a function from the non-standard package **LearnBayes** (Albert, 2018) for random data generation in the Examples section. The proposed package consists of two main parts which are the functions for sampling methods described in Table 1 and the functions for inference procedures described in Table 2 based on RSS. The sampling part of the package includes perfect and imperfect rankings with a concomitant variable allowing researchers to sample with classical RSS and the modified versions. The functions for inference procedures provide estimation for parameters and some hypothesis testing procedures based on RSS.

Sampling with RSSampling

In this part, we introduce a core function, which is called `rankedsets`, to obtain `s` ranked sets consisting of randomly chosen sample units with the set size m. By using this function, we developed the functions given in Table 1 which provide researchers means to obtain a sample under different sampling schemes. One can also use `rankedsets` function for the studies based on other modified RSS methods which are not mentioned in this paper.

The function `rss` provides the ranked set sample with perfect ranking from a specific data set, X, provided in matrix form where the columns and rows represent the sets and cycles, respectively. One can see the randomly chosen ranked sets by defining `sets = TRUE` (default `sets = FALSE`) with the set size m and the cycle size r. For the modified RSS methods, the function `Mrss` provides a sample from MRSS, ERSS, PRSS, and BGRSS which are represented by `"m"`,`"e"`, `"p"`, and `"bg"`, respectively. The `type = "r"`, defined as the default, represents the classical RSS. For the sampling

Function	Description
`rss`	Performs classical RSS method
`Mrss`	Performs modified RSS methods (MRSS, ERSS, PRSS,BGRSS)
`Rrss`	Performs robust RSS methods (L-RSS, TBRSS, RERSS)
`Drss`	Performs double RSS methods (DRSS, DMRSS, DERSS, DPRSS)
`con.rss`	Performs classical RSS method by using a concomitant variable
`con.Mrss`	Performs modified RSS methods (MRSS, ERSS, PRSS,BGRSS) by using a concomitant variable
`con.Rrss`	Performs robust RSS methods (L-RSS, TBRSS, RERSS) by using a concomitant variable
`obsno.Mrss`	Determines the observation numbers of the units which will be chosen to the sample for classical and modified RSS methods by using a concomitant variable

Table 1: The functions for the sampling methods in **RSSampling** package

procedure PRSS, there is an additional parameter `p` which defines the percentile. We note that, when `p = 0.25` in PRSS, one can obtain a sample with quartile RSS given by Muttlak (2003b). `Rrss` provides samples from L-RSS, TBRSS, and RERSS methods which are represented by `"l"`, `"tb"`, and `"re"`, respectively. The parameter `alpha` is the common parameter for these methods and defines the cutting value. `Drss` function is for double versions of RSS, MRSS, ERSS, and PRSS under perfect ranking. `type = "d"` is defined as the default which represents the double RSS. Values `"dm"`, `"de"`, and `"dp"` are defined for DMRSS, DERSS, and DPRSS methods, respectively.

In the literature, most of the theoretical inferences and numerical studies are conducted based on perfect ranking. However, in real life applications, the ranking process is done with an expert judgment or a concomitant variable. Let us consider RSS with a concomitant variable Y. A set of m units is drawn from the population, then the units are ranked by the order of Y. The concomitant variable $Y_{i(j:m)}$ represents the jth ranked unit in ith set and the variable of interest $X_{(i,j)}$ represents the jth unit in ith set, where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, m$. In the following example, the procedure of RSS using Y is given for $m = 3$.

$$(\mathbf{Y_{1(1:3)}}, \mathbf{X_{(1,1)}}) \leqslant \; (Y_{1(2:3)}, X_{(1,2)}) \leqslant \; (Y_{1(3:3)}, X_{(1,3)}) \; \longrightarrow \mathbf{X_{(1,1)}}$$

$$(Y_{2(1:3)}, X_{(2,1)}) \leqslant \; (\mathbf{Y_{2(2:3)}}, \mathbf{X_{(2,2)}}) \leqslant \; (Y_{2(3:3)}, X_{(2,3)}) \; \longrightarrow \mathbf{X_{(2,2)}}$$

$$(Y_{3(1:3)}, X_{(3,1)}) \leqslant \; (Y_{3(2:3)}, X_{(3,2)}) \leqslant \; (\mathbf{Y_{3(3:3)}}, \mathbf{X_{(3,3)}}) \longrightarrow \mathbf{X_{(3,3)}}$$

The functions `con.rss`, `con.Mrss`, and `con.Rrss` provide methods to obtain a sample under imperfect ranking. With the `con.rss` function, a researcher can obtain a classical ranked set sample from a specific data set using a concomitant variable Y with the set size m and cycle size r to make inference about the variable of interest X. The functions `con.Mrss` and `con.Rrss` have similar usage with `con.rss` function except the selection method which is defined by `type` parameter. Also, these functions are simply extensions of the `Mrss` and `Rrss` for concomitant variable cases.

In a real-world research, the values of the variable of interest X are unknown and the researchers measure X values of the sample units after choosing them from the population with a specific sampling method. The function `obsno.Mrss` provides the code for this kind of application, when the researchers prefer to use RSS methods. After determining the sample frame and the concomitant variable to be used for ranking, the code provides the number of the units to be selected according the values of the concomitant variable. Then, the researcher obtain easily the observation numbers of the units which will be chosen to the sample. `type = "r"` is defined as the default which represents the classical RSS. MRSS, ERSS, PRSS, and BGRSS are represented by `"m"` , `"e"`, `"p"`, and `"bg"`, respectively.

Inference with RSSampling

Statistical inference refers to the process of drawing conclusions and having an information about the interested population. Researchers are generally interested in fundamental inferences for the parameters such as mean and variance. Using the **RSSampling** package, we provide an easy way to estimate the parameters about the interested population and to use some distribution-free tests; namely the sign, Mann-Whitney-Wilcoxon, and Wilcoxon signed-rank tests for nonparametric inference when the sampling procedure is RSS.

Function	Description
`meanRSS`	Performs mean estimation and hypothesis testing with classical RSS method
`varRSS`	Performs variance estimation with classical RSS method
`regRSS`	Performs regression estimation for mean of interested population with classical RSS method
`sign1testrss`	Performs one sample sign test with classical RSS method
`mwwutestrss`	Performs Mann-Whitney-Wilcoxon test with classical RSS method
`wsrtestrss`	Performs Wilcoxon signed-rank test with classical RSS method

Table 2: The functions for inference in **RSSampling** package

The `meanRSS` function provides point estimation, confidence interval estimation, and asymptotic hypothesis testing for the population mean based on RSS see, (Chen et al., 2003). For the variance estimation based on RSS, we define `varRSS` function which has two `type` parameters; `"Stokes"` and `"Montip"`. Stokes (1980b) proved that estimator of variance is asymptotically unbiased regardless of presence of ranking error. For the `"Montip"` type estimation, Tiensuwan and Sarikavanij (2003) showed that there is no unbiased estimator of variance for one cycle but they proposed unbiased estimator of variance for more than one cycle. With `regRSS` function, regression estimator for mean of interested population can be obtained based on RSS. The β coefficient (`"B"` in `regRSS` function) is calculated under the assumption of known population mean for concomitant Y. Note that, the ranked set samples for interested variable X and for concomitant variable Y must be the same length. One can find the detailed information about regression estimator based on RSS in Yu and Lam (1997).

Finally, for nonparametric inference, `sign1testrss`, `mwwutestrss`, and `wsrtestrss` functions implement, respectively, the sign test, the Mann-Whitney-Wilcoxon test, and the Wilcoxon signed-rank test depending on RSS. The normal approximation is used to construct the test statistics and an approximate confidence intervals. For detailed information on these test methods, see the book of Chen et al. (2003).

Examples

In this section, we present examples illustrating the **RSSampling** package.

Sampling with TBRSS using a concomitant variable

This example shows the process to obtain a sample by using TBRSS method for the variable of interest, X, ranked by using the concomitant variable Y assuming that they are distributed as multivariate normal. We determined the set size `m` is 4 and the cycle size `r` is 2. The ranked sets of Y and the sets of X are obtained using the function `con.Rrss`. Thus, the resultant sample for X is given as below.

```
##Loading packages
library("RSSampling")
library("LearnBayes")

## Imperfect ranking example for interested (X) and concomitant (Y) variables
## from multivariate normal dist.
set.seed(1)
mu <- c(10, 8)
variance <- c(5, 3)
a <- matrix(c(1, 0.9, 0.9, 1), 2, 2)
v <- diag(variance)
Sigma <- v%*%a%*%v
x <- rmnorm(10000, mu, Sigma)
xx <- as.numeric(x[,1])
xy <- as.numeric(x[,2])

## Selecting a truncation-based ranked set sample
con.Rrss(xx, xy, m = 4, r = 2, type = "tb", sets = TRUE, concomitant = FALSE,
         alpha = 0.25)

$corr.coef
[1] 0.9040095

$var.of.interest
         [,1]      [,2]      [,3]     [,4]
[1,] 12.332134 13.116611 15.675967 21.72312
[2,] 11.350275  8.846237 10.164005 17.07950
[3,]  4.143757  9.608573  8.708221 11.57671
[4,]  2.284106  9.535388 12.709489 14.11595
[5,]  3.212739  8.089833 11.430411 14.53190
[6,]  6.556222 12.759335 13.210037 11.02219
[7,]  3.337564 -0.864634 12.800243 13.47315
[8,]  5.988893  8.850680 13.208956 15.82731
```

```
$concomitant.var.
         [,1]      [,2]      [,3]      [,4]
[1,] 8.034720 10.398398 11.800919 13.754743
[2,] 8.003575  8.118947 11.136804 12.149531
[3,] 4.733177  7.377396  8.866563 11.658837
[4,] 4.027061  8.008146  9.977435 10.912382
[5,] 3.909958  6.220087  7.564130  8.739562
[6,] 5.893001  8.760754 10.067927 10.244593
[7,] 2.119661  2.813413 10.651769 10.775596
[8,] 5.406154  7.722866  8.602551 10.874853

$sample.x
         m = 1     m = 2     m = 3    m = 4
r = 1 12.332134  8.846237  8.708221 14.11595
r = 2  3.212739 12.759335 12.800243 15.82731
```

Obtaining observation number in MRSS method

Random determination of the sample units is an important task for practitioners. The function `obsno.Mrss` is for the practitioners who have the frame of the population with unknown variable X and known concomitant variable Y. In the following example, the observation numbers for median ranked set sample units are obtained in order to take the measurement of the interested variable X.

```
## Loading packages
library("RSSampling")

## Generating concomitant variable (Y) from exponential dist.
set.seed(5)
y = rexp(10000)

## Determining the observation numbers of the units which are chosen to sample
obsno.Mrss(y, m = 3, r = 5, type = "m")

              m = 1       m = 2       m = 3
      r = 1 "Obs.  2452" "Obs.  6417" "Obs.  3227"
      r = 2 "Obs.  9094" "Obs.  1805" "Obs.  9877"
      r = 3 "Obs.  1333" "Obs.  9252" "Obs.  3219"
      r = 4 "Obs.  6397" "Obs.  7038" "Obs.  5019"
      r = 5 "Obs.  446"  "Obs.  9663" "Obs.  10"
```

A simulation study based on RSS using a concomitant variable

In order to illustrate the usage of the package, we give a simulation study with 10,000 repetitions for mean estimation of X based on RSS method using a concomitant variable. It demonstrates the effect of the correlation level between X and Y on the mean squared error (MSE) of estimation. Samples are obtained when `m = 5` and `r = 10` assuming that X and Y are distributed as multivariate normal. Figure 2 as an output of the simulation study indicates that when the correlation level is increasing, MSE values are decreasing.

```
## Loading packages
library("RSSampling")
library("LearnBayes")

## Imperfect ranking example for interested (X) and concomitant (Y) variables
## from multivariate normal dist.
mu <- c(10, 8)
variance <- c(5, 3)
rho = seq(0, 0.9, 0.1)
se.x = mse.x = numeric()
repeatsize = 10000
```

```
for (i in 1:length(rho)) {
    set.seed(1)
    a <- matrix(c(1, rho[i], rho[i], 1), 2, 2)
    v <- diag(variance)
    Sigma <- v%*%a%*%v
    x <- rmnorm(10000, mu, Sigma)
    xx <- as.numeric(x[,1])
    xy <- as.numeric(x[,2])
    for (j in 1:repeatsize) {
        set.seed(j)
        samplex = con.Mrss(xx, xy, m = 5, r = 10, type = "r", sets = FALSE,
                           concomitant = FALSE)$sample.x
        se.x[j] = (mean(samplex)-mu[1])^2
    }
    mse.x[i] = sum(se.x)/repeatsize
}
plot(rho[-1], mse.x[-1], type = "o", lwd = 2,
     main = "MSE values based on increasing correlation levels",
     xlab = "corr.coef.", ylab = "MSE", cex = 1.5, xaxt = "n")
axis(1, at = seq(0.1, 0.9, by = 0.1))
```

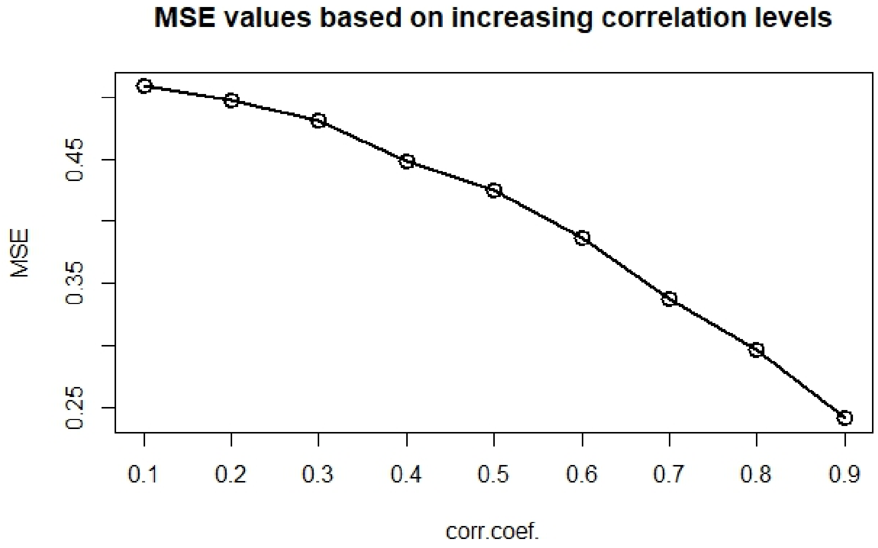


Figure 2: MSE values based on increasing correlation levels

A real data example

In this real data example, we used the abolone data set which is freely available at https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data. The data consists of 9 variables of 4177 units and the variables are; sex (Male/Female/Infant), length (mm), diameter (mm), height (mm), whole weight (grams), shucked weight (grams), viscera weight (grams), shell weight (grams), and rings (+1.5 gives the age of abalone in years), respectively. The data comes from an original study of the population biology of abalone by Nash et al. (1994). Also, Cetintav et al. (2016) and Sevinç et al. (2018) used the abalone data set for application of the fuzzy based modification of RSS and partial groups RSS methods, respectively. The data set can be obtained easily by using the following R command.

```
abaloneData <- read.csv(url("https://archive.ics.uci.edu/ml/machine-learning-databases
               /abalone/abalone.data"), header = FALSE, col.names = c("sex", "length",
               "diameter", "height", "whole.weight", "shucked.weight", "viscera.weight",
               "shell.weight", "rings"))
```

Suppose that we aimed to estimate the mean of *viscera weight* and confidence interval and also test the hypothesis claiming that the mean of the *viscera weight* is equal to 0.18. The measurement of *viscera weight* which is the gut weight of abalone after bleeding is an expensive and time-consuming process. Because the measurement of *whole weight* is easy and highly correlated with *viscera weight* (the correlation coefficient is 0.966), we used *whole weight* as the concomitant variable to obtain a sample of size 25 in RSS method. We have the following results for *viscera weight*.

```
cor(abaloneData$viscera.weight, abaloneData$whole.weight)
[1] 0.9663751

set.seed(50)
sampleRSS = con.rss(abaloneData$viscera.weight, abaloneData$whole.weight, m = 5, r = 5,
                    sets = TRUE, concomitant = FALSE)$sample.x

meanRSS(sampleRSS, m = 5, r = 5, alpha = 0.05, alternative = "two.sided", mu_0 = 0.18)
$mean
[1] 0.17826

$CI
[1] 0.1293705 0.2271495

$z.test
[1] -0.06975604

$p.value
[1] 0.9443878

varRSS(sampleRSS, m = 5, r = 5, type = "Stokes")
[1] 0.0135364
```

The results from our sample data indicate that the estimated mean and the variance are 0.17826 and 0.01354, respectively. According to the hypothesis testing result, we conclude that there is no strong evidence against the null hypothesis (`p.value`$>$ 0.05).

Conclusion

RSS is an efficient data collection method compared to SRS especially in situations where the measurement of a unit is expensive but the ranking is less costly. In this study, we propose a package which obtains sample from RSS and its modifications and provide functions to allow some inferential procedures by RSS. We create a set of functions for sampling under both perfect and imperfect rankings with a concomitant variable. For the inferential procedures, we consider mean, variance, and regression estimator and sign, Mann-Whitney-Wilcoxon, and Wilcoxon signed-rank tests for the distribution free tests. Proposed functions in the package are illustrated with the examples and analysis of a real data is given. Future improvements of the package may be provided by adding new inference procedures based on RSS methods.

Acknowledgments

The authors thank two anonymous referees and the associate editor for their helpful comments and suggestions which improved the presentation of the paper. This study is supported by the Scientific and Technological Research Council of Turkey (TUBITAK-COST Grant No. 115F300) under ISCH COST Action IS1304.

Bibliography

A. D. Al-Nasser. L ranked set sampling: A generalization procedure for robust visual sampling. *Communications in Statistics—Simulation and Computation*, 36(1):33–43, 2007. URL https://doi.org/10.1080/03610910601096510. [p403, 405]

A. D. Al-Nasser and A. B. Mustafa. Robust extreme ranked set sampling. *Journal of Statistical Computation and Simulation*, 79(7):859–867, 2009. URL https://doi.org/10.1080/00949650701683084. [p403, 406]

A. I. Al-Omari and C. N. Bouza. Review of ranked set sampling: Modifications and applications. *Revista Investigación Operacional*, 3(35):215–240, 2014. [p403]

A. I. Al-Omari and M. Z. Raqab. Estimation of the population mean and median using truncation-based ranked set samples. *Journal of Statistical Computation and Simulation*, 83(8):1453–1471, 2013. URL https://doi.org/10.1080/00949655.2012.662684. [p403, 405]

M. F. Al-Saleh and M. A. Al-Kadiri. Double-ranked set sampling. *Statistics and Probability Letters*, 48(2):205–212, 2000. URL https://doi.org/10.1016/S0167-7152(99)00206-0. [p403, 404]

J. Albert. *LearnBayes: Functions for Learning Bayesian Inference*, 2018. URL https://cran.r-project.org/web/packages/LearnBayes/index.html. [p406]

L. L. Bohn and D. A. Wolfe. Nonparametric two sample procedures for ranked-set sampling data. *Journal of the American Statistical Association*, 87:552–561, 1992. URL https://doi.org/10.1080/01621459.1992.10475239. [p401]

L. L. Bohn and D. A. Wolfe. The effect of imperfect judgment rankings on properties of procedures based on the ranked-set samples analog of the Mann-Whitney-Wilcoxon statistic. *Journal of the American Statistical Association*, 89:168–176, 1994. URL https://doi.org/10.1080/01621459.1994.10476458. [p401]

B. Cetintav, G. Ulutagay, S. Gurler, and N. Demirel. Mean estimation based on fwa using ranked set sampling with single and multiple rankers. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 790–797. Springer, 2016. URL https://doi.org/10.1007/978-3-319-40581-0_64. [p411]

Z. Chen. On ranked-set sample quantiles and their applications. *Journal of Statistical Planning and Inference*, 83:125–135, 2000. URL https://doi.org/10.1016/S0378-3758(99)00071-3. [p401]

Z. Chen. Ranked-set sampling with regression-type estimators. *Journal of Statistical Planning and Inference*, 92:181–192, 2001. URL https://doi.org/10.1016/S0378-3758(00)00140-3. [p401]

Z. Chen, Z. Bai, and B. Sinha. *Ranked Set Sampling: Theory and Applications*, volume 176. Springer-Verlag, 2003. URL https://doi.org/10.1007/978-0-387-21664-5. [p401, 408, 409]

H. A. David and D. N. Levine. Ranked set sampling in the presence of judgment error. *Biometrics*, 28:553–555, 1972. [p401]

T. R. Dell and J. L. Clutter. Ranked set sampling theory with order statistics background. *Biometrika*, 28:545–555, 1972. URL https://doi.org/10.2307/2556166. [p401]

U. S. E. P. A. EPA. *Guidance for Choosing a Sampling Design for Environmental Data Collection for Use in Developing a Quality Assurance Project Plan (EPA QA/G-5S)*. Office of Environmental Information, Washington, DC, 2012. [p402]

L. S. Halls and T. R. Dell. Trial of ranked set sampling for forage yields. *Forest Science*, 12:22–26, 1966. URL https://doi.org/10.1093/forestscience/12.1.22. [p401]

A. Haq, J. Brown, E. Moltchanova, and A. I. Al-Omari. Partial ranked set sampling design. *Environmetrics*, 24(3):201–207, 2013. URL https://doi.org/10.1002/env.2203. [p402]

T. P. Hettmansperger. The ranked set sample sign test. *Journal of Nonparametric Statistics*, 4:263–270, 1995. URL https://doi.org/10.1080/10485259508832617. [p401]

A. Jemain and A. Al-Omari. Double percentile ranked set samples for estimating the population mean. *Advances and Applications in Statistics*, 6(3):261–276, 2006. [p404]

A. A. Jemain, A. I. Al-Omari, and K. Ibrahim. Some variations of ranked set sampling. *Electronic Journal of Applied Statistical Analysis*, 1:1–15, 2008. URL https://doi.org/10.1285/i20705948v1n1p1. [p402, 404]

A. Kaur, G. Patil, A. Sinha, and C. Taillie. Ranked set sampling: An annotated bibliography. *Environmental and Ecological Statistics*, 2(1):25–54, 1995. URL https://doi.org/10.1007/BF00452930. [p401]

P. H. Kvam and F. J. Samaniego. On the inadmissibility of empirical averages as estimators in ranked set sampling. *Journal of Statistical Planning and Inference*, 36:39–55, 1993. URL https://doi.org/10.1016/0378-3758(93)90100-K. [p401]

G. A. McIntyre. A method of unbiased selective sampling using ranked sets. *Australian Journal of Agricultural Research*, 3:385–390, 1952. URL https://doi.org/10.1071/AR9520385. [p401]

H. A. Muttlak. Median ranked set sampling. *J. Appl. Statist. Sci*, 6:245–255, 1997. [p402, 403]

H. A. Muttlak. Modified ranked set sampling methods. *Pakistan Journal of Statistics*, 19(3):315–323, 2003a. [p403]

H. A. Muttlak. Investigating the use of quartile ranked set samples for estimating the population mean. *Applied Mathematics and Computation*, 146(2):437–443, 2003b. URL https://doi.org/10.1016/S0096-3003(02)00595-7. [p407]

W. J. Nash, T. L. Sellers, S. R. Talbot, A. J. Cawthorn, and W. B. Ford. The population biology of abalone (haliotis species) in tasmania. i. blacklip abalone (h. rubra) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report*, (48), 1994. [p411]

O. Ozturk. Ratio estimators based on a ranked set sample in a finite population setting. *Journal of the Korean Statistical Society*, 47(2):226–238, 2018. URL https://doi.org/10.1016/j.jkss.2018.02.001. [p401]

H. M. Samawi. On double extreme rank set sample with application to regression estimator. *Metron International Journal of Statistics*, 60:50–63, 2002. [p404]

H. M. Samawi and E. M. Tawalbeh. Double median ranked set sample: Comparing to other double ranked samples for mean and ratio estimators. *Journal of Modern Applied Statistical Methods*, 1 (2):52, 2002. URL https://doi.org/10.22237/jmasm/1036109460. [p404]

H. M. Samawi, W. Abu Dayyeh, and M. S. Ahmed. Estimating the population mean using extreme ranked set sampling. *Biometrical Journal*, 38:577–568, 1996. URL https://doi.org/10.1002/bimj.4710380506. [p402, 403]

G. Schneider. *NSM3: Functions and Datasets to Accompany Hollander, Wolfe, and Chicken–Nonparametric Statistical Methods*, 2015. URL https://CRAN.R-project.org/package=NSM3. [p401]

B. Sevinç, S. Gürler, and B. Çetintav. Partial groups ranked set sampling and mean estimation. *Journal of Statistical Computation and Simulation*, pages 1–12, 2018. URL https://doi.org/10.1080/00949655.2018.1488255. [p411]

S. L. Stokes. Ranked set sampling with concomitant variables. *Communications in Statistics - Theory and Methods*, 12:1207–1211, 1977. URL https://doi.org/10.1080/03610927708827563. [p401]

S. L. Stokes. Inferences on the correlation coefficient in bivariate normal populations from ranked set samples. *Journal of the American Statistical Association*, 75:989–995, 1980a. URL https://doi.org/10.1080/01621459.1980.10477584. [p401]

S. L. Stokes. Estimation of variance using judgment ordered ranked set samples. *Biometrics*, 36: 35–42, 1980b. URL https://doi.org/10.2307/2530493. [p401, 409]

S. L. Stokes and T. W. Sager. Characterization of a ranked set sample with application to estimating distribution functions. *Journal of the American Statistical Association*, 83:374–381, 1988. URL https://doi.org/10.1080/01621459.1988.10478607. [p401]

K. Takahasi and K. Wakimoto. On unbiased estimates of the population mean based on the sample stratified by means of ordering. *Annals of the Institute of Statistical Mathematics*, 20:1:1–31, 1968. URL https://doi.org/10.1007/BF02911622. [p401]

M. Tiensuwan and S. Sarikavanij. On estimation of population variance based on a ranked set sample. *J Appl Stat Sci*, 12:283–295, 2003. [p409]

D. A. Wolfe. Ranked set sampling: Its relevance and impact on statistical inference. *ISRN Probability and Statistics*, 2012:1–32, 2012. URL https://doi.org/10.5402/2012/568385. [p401]

P. L. H. Yu and K. Lam. Regression estimator in ranked set sampling. *Biometrics*, pages 1070–1080, 1997. URL https://doi.org/10.2307/2533564. [p401, 409]

E. Zamanzade and M. Vock. Variance estimation in ranked set sampling using a concomitant variable. *Statistics & Probability Letters*, 105:1–5, 2015. URL https://doi.org/10.1016/j.spl.2015.04.034. [p401]

Z. Zhang, T. Liu, and B. Zhang. Jackknife empirical likelihood inferences for the population mean with ranked set samples. *Statistics & Probability Letters*, 108:16–22, 2016. URL https://doi.org/10.1016/j.spl.2015.09.016. [p401]

Busra Sevinc
The Graduate School of Natural and Applied Sciences, Dokuz Eylul University
Izmir, Turkey
busra.sevincc@gmail.com

Bekir Cetintav
Department of Statistics, Mehmet Akif Ersoy University
Burdur, Turkey
bekircetintav@mehmetakif.edu.tr

Melek Esemen
The Graduate School of Natural and Applied Sciences, Dokuz Eylul University
Izmir, Turkey
melek.esemen@gmail.com

Selma Gurler
Department of Statistics, Dokuz Eylul University
Izmir, Turkey
selma.erdogan@deu.edu.tr

swgee: An R Package for Analyzing Longitudinal Data with Response Missingness and Covariate Measurement Error

by Juan Xiong and Grace Y. Yi

Abstract Though longitudinal data often contain missing responses and error-prone covariates, relatively little work has been available to simultaneously correct for the effects of response missingness and covariate measurement error on analysis of longitudinal data. Yi (2008) proposed a simulation based marginal method to adjust for the bias induced by measurement error in covariates as well as by missingness in response. The proposed method focuses on modeling the marginal mean and variance structures, and the missing at random mechanism is assumed. Furthermore, the distribution of covariates are left unspecified. These features make the proposed method applicable to a broad settings. In this paper, we develop an R package, called **swgee**, which implements the method proposed by Yi (2008). Moreover, our package includes additional implementation steps which extend the setting considered by Yi (2008). To describe the use of the package and its main features, we report simulation studies and analyses of a data set arising from the Framingham Heart Study.

Introduction

Longitudinal studies are commonly conducted in the health sciences, biochemical, and epidemiology fields; these studies typically collect repeated measurements on the same subject over time. Missing observations and covariate measurement error frequently arise in longitudinal studies and they present considerable challenges in statistical inference about such data (Carroll et al., 2006; Yi, 2008). It has been well documented that ignoring missing responses and covariate measurement error may lead to severely biased results, thus leading to invalid inferences (Fuller, 1987; Carroll et al., 2006).

Regarding longitudinal data with missing responses, there has been extensive methods such as maximum likelihood, multiple imputation, and weighted generalized estimating equations (GEE) method (Little and Rubin, 2002). In terms of methods of handling measurement error in covariate, many methods have been developed for various settings. Comprehensive discussions can be found in Fuller (1987), Gustafson (2003), Carroll et al. (2006), Buonaccorsi (2010) and Yi (2017). However, there has been relatively little work on simultaneously addressing the effects of response missingness and covariate measurement error in longitudinal data analysis, although some work such as Wang et al. (2008), Liu and Wu (2007) and Yi et al. (2012), are available. In particular, Yi (2008) proposed an estimation method based on the marginal model for the response process, which does not require the full specification of the distribution of the response variable but models only the mean and variance structures. Furthermore, a functional method is applied to relax the need of modeling the covariate process. These features make the method of Yi (2008) flexible for many applications.

Relevant to our R package, a set of R packages and statistical software have been available for performing the GEE and weighted GEE analyses for longitudinal data with missing observations. In particular, package **gee** (Carey, 2015) and **yags** (Carey, 2011) perform the GEE analyses under the strong assumption of missing completely at random (MCAR) (Kenward, 1998). Package **wgeesel** (Xu et al., 2018) can perform the multiple model selection based on weighted GEE/GEE. Package **geepack** (Hojsgaard et al., 2016) implements the weighted GEE analyses under the missing at random (MAR) assumption, in which an optional vector of weights can be used in the fitting process but the weight vector has to be externally calculated. In addition, the statistical software SAS/STAT version 13.2 (SAS Institute Inc., 2014) includes an experimental version of the function PROC GEE (Lin and Rodriguez, 2015), which fits weighted GEE models.

Our **swgee** package has several features distinguishing from existing packages. First, **swgee** is designed to analyze longitudinal data with both missing responses and error-prone covariates. To the best of our knowledge, this is the first R package that can simultaneously account for response missingness and covariate measurement error. Secondly, this simulation based marginal method can be applied to a broad range of problems because the associated model assumptions are minimal. **swgee** can be directly applied to handle continuous and binary responses as well as count data with dropouts under the MAR and MCAR mechanisms. Thirdly, observations are weighted inversely proportional to their probability of being observed, with weights calculated internally. Lastly, the **swgee** package employs the simulation extrapolation (SIMEX) algorithm to account for the effect of measurement error in covariates.

The remainder is organized as follows. Section Notation and framework introduces the notation and model setup. In Section Methodology, we describe the method proposed by Yi (2008) and its implementation in R in Section Implementation in R. The developed R package is illustrated with simulation studies and analyses of a data set arising from the Framingham Heart Study in Section Examples. General discussion is included in Section Summary and discussion.

Notation and framework

For $i = 1, \ldots, n$ and $j = 1, \ldots, m$, let Y_{ij} be the response variable for subject i at time point j, let $\mathbf{X}_{ij}$ be the vector of covariates subject to error, and $\mathbf{Z}_{ij}$ be the vector of covariates which are error-free. Write $\mathbf{Y}_i = (Y_{i1}, Y_{i2}, \ldots, Y_{im})'$, $\mathbf{X}_i = (\mathbf{X}'_{i1}, \mathbf{X}'_{i2}, \ldots, \mathbf{X}'_{im})'$, and $\mathbf{Z}_i = (\mathbf{Z}'_{i1}, \mathbf{Z}'_{i2}, \ldots, \mathbf{Z}'_{im})'$.

Response model

For $i = 1, \ldots, n$ and $j = 1, \ldots, m$, let $\mu_{ij} = \mathrm{E}(Y_{ij}|\mathbf{X}_i, \mathbf{Z}_i)$ and $v_{ij} = \mathrm{var}(Y_{ij}|\mathbf{X}_i, \mathbf{Z}_i)$ be the conditional expectation and variance of Y_{ij}, given the covariates $\mathbf{X}_i$ and $\mathbf{Z}_i$, respectively. We model the influence of the covariates on the marginal response mean by means of a regression model:

$$g(\mu_{ij}) = \mathbf{X}'_{ij}\boldsymbol{\beta}_x + \mathbf{Z}'_{ij}\boldsymbol{\beta}_z, \tag{F.2.1}$$

where $\boldsymbol{\beta} = (\boldsymbol{\beta}'_x, \boldsymbol{\beta}'_z)'$ is the vector of regression parameters and $g(\cdot)$ is a specified monotone function. The intercept term, if any, of the model may be included as the first element of $\boldsymbol{\beta}_z$ by including the unit vector as the first column of $\mathbf{Z}_i$.

To model the variance of Y_{ij}, we consider

$$v_{ij} = h(\mu_{ij}; \phi), \tag{F.2.2}$$

where $h(\cdot;\cdot)$ is a given function and ϕ is the dispersion parameter that is known or to be estimated. We treat ϕ as known with emphasis setting on estimation of the $\boldsymbol{\beta}$ parameter. Here we assume that $\mathrm{E}(Y^k_{ij}|\mathbf{X}_i, \mathbf{Z}_i) = \mathrm{E}(Y^k_{ij}|\mathbf{X}_{ij}, \mathbf{Z}_{ij})$ for $k = 1$ and 2, that is, the dependence of the mean μ_{ij} and the variance v_{ij} on the subject-level covariates $\mathbf{X}_i$ and $\mathbf{Z}_i$ is completely reflected by the dependence on the time-specific covariates $\mathbf{X}_{ij}$ and $\mathbf{Z}_{ij}$. This assumption has been widely used in marginal analysis of longitudinal analysis (e. g. , Diggle and Kenward, 1994; Lai and Small, 2007). The necessity of these assumptions was discussed by Yi (2017, Section 5.1.1).

Missing data model

For $i = 1, \ldots, n$ and $j = 1, \ldots, m$, let O_{ij} be 1 if Y_{ij} is observed and 0 otherwise, and let $\mathbf{O}_i = (O_{i1}, O_{i2}, \ldots, O_{im})'$ be the vector of missing data indicators. Dropouts or monotone missing data patterns are considered here. That is, $O_{ij} = 0$ implies $O_{ij'} = 0$ for all $j' > j$. We assume that $O_{i1} = 1$ for every subject i. To reflect the dynamic nature of the observation process over time, we assume an MAR mechanism for the missing process. That is, given the covariates, the missingness probability depends on the observed responses but not unobserved response components (Little and Rubin, 2002). Let $\lambda_{ij} = P(O_{ij} = 1|O_{i,j-1} = 1, \mathbf{X}_i, \mathbf{Z}_i, \mathbf{Y}_i)$ and $\pi_{ij} = P(O_{ij} = 1|\mathbf{X}_i, \mathbf{Z}_i, \mathbf{Y}_i)$, then

$$\pi_{ij} = \prod_{t=2}^{j} \lambda_{it}. \tag{F.2.3}$$

Logistic regression models are used to model the dropout process:

$$\mathrm{logit}(\lambda_{ij}) = \mathbf{u}'_{ij}\boldsymbol{\alpha}, \tag{F.2.4}$$

for $j = 2, \ldots, m$, where $\mathbf{u}_{ij}$ is the vector consisting of the information of the covariates $\mathbf{X}_i$, $\mathbf{Z}_i$ and the observed responses, and $\boldsymbol{\alpha}$ is the vector of regression parameters. Write $\boldsymbol{\theta} = (\boldsymbol{\alpha}', \boldsymbol{\beta}')'$ and let $q = \dim(\boldsymbol{\theta})$.

Measurement error model

For $i = 1, \ldots, n$ and $j = 1, \ldots, m$, let $\mathbf{W}_{ij}$ be the observed measurements of the covariates $\mathbf{X}_{ij}$. Covariates $\mathbf{X}_{ij}$ and their observed measurements $\mathbf{W}_{ij}$ are assumed to follow a classical additive measurement error model:

$$\mathbf{W}_{ij} = \mathbf{X}_{ij} + \mathbf{e}_{ij}, \tag{F.2.5}$$

where the $\mathbf{e}_{ij}$ are independent of $\mathbf{X}_i$, $\mathbf{Z}_i$ and $\mathbf{Y}_i$. And $\mathbf{e}_{ij}$ follows $\mathrm{N}(\mathbf{0}, \mathbf{\Sigma}_e)$ with the covariance matrix $\mathbf{\Sigma}_e$. This model has been widely used in the context of handling measurement error problems. Yi (2008) assumed that $\mathbf{\Sigma}_e$ is known or can be estimated from replication experiments (e. g. , Carroll et al., 2006; Yi, 2017).

Methodology

Weighted estimation function

The inverse probability weighted generalized estimating equations method is often employed to accommodate the missing data effects (e. g. , Robins et al., 1995; Preisser et al., 2002; Qu et al., 2011) when primary interest lies in the estimation of the marginal mean parameters $\boldsymbol{\beta}$ in the model (1). For $i = 1, \ldots, n$, let M_i be the random dropout time for subject i and m_i be a realization. Define $L_i(\boldsymbol{\alpha}) = (1 - \lambda_{im_i}) \prod_{t=2}^{m_i-1} \lambda_{it}$, where λ_{it} is determined by model (4). Let $\mathbf{S}_i(\boldsymbol{\alpha}) = \partial logL_i(\boldsymbol{\alpha})/\partial\boldsymbol{\alpha}$ be the vector of score functions contributed from subject i. Let $\mathbf{D}_i = \partial\boldsymbol{\mu}_i'/\partial\boldsymbol{\beta}$ be the matrix of the derivatives of the mean vector $\boldsymbol{\mu}_i = (\mu_{i1}, \ldots, \mu_{im})'$ with respect to $\boldsymbol{\beta}$ and let $\mathbf{\Delta}_i = \mathrm{diag}(I(O_{ij} = 1)/\pi_{ij}, j = 1, 2, \ldots, m)$ be the weighted matrix accommodating missingness, where $I(\cdot)$ is the indicator function. Let $\mathbf{V}_i = \mathbf{A}_i^{1/2}\mathbf{C}_i\mathbf{A}_i^{1/2}$ be the conditional covariance matrix of $\mathbf{Y}_i$, given $\mathbf{X}_i$ and $\mathbf{Z}_i$, where $\mathbf{A}_i = \mathrm{diag}(v_{ij}, j = 1, 2, \ldots, m)$ and $\mathbf{C}_i = [\rho_{i;jk}]$ is the correlation matrix with diagonal elements equal 1 and $\rho_{i;jk}$ being the conditional correlation coefficient of response components Y_{ij} and Y_{ik} for $j \neq k$, given $\mathbf{X}_i$ and $\mathbf{Z}_i$. Define

$$\mathbf{U}_i(\boldsymbol{\theta}) = \mathbf{D}_i\mathbf{V}_i^{-1}\mathbf{\Delta}_i(\mathbf{Y}_i - \boldsymbol{\mu}_i)$$

and

$$\mathbf{H}_i(\boldsymbol{\theta}) = (\mathbf{U}_i'(\boldsymbol{\theta}), \mathbf{S}_i'(\boldsymbol{\alpha}))'. \tag{F.3.1}$$

In the absence of measurement error, that is, covariates $\mathbf{X}_{ij}$ are precisely observed, we have $\mathrm{E}[\mathbf{H}_i(\boldsymbol{\theta})] = \mathbf{0}$. Hence, $\mathbf{H}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \mathbf{H}_i(\boldsymbol{\theta})$ are unbiased estimation functions for $\boldsymbol{\theta}$ (e. g. , Yi, 2017, Chapter 1). Under regularity conditions, the consistent estimator $\widehat{\boldsymbol{\theta}}$ of $\boldsymbol{\theta}$ can be obtained by solving

$$\mathbf{H}(\boldsymbol{\theta}) = \mathbf{0}, \tag{F.3.2}$$

where the weight matrix $\mathbf{\Delta}_i$ is used to adjust for the contributions of subject i with his/her missingness probabilities incorporated. Specifically, the probability π_{ij} is determined by (3) in conjunction with (4). Correlation matrix $\mathbf{C}_i$ can be replaced by the moment estimate, or alternatively, a working independence matrix $\mathbf{A}_i$ may be used to replace $\mathbf{V}_i$ (Liang and Zeger, 1986). A detail discussion can be found in Yi (2017, Chapter 4).

SIMEX approach

When measurement error is present in covariates $\mathbf{X}_{ij}$, $\mathbf{H}(\boldsymbol{\theta})$ is no longer unbiased if naively replacing $\mathbf{X}_{ij}$ with its observed measurement $\mathbf{W}_{ij}$. Yi (2008) developed a simulation-extrapolation (SIMEX) method to adjust for the bias induced by using $\mathbf{W}_{ij}$, as well as the missingness effects in the response variables. This method originates from the SIMEX method by Cook and Stefanski (1994) who considered cross-sectional data with measurement error alone. The basic idea of the SIMEX method is to first add additional variability to the observed measurement $\mathbf{W}_{ij}$, then establish the trend how different degrees of measurement error may induce bias in estimation of the model parameters, and finally extrapolate this trend to the case of no measurement error.

Now, we describe the SIMEX method developed by Yi (2008). Let B be a given positive integer and $\mathbf{\Lambda} = \{\lambda_1, \lambda_2, \ldots, \lambda_M\}$ be a sequence of nonnegative numbers taken from $[0, \lambda_M]$ with $\lambda_1 = 0$.

- Step 1: Simulation For $i = 1, \ldots, n$ and $j = 1, \ldots, m$, generate $\mathbf{e}_{ijb} \sim \mathrm{N}(\mathbf{0}, \mathbf{\Sigma}_e)$ for $b = 1, 2, \ldots, B$. For a given $\lambda \in \mathbf{\Lambda}$, set

$$\mathbf{W}_{ij}(b, \lambda) = \mathbf{W}_{ij} + \sqrt{\lambda}\mathbf{e}_{ijb}.$$

- Step 2: Estimation For given λ and b, we obtain an estimate $\widehat{\boldsymbol{\theta}}(b, \lambda)$ by solving equation (7) with $\mathbf{X}_{ij}$ replaced by $\mathbf{W}_{ij}(b, \lambda)$. Let $\widehat{\mathbf{\Gamma}}(b, \lambda) = \sum_{i=1}^{n} [\partial\mathbf{H}_i'(\boldsymbol{\theta}; b, \lambda)/\partial\boldsymbol{\theta}]|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}(b,\lambda)}$ and $\widehat{\mathbf{\Sigma}}(b, \lambda) =$

$\sum_{i=1}^{n}\left[\mathbf{H}_i(\boldsymbol{\theta};b,\lambda)\mathbf{H}_i'(\boldsymbol{\theta};b,\lambda)\right]|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}(b,\lambda)}$, then the covariance matrix of $\widehat{\boldsymbol{\theta}}(b,\lambda)$ is estimated by:

$$\widehat{\boldsymbol{\Omega}}(b,\lambda) = n\cdot\left\{[\widehat{\boldsymbol{\Gamma}}(b,\lambda)]^{-1}\cdot\widehat{\boldsymbol{\Sigma}}(b,\lambda)\cdot[\widehat{\boldsymbol{\Gamma}}(b,\lambda)]^{-1'}\right\}|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}(b,\lambda)}.$$

Let $\widehat{\theta}_r(b,\lambda)$ be the rth component of $\widehat{\boldsymbol{\theta}}(b,\lambda)$ and let $\widehat{\Omega}_r(b,\lambda)$ be the rth diagonal element of $\widehat{\boldsymbol{\Omega}}(b,\lambda)$ for $r=1,2,\ldots,q$. We then calculate the average of those estimates over b for each λ:

$$\begin{aligned}
\widehat{\theta}_r(\lambda) &= B^{-1}\sum_{b=1}^{B}\widehat{\theta}_r(b,\lambda);\\
\widehat{\Omega}_r(\lambda) &= B^{-1}\sum_{b=1}^{B}\widehat{\Omega}_r(b,\lambda);\\
\widehat{S}_r(\lambda) &= (B-1)^{-1}\sum_{b=1}^{B}(\widehat{\theta}_r(b,\lambda)-\widehat{\theta}_r(\lambda))^2;\\
\widehat{\tau}_r(\lambda) &= \widehat{\Omega}_r(\lambda)-\widehat{S}_r(\lambda).
\end{aligned}$$

- Step 3: Extrapolation For $r=1,2,\ldots,q$, fit a regression model to each of the sequences $\{(\lambda,\widehat{\theta}_r(\lambda)) : \lambda\in\boldsymbol{\Lambda}\}$ and $\{(\lambda,\widehat{\tau}_r(\lambda)) : \lambda\in\boldsymbol{\Lambda}\}$, respectively, and extrapolate it to $\lambda=-1$, let $\widehat{\theta}_r(-1)$ and $\widehat{\tau}_r(-1)$ denote the resulting predicted values. Then, $\widehat{\boldsymbol{\theta}}=(\widehat{\theta}_1,\widehat{\theta}_2,\ldots,\widehat{\theta}_q)'$ is the SIMEX estimator of $\boldsymbol{\theta}$ and $\sqrt{\widehat{\tau}_r}$ is the associated standard error for the estimator $\widehat{\theta}_r$ for $r=1,2,\ldots,q$.

The SIMEX approach is very appealing because of its simplicity of implementation and no requirement of modeling the true covariates $\mathbf{X}_i$. However, to use this method, several aspects need to be considered. As suggested by Carroll et al. (2006), the specification of Λ is not unique; a typical choice of grid Λ is the equal cut points of interval $[0,2]$ with $M=5$ or 9. Choosing $B=100$ or 200 is often sufficient for many applications. The quadratic regression function is commonly used for Step 3 to yield reasonable results. (e. g. , He et al., 2012).

Finally, we extend the method by Yi (2008) to accommodating the case where the covariance matrix $\boldsymbol{\Sigma}_e$ for model (5) is unknown but repeated surrogate measurements of $\mathbf{X}_{ij}$ are available. Let $\mathbf{W}_{ijk}$ denote the repeated surrogate measurements of $\mathbf{X}_{ij}$ for $i=1,\ldots,n; j=1,\ldots,m$; and $k=1,\ldots,K$. The surrogate measurements $\mathbf{W}_{ijk}$ and the true covariate $\mathbf{X}_{ij}$ are linked by the model

$$\mathbf{W}_{ijk}=\mathbf{X}_{ij}+\mathbf{e}_{ijk}, \tag{F.3.3}$$

where the $\mathbf{e}_{ijk}$ are independent of $\mathbf{X}_i$, $\mathbf{Z}_i$ and $\mathbf{Y}_i$, and $\mathbf{e}_{ijk}$ follows $\mathrm{N}(\mathbf{0},\boldsymbol{\Sigma}_e)$ with the covariance matrix $\boldsymbol{\Sigma}_e$. We now adapt the arguments of Devanarayan and Stefanski (2002) to modify the simulation step of the preceding SIMEX method. For a given b and $\lambda\in\boldsymbol{\Lambda}$, set

$$\mathbf{W}_{ij}(b,\lambda)=\overline{\mathbf{W}}_{ij}+\sqrt{\lambda/K}\sum_{k=1}^{K}c_{ijk}(b)\mathbf{W}_{ijk}, \tag{F.3.4}$$

where $\overline{\mathbf{W}}_{ij}=K^{-1}\sum_{k=1}^{K}\mathbf{W}_{ijk}$ and $\mathbf{c}_{ij}(b)=(c_{ij1}(b),\ldots,c_{ijk}(b))'$ is a normalized contrast satisfying $\sum_{k=1}^{K}c_{ijk}=0$ and $\sum_{k=1}^{K}c_{ijk}^2=1$.

A simple way to generate a contrast $\mathbf{c}_{ij}(b)$ can be done by independently generating K variates, $d_{ijk}(b)$, from $\mathrm{N}(0,1)$ for $k=1,\ldots,K$ and a given b. Let $\bar{d}_{ij}(b)=K^{-1}\sum_{k=1}^{K}d_{ijk}(b)$. Then $c_{ijk}(b)$ is set as

$$c_{ijk}(b,\lambda)=\frac{d_{ijk}(b)-\bar{d}_{ij}(b)}{\sqrt{\sum_{k=1}^{K}\{d_{ijk}(b)-\bar{d}_{ij}(b)\}^2}}.$$

Once $\mathbf{W}_{ij}(b,\lambda)$ of (9) is available, we repeat Steps 2 and 3 to obtain the SIMEX estimator and the associated standard error.

Implementation in R

We implement the SIMEX procedure described in Section Methodology in R and develop the package, called **swgee**. Our package **swgee** takes the advantage of existing R packages **geepack** (Hojsgaard et al., 2016) and **mvtnorm** (Genz and Bretz, 2009; Genz et al., 2018). Specifically, the function `swgee` produces the estimates for elements of the parameter vector $\boldsymbol{\beta}$, which are of primary interest, the associated standard errors, and P-values.

Our R function `swgee` requires the input data set to be sorted by subject i and visit time j for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. If a subject is missing at a certain time, the corresponding measurements should be recorded as NAs. As long as the user provides the missing data model (4), the function `swgee` can internally generate the missing data indicators O_{ij} for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, and then apply the user specified model (4) to fit the data. The missingness probabilities π_{ij} are calculated by (3) and then used to construct the weight matrix $\mathbf{\Delta}_i$ for the estimating equation (6). The estimate of the missing data model (4) parameter $\boldsymbol{\alpha}$ can also be retrieved from the function `swgee` output.

The form of calling function `swgee` is given by

```
swgee(formula, data, id, family, corstr, missingmodel, SIMEXvariable,
    SIMEX.err, repeated = FALSE, repind = NULL, B, lambda)
```

where the arguments are described as follows:

- `formula`: This argument specifies the model to be fitted, with the variables coming with data. See the documentation of `geeglm` and its `formula` for details.
- `data`: This is the same as the data argument in the R function `geeglm`, which specifies the data frame showing how variables occur in the formula, along with the id variable.
- `id`: This is the vector which identifies the labels of subjects. i.e., the id for subject i is i, using the notation of Section Response model, where $i = 1, 2, \ldots, n$. Data are arranged so that observations for the same subject are listed in consecutive rows in order of time, and consequently, the id for a subject would repeat the same number of times as the observation times.
- `family`: This argument describes the error distribution together with the link function for model (1). See the documentation of `geeglm` and its argument `family` for details.
- `corstr`: This is a character string specifying the correlation structure. See the documentation of `geeglm` and its argument `corstr` for details.
- `missingmodel`: This argument specifies the formula to be fitted for the missing data model (4). See the documentation of `geeglm` and its `formula` for details.
- `SIMEXvariable`: This is the vector of characters containing the names of the covariates which are subject to measurement error.
- `SIMEX.err`: This argument specifies the covariance matrix of measurement errors in the measurement error model (5).
- `repeated`: This is the indicator whether measurement error model is given by (5) or by (8). The default value FALSE corresponding to model (5).
- `repind`: This is the index of the repeated surrogate measurements $\mathbf{W}_{ijk}$ for each covariate $\mathbf{X}_{ij}$. It has an R list form. If repeated = TRUE, repind must be specified.
- `B`: This argument sets the number of simulated samples for the simulation step. The default is set to be 50.
- `lambda`: This is the vector $\{\lambda_1, \lambda_2, \ldots, \lambda_M\}$ we describe in Step 1 of Section SIMEX approach. Its values need to be specified by the user.

Examples

An example data set

To illustrate the usage of the developed R package **swgee**, we apply the package to a subset of GWA13 (Genetic Analysis Workshops) data arising from the Framingham Heart Study. The data set consists of measurements of 100 patients from a series of exams with 5 assessments for each individual. Measurements such as height, weight, age, systolic blood pressure (SBP) and cholesterol level (CHOL) are collected at each assessment, and 14% patients dropped out of the study. The original data were analyzed by Yi (2008). It is of interest to study how an individual's obesity may change with age (Z_{ij}) and how it is associated with SBP (X_{ij1}) and CHOL (X_{ij2}), where $i = 1, \ldots, 100$, and $j = 1, \ldots, 5$. The response Y_i is the indicator of obesity status of subject i as in Yi (2008); SBP is rescaled as log(SBP − 50) as in Carroll et al. (2006); and CHOL is standardized. The response and the covariates are postulated by the logistic regression model:

$$\text{logit}\ \mu_{ij} = \beta_0 + \beta_{x1} X_{ij1} + \beta_{x2} X_{ij2} + \beta_z Z_{ij},$$

where β_0, β_{x1}, β_{x2} and β_z are regression coefficients of interest. We assume that errors in both risk factors X_{ij1} and X_{ij2} can be represented by model (5). The missing data process is characterized by the logistic regression model:

$$\text{logit}\lambda_{ij} = \alpha_1 + \alpha_2 Y_{i,j-1} + \alpha_3 X_{i,j-1,1} + \alpha_4 X_{i,j-1,2} + \alpha_{5c} z_{i,j-1},$$

for $j = 2, \ldots, 5$.

We now apply the developed R package **swgee**, which can be downloaded from CRAN and then loaded in R:

```
R> library("swgee")
```

Next, load the data that are properly organized with the variable names specified. In the example here, the data set, named as bmidata, is included by issuing

```
R> data("BMI")
R> bmidata <- BMI
```

We are concerned how measurement error in SBP and CHOL impacts estimation of parameter $\boldsymbol{\beta} = (\beta_0, \beta_{x1}, \beta_{x2}, \beta_z)'$. For illustrative purposes, we use setting with $B = 100$, $\lambda_M = 2$ and $M = 5$. In this example, we assume that parameters in $\boldsymbol{\Sigma}_e = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix}$ with $\sigma_{12} = \sigma_{21}$ are known. This is a typical case when conducting sensitivity analysis. Here we set $\sigma_1 = \sigma_2 = 0.5$ and $\sigma_{12} = \sigma_{21} = 0$ as an example.

The naive GEE approach without considering missingness and measurement error effects in covariates gives the output:

```
R> output1 <- gee(bbmi~sbp+chol+age, id=id, data=bmidata,
+      family=binomial(link="logit"), corstr="independence")

R> summary(output1)

 GEE:  GENERALIZED LINEAR MODELS FOR DEPENDENT DATA
 gee S-function, version 4.13 modified 98/01/27 (1998)

Model:
 Link:                      Logit
 Variance to Mean Relation: Binomial
 Correlation Structure:     Independent

Call:
gee(formula = bbmi ~ sbp + chol + age, id = id, data = bmidata,
    family = binomial(link = "logit"), corstr = "independence")

Summary of Residuals:
        Min          1Q      Median          3Q         Max
-0.26533967 -0.11385369 -0.08572483 -0.06279540  0.95475735

Coefficients:
               Estimate Naive S.E.    Naive z Robust S.E.   Robust z
(Intercept) -5.43746374 1.42090827 -3.8267521  1.64320527 -3.3090593
sbp          0.59071183 0.30643396  1.9276970  0.24338420  2.4270755
chol         0.11109496 0.13654324  0.8136247  0.23086218  0.4812177
age          0.01297337 0.01339946  0.9682008  0.01814546  0.7149652

Estimated Scale Parameter:  1.017131
Number of Iterations:  1
Working Correlation
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
```

```
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

To adjust for possible effects of missingness as well as measurement error in variables SBP and CHOL, we call the developed function `swgee` for the analysis:

```
R> set.seed(1000)
R> sigma <- diag(rep(0.25, 2))
R> output2 <- swgee(bbmi~sbp+chol+age, data=bmidata, id=id,
+     family=binomial(link="logit"), corstr="independence",
+     missingmodel=O~bbmi+sbp+chol+age, SIMEXvariable=c("sbp","chol"),
+     SIMEX.err=sigma, repeated=FALSE, B=100, lambda=seq(0, 2, 0.5))

> summary(output2)
Call: beta
             Estimate    StdErr t.value   p.value
(Intercept) -8.004577  2.060967 -3.8839 0.0001028 ***
sbp          1.196363  0.356868  3.3524 0.0008011 ***
chol         0.099984  0.264180  0.3785 0.7050810
age          0.012718  0.017201  0.7394 0.4596520
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1  1

Call: alpha
        Estimate    StdErr t.value  p.value
alpha1  9.019084  3.086533  2.9221 0.003477 **
alpha2 -0.786135  0.656843 -1.1968 0.231370
alpha3 -0.568740  0.732885 -0.7760 0.437732
alpha4 -0.128941  0.247757 -0.5204 0.602761
alpha5 -0.064257  0.025982 -2.4731 0.013395 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1  1
```

The function `swgee` can store individual estimated coefficients in the simulation step, and this enables us to show the extrapolation curve through the developed R function `plot.swgee`. The `plot.swgee` function plots the extrapolation of the estimate of each covariate effect with the quadratic extrapolants. Figure 1 displays the graph for the variable SBP in the example for which the quadratic extrapolation function is applied from the following command:

```
R> plot(output2,"sbp")
```

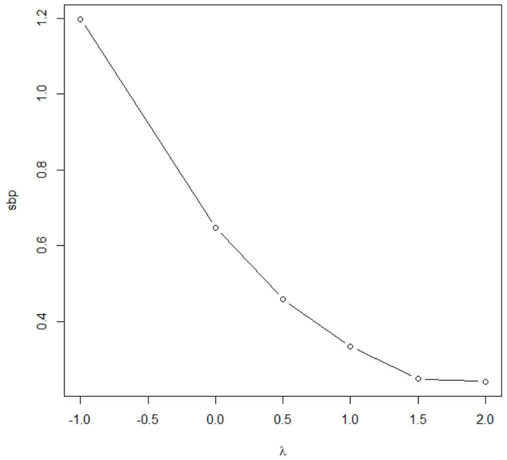


Figure 1: Display of the SIMEX estimate for the example: the dot is the SIMEX estimate obtained from the quadratic extrapolation.

Simulation studies

In this section, we conduct simulation studies to investigate the impact of ignoring covariate measurement error and response missingness on estimation, where the implementation is carried out using the usual GEE method. Furthermore, we assess the performance of the swgee method which accommodates the effects induces from error-prone covariates and missing responses. We set $n = 200$ and $m = 3$, and generate 500 simulations for each parameter configuration. Consider the logistic regression model

$$\text{logit}(\mu_{ij}) = \beta_0 + \beta_{x1}x_{ij1} + \beta_{x2}x_{ij2} + \beta_z z_{ij}, \tag{F.5.1}$$

where $\beta_0 = 0$, $\beta_{x1} = \log(1.5)$, $\beta_{x2} = \log(1.5)$, $\beta_z = \log(0.75)$ and z_{ij} is generated independently from $\text{Bin}(1, 0.5)$ to represent a balanced design. The true covariate $\mathbf{X}_{ij} = (x_{ij1}, x_{ij2})'$ is generated from the normal distribution $\text{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$, where $\boldsymbol{\mu}_x = (0.5, 0.5)'$ and $\boldsymbol{\Sigma}_x = \begin{pmatrix} \sigma_{x1}^2 & \rho_x\sigma_{x1}\sigma_{x2} \\ \rho_x\sigma_{x1}\sigma_{x2} & \sigma_{x2}^2 \end{pmatrix}$ with $\sigma_{x1} = \sigma_{x2} = 1$. The surrogate value $\mathbf{W}_{ij} = (W_{ij1}, W_{ij2})'$ is generated from $\text{N}(\mathbf{X}_{ij}, \boldsymbol{\Sigma}_e)$ with $\boldsymbol{\Sigma}_e = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$. ρ and ρ_x are set to 0.50 to represent moderate correlations. To feature minor, moderate and severe degrees of measurement error, we consider $\sigma_1, \sigma_2 = 0.25$, 0.50 or 0.75. The missing data indicator is generated from model (4), where $\alpha_0 = \alpha_1 = 0.5$, $\alpha_2 = \alpha_3 = 0.1$, and $\alpha_z = 0.2$. In implementing the swgee method, we choose $B = 100$, $\lambda_M = 2$, $M = 5$, and a quadratic regression for each extrapolation step.

In Table 1, we report on the results of the biases of the estimates (Bias), the empirical standard error (SE), and the coverage rate (CR in percent) for 95% confidence intervals. When measurement error is minor, (i.e. $\sigma_1 = \sigma_2 = 0.25$), both gee and swgee provide reasonable results with fairly small finite sample biases and coverage rates close to the nominal level 95%. When there is moderate or substantial measurement error in covariates $\mathbf{X}_{ij}$, the performance of the gee method deteriorates remarkably in estimation of error-prone covariate effects, leading to considerably biased estimates for β_{x1} and β_{x2}. The corresponding coverage rates for 95% confidence intervals can be quite low. In contrast, the swgee method remarkably improve the performance, providing a lot smaller biases and much higher coverage rates. The estimates for β_z are not subject to much impact of measurement error, which is partially attributed by that the precisely observed covariates z_{ij} are generated independently of error-prone covairates $\mathbf{X}_{ij}$ under the current simulation study.

In summary, ignoring measurement error may lead to substantially biased results. Properly addressing covariate measurement error in estimation procedures is necessary. The proposed swgee method performs reasonably well under various configurations. As expected, its performance may become less satisfactory when measurement error becomes substantial. However, the swgee method does significantly improve the performance of the gee analysis.

Summary and discussion

Missing observations and covariate measurement error commonly arise in longitudinal data. However, there has been relatively little work on simultaneously accounting for the effects of response missingness and covariate measurement error on estimation of response model parameters for longitudinal data. Yi (2008) described a simulation based marginal method to adjust for the biases induced by both missingness and covariate measurement error. The proposed method does not require the full specification of the distribution of the response vector but only requires modeling its mean and covariance structure. In addition, the distribution of covariates is left unspecified, which is desirable for many practical problems. These features make the proposed method flexible.

Here we not only develop the R package **swgee** to implement the method by Yi (2008), but also include an extended setting in the package. Our aim is to provide analysts an accessible tool for the analysis of longitudinal data with missing responses and error-prone covariates. Our illustrations show that the developed package has the advantages of simplicity and versatility.

σ_1	σ_2	Method	β_{x1}			β_{x2}			β_z		
			Bias	SE	CR	Bias	SE	CR	Bias	SE	CR
0.25	0.25	gee	-0.0310	0.1228	92.6	-0.0158	0.1246	92.6	0.0063	0.2121	94.6
0.25	0.25	swgee	-0.0062	0.1420	95.0	0.0104	0.1425	95.2	0.0036	0.2354	95.6
0.25	0.50	gee	-0.0019	0.1212	95.4	-0.0997	0.1156	83.4	0.0082	0.2110	94.2
0.25	0.50	swgee	-0.0003	0.1415	95.0	-0.0087	0.1543	93.0	0.0035	0.2361	95.6
0.25	0.75	gee	0.0328	0.1189	95.4	-0.1841	0.1022	51.0	0.0101	0.2100	94.0
0.25	0.75	swgee	0.0205	0.1407	95.8	-0.0660	0.1562	86.4	0.0046	0.2359	95.6
0.50	0.25	gee	-0.1156	0.1114	78.2	0.0139	0.1236	94.2	0.0078	0.2113	94.6
0.50	0.25	swgee	-0.0282	0.1520	93.2	0.0177	0.1431	95.4	0.0031	0.2362	95.2
0.50	0.50	gee	-0.0948	0.1114	81.8	-0.0780	0.1161	85.6	0.0102	0.2099	94.2
0.50	0.50	swgee	-0.0228	0.1510	93.8	-0.0022	0.1542	93.6	0.0030	0.2370	95.4
0.50	0.75	gee	-0.0629	0.1103	87.8	-0.1727	0.1036	55.6	0.0125	0.2088	94.2
0.50	0.75	swgee	-0.0052	0.1499	94.8	-0.0608	0.1570	87.2	0.0042	0.2369	95.2
0.75	0.25	gee	-0.1991	0.0966	45.6	0.0484	0.1216	94.2	0.0092	0.2107	94.6
0.75	0.25	swgee	-0.0870	0.1508	86.4	0.0395	0.1430	93.6	0.0034	0.2366	95.2
0.75	0.50	gee	-0.1889	0.0976	50.0	-0.0458	0.1154	89.8	0.0121	0.2091	94.0
0.75	0.50	swgee	-0.0831	0.1509	87.8	0.0165	0.1539	94.0	0.0034	0.2375	95.4
0.75	0.75	gee	-0.1636	0.0974	58.8	-0.1468	0.1039	66.4	0.0147	0.2077	94.2
0.75	0.75	swgee	-0.0678	0.1505	90.0	-0.0442	0.1574	88.8	0.0046	0.2374	95.2

Table 1: Simulation Results

Acknowledgments

Juan Xiong was supported by the Natural Science Foundation of SZU (grant no.2017094). Grace Y. Yi was supported by the Natural Sciences and Engineering Research Council of Canada. The authors thanks Boston University and the National Heart, Lung, and Blood Institute (NHLBI) for providing the data set from the Framingham Heart Study (No. N01-HC-25195) in the illustration. The Framingham Heart Study is conducted and supported by the NHLBI in collaboration with

Boston University. This manuscript was not prepared in collaboration with investigators of the Framingham Heart Study and does not necessarily reflect the opinions or views of the Framingham Heart Study, Boston University, or NHLBI.

Conflict of Interest: None declared.

Bibliography

J. P. Buonaccorsi. *Measurement Error: Models, Methods, and Applications*. Chapman & Hall/CRC, Boca Raton, Florida, 2010. [p416]

V. J. Carey. **yags***: Yet Another GEE Solve*, 2011. R package version 6.1-13. [p416]

V. J. Carey. **gee***: Generalized Estimation Equation Solver*, 2015. R package version 4.13-19. [p416]

R. J. Carroll, D. Ruppert, L. A. Stefanski, and C. M. Crainiceanu. *Measurement Error in Nonlinear Models: A Modern Perspective*. Chapman & Hall/CRC, Boca Raton, Florida, 2nd edition, 2006. [p416, 418, 419, 420]

J. R. Cook and L. A. Stefanski. Simulation-extrapolation estimation in parametric measurement error models. *Journal of the American Statistical Association*, 89(428):1314–1328, 1994. URL https://doi.org/10.1080/01621459.1994.10476871. [p418]

V. Devanarayan and L. A. Stefanski. Empirical simulation extrapolation for measurement error models with replicate measurements. *Statistics and Probability Letters*, 59(3):219–225, 2002. URL https://doi.org/10.1016/S0167-7152(02)00098-6. [p419]

P. J. Diggle and M. G. Kenward. Informative drop-out in longitudinal data analysis (with discussion). *Applied Statistics*, 43(1):49–93, 1994. URL https://doi.org/10.2307/2986113. [p417]

W. A. Fuller. *Measurement Error Models*. John Wiley & Sons, New York, 1987. [p416]

A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Springer-Verlag, New York, 2009. [p419]

A. Genz, F. Bretz, T. Miwa, X. Mi, and T. Hothorn. **mvtnorm***: Multivariate Normal and t Distributions*, 2018. R package version 1.0-7. [p419]

P. Gustafson. *Measurement Error and Misclassification in Statistics and Epidemiology*. Chapman & Hall/CRC, Boca Raton, Florida, 2003. [p416]

W. He, J. Xiong, and G. Y. Yi. Simex R package for accelerated failure time models with covariate measurement error. *Journal of Statistical Software*, 46(1):1–14, 2012. URL https://doi.org/10.18637/jss.v046.c01. [p419]

S. Hojsgaard, U. Halekoh, and J. Yan. **geepack***: Generalized Estimating Equation Package*, 2016. R package version 1.2-1. [p416, 419]

M. G. Kenward. Selection models for repeated measurements with non-random dropout: An illustration of sensitivity. *Statistics in Medicine*, 17(23):2723–2732, 1998. URL https://doi.org/10.1002/(SICI)1097-0258(19981215)17:23<2723::AID-SIM38>3.0.CO;2-5. [p416]

T. L. Lai and D. S. Small. Marginal regression analysis of longitudinal data with time-dependent covariates: A generalized method-of-moments approach. *Journal of The Royal Statistical Society Series B-statistical Methodology*, 69(1):79–99, 2007. URL https://doi.org/10.1111/j.1467-9868.2007.00578.x. [p417]

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. [p418]

G. Lin and R. N. Rodriguez. Weighted methods for analyzing missing data with the gee procedure. *Paper SAS166-2015*, pages 1–8, 2015. [p416]

R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, New Jersey, 2nd edition, 2002. [p416, 417]

W. Liu and L. Wu. Simultaneous inference for semiparametric nonlinear mixed-effects models with covariate measurement errors and missing responses. *Biometrics*, 63(2):342–350, 2007. URL https://doi.org/10.1111/j.1541-0420.2006.00687.x. [p416]

J. S. Preisser, K. K. Lohman, and P. J. Rathouz. Performance of weighted estimating equations for longitudinal binary data with drop-outs missing at random. *Statistics in Medicine*, 21(20): 3035–3054, 2002. URL https://doi.org/10.1002/sim.1241. [p418]

A. Qu, G. Y. Yi, P. X. K. Song, and P. Wang. Assessing the validity of weighted generalized estimating equations. *Biometrika*, 98(1):215–224, 2011. URL https://doi.org/10.1093/biomet/asq078. [p418]

J. M. Robins, A. Rotnitzky, and L. Zhao. Analysis of semiparametric regression models for repeated outcomes in the presence of missing data. *Journal of the American Statistical Association*, 90 (429):106–121, 1995. URL https://doi.org/10.1080/01621459.1995.10476493. [p418]

SAS Institute Inc. *SAS/STAT Software, Version 13.2.* Cary, NC, 2014. URL http://www.sas.com/. [p416]

C. Y. Wang, Y. Huang, E. C. Chao, and M. K. Jeffcoat. Expected estimating equations for missing data, measurement error, and misclassification, with application to longitudinal nonignorable missing data. *Biometrics*, 64(1):85–95, 2008. URL https://doi.org/10.1111/j.1541-0420.2007.00839.x. [p416]

C. Xu, Z. Li, and M. Wang. **wgeesel***: Weighted Generalized Estimating Equations and Model Selection*, 2018. R package version 1.5. [p416]

G. Y. Yi. A simulation-based marginal method for longitudinal data with dropout and mismeasured covariates. *Biostatistics*, 9(3):501–512, 2008. URL https://doi.org/10.1093/biostatistics/kxm054. [p416, 417, 418, 419, 420, 423]

G. Y. Yi. *Statistical Analysis with Measurement Error or Misclassification.* Springer-Verlag, New York, 2017. [p416, 417, 418]

G. Y. Yi, Y. Ma, and R. J. Carroll. A functional generalized method of moments approach for longitudinal studies with missing responses and covariate measurement error. *Biometrika*, 99(1): 151–165, 2012. URL https://doi.org/10.1093/biomet/asr076. [p416]

Juan Xiong
Department of Preventive Medicine
School of Medicine
Shenzhen University
3688 Nanhai Avenue, Shenzhen, China 518060
jxiong@szu.edu.cn

Grace Y. Yi
Department of Statistics and Actuarial Science
University of Waterloo
200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1
yyi@uwaterloo.ca

unival: An FA-based R Package For Assessing Essential Unidimensionality Using External Validity Information

by Pere J. Ferrando, Urbano Lorenzo-Seva and David Navarro-Gonzalez

Abstract

The **unival** package is designed to help researchers decide between unidimensional and correlated-factors solutions in the factor analysis of psychometric measures. The novelty of the approach is its use of *external* information, in which multiple factor scores and general factor scores are related to relevant external variables or criteria. The **unival** package's implementation comes from a series of procedures put forward by Ferrando and Lorenzo-Seva (2019) and new methodological developments proposed in this article. We assess models fitted using **unival** by means of a simulation study extending the results obtained in the original proposal. Its usefulness is also assessed through a real-world data example. Based on these results, we conclude **unival** is a valuable tool for use in applications in which the dimensionality of an item set is to be assessed.

Introduction

Assessing the dimensionality of a set of items is one of the central purposes of psychometric factor analysis (FA) applications. At present, both the exploratory (EFA) and the confirmatory (CFA) models can be considered to be fully developed structural equation models (Ferrando and Lorenzo-Seva, 2017). So, in principle, dimensionality can be rigorously assessed by using the wide array of goodness-of-fit procedures available for structural models in general. However, it is becoming increasingly clear that reliance on goodness-of-fit alone is not the way to judge the most appropriate dimensionality for studying a particular set of item scores (Rodriguez et al., 2016a,b).

The problem noted above is particularly noticeable in instruments designed to measure a single trait. In the vast majority of cases, item scores derived from these instruments fail to meet the strict unidimensionality criteria required by Spearman's model. This failure, in turn, led to the proposal of multiple correlated-factor solutions as the most appropriate structure for them (Ferrando and Lorenzo-Seva, 2018, in press; Furnham, 1990; Reise et al., 2013, 2015). However, most instruments designed to be unidimensional do, in fact, yield data compatible with an essentially unidimensional solution (Floyd and Widaman, 1995; Reise et al., 2013, 2015). When this is the case, treating the item scores as multidimensional has several undesirable consequences, mainly, (a) lack of clarity in the interpretation and unnecessary theoretical complexities, and (b) weakened factor score estimates that do not allow accurate individual measurement (Ferrando and Lorenzo-Seva, 2018, in press; Furnham, 1990; Reise et al., 2013, 2015). Indeed, treating clearly multidimensional scores as unidimensional also has such negative consequences as biased item parameter estimates, loss of information, and factor score estimates that cannot be univocally interpreted (see Ferrando and Lorenzo-Seva, 2018; Reise et al., 2013).

In recent years, several indices and criteria have been proposed to assess dimensionality using different perspectives of model appropriateness. These developments, in turn, have been integrated in comprehensive proposals addressing the dimensionality issue from multi-faceted views including, but are not limited to, standard goodness-of-fit results (Ferrando and Lorenzo-Seva, 2018; Raykov and Marcoulides, 2018; Rodriguez et al., 2016a,b). It is worth noting these approaches generally reflect a trend in which the measurement part of the FA model is again relevant (e.g. Curran et al., 2018). Considering the ultimate aim of most psychometric measures is individual measurement, the scoring stage of the FA should be expected to be the most important part of it (Ferrando and Lorenzo-Seva, 2018, in press). Furthermore, if this view is adopted, a basic criterion for deciding if a given FA solution is appropriate is the extent to which the score estimates derived from this solution are strong, reliable, determinate, unbiased, and clearly interpretable (Ferrando and Lorenzo-Seva, 2018; Beauducel et al., 2016; Furnham, 1990; Reise et al., 2013, 2015). Procedures explicitly based on the quality of the score estimates are already available in widely used programs such as FACTOR (Lorenzo-Seva and Ferrando, 2013), and more sophisticated procedures based on Haberman's (2008) added-value principle have been also proposed (Ferrando and Lorenzo-Seva, in press).

A common characteristic of all the proposals discussed so far is their use of *internal* information from the data exclusively: that is to say, the information provided by the item scores of the measure under study. In contrast, the approach implemented here is based on *external* sources of information: that is to say, the information provided by the relations between the factor score estimates derived from a given solution and relevant external variables or criteria. This additional information is a valuable complementary tool that can help reach a decision on whether the instrument under scrutiny is essentially unidimensional or truly multidimensional.

The present article aims to introduce **unival**, a new contributed R package implementing a recently proposed external procedure of the type described above (Ferrando and Lorenzo-Seva, 2019). It also discusses new methodological developments allowing the procedure to be used in a wider range of situations than those considered in the original proposal. The rest of the article is organized as follows. First, we provide a summary the needed theoretical bases, and explain the new methodological contributions. Then, we give details about the package and how to use it. Finally, we assess the functioning of the program and the new developments proposed here with a simulation study and give real-world data examples.

Theoretical foundations: A review

Consider two alternative FA solutions – unidimensional and multiple-correlated – which are fitted to a set of item scores. Suppose further that both solutions are found to be acceptable by internal criteria, a situation which is quite usual in applications (e.g. Ferrando and Navarro-Gonzalez, 2018). The aim of the proposal summarized here is to assess which of the competing solutions is more appropriate in terms of external validity.

The null hypothesis in the proposal assumes (a) there is a general common factor running through the entire set of items, and (b) all the relations between the multiple factors and the relevant external variables are mediated by the general factor. In this case, the unidimensional solution is the most appropriate in terms of validity. At this point we note the proposal is intended to work on a variable-by-variable basis. So, it will be summarized using a single external variable.

The null hypothesis above can be described by using a second-order FA schema as follows. Assumption (a) above implies the correlated factors in the multiple solution, which we shall denote from now on as primary factors, behave as indicators of a single general factor. Assumption (b) implies the only parts of the primary factor not accounted for by the general factor are unrelated to the external variable.

The implications of the null model in terms of validity relations are considered in two facets: differential and incremental. In differential validity terms, the score estimates derived from the primary factors are expected to be related to the external variable in the same way as they are related to the general factor. As for incremental validity, the implications of the null model are the prediction of the external variable which is made from the single (general) factor score estimates cannot be improved upon by using the primary factor score estimates in a multiple regression schema.

Let $\hat{\theta}_{ik}$ be the factor-score estimate of individual i in the k primary factor, and let θ_{ik} be the corresponding true factor score. We write

$$\hat{\theta}_{ik} = \theta_{ik} + \varepsilon_{ik}, \tag{H.2.1}$$

where ε_{ik} denotes the measurement error. The true scores θ_k are assumed to be distributed with zero expectation and unit variance. It is further assumed $\hat{\theta}_{ik}$ is conditionally unbiased (i.e. $E(\hat{\theta}_{ik}|\theta_{ik}) = \theta_{ik}$, which implies the measurement errors are uncorrelated with the true trait levels. It then follows the squared correlation between $\hat{\theta}_k$ and θ_k is

$$\rho_{(\hat{\theta}_k,\theta_k)} = \frac{Var(\theta_k)}{Var(\hat{\theta}_k)} = \frac{1}{1+Var(\varepsilon_k)} = \frac{1}{1+E(Var(\varepsilon_{ik}|\theta_{ik}))} = \rho_{(\hat{\theta}_k,\hat{\theta}_k)} \tag{H.2.2}$$

which is taken as the marginal reliability of the factor score estimates (see Ferrando and Lorenzo-Seva, in press). Denote now by y the external variable or criterion also assumed to be scaled with zero mean and unit variance and by $\rho_{(\hat{\theta}_k,y)}$ the correlation between the k_{th} factor score estimates and the criterion (i.e. the raw validity coefficient). From the results above it follows that the disattenuated correlation between the estimated primary factor scores and the criterion

$$\hat{\rho}_{(\theta_k,y)} = \frac{\rho_{(\hat{\theta}_k,y)}}{\sqrt{\rho_{(\hat{\theta}_k,\hat{\theta}_k)}}} \tag{H.2.3}$$

is an unbiased estimate of the corresponding correlation between the true primary scores and the criterion (i.e. the true validity coefficient). Now let γ_{kg} be the loading of the k primary factor on the general factor (i.e. the second-order loading). If the null model is correct, the following result should hold

$$\frac{\hat{\rho}(\theta_1,y)}{\gamma_{1g}} = \cdots \frac{\hat{\rho}(\theta_k,y)}{\gamma_{kg}} = \cdots \frac{\hat{\rho}(\theta_q,y)}{\gamma_{qg}}. \tag{H.2.4}$$

In words, equation H.2.4 means the primary factors relate to the external variable in the same proportion to how they relate to the general factor. So, after correcting for this proportionality, the corrected indices should all be equal (i.e. no differential validity). To test this result, **unival** uses the following schema. First, it provides the Bootstrap-based confidence interval for each of the scaled coefficients in equation H.2.4. Second, the median value of the scaled coefficients is obtained, and the most extreme scaled value is subtracted from the median. Next, a confidence interval for this difference is obtained via Bootstrap resampling, and a check is made to see whether the zero value falls within this interval or not. This second procedure provides a single difference statistic regardless of the number of primary factors.

If the equality test is found not tenable, then the alternative explanation (i.e. differential validity) is the unique parts of the primary factors are still differentially related to the external variable beyond the relations that are mediated by the general factor. If this were so, validity information would be lost if the unidimensional model was chosen instead of the multiple model.

We turn now to incremental validity. The starting point of the proposal by (Ferrando and Lorenzo-Seva, 2019) was based on two results. First, the score estimates on the general factor are a linear composite of the score estimates on the primary factors in which the weights aim to maximize the accuracy of the general scores. And second, the multiple-regression composite, which is also based on the primary factor score estimates, has weights aimed at maximizing the correlation with the external variable. In a truly unidimensional solution both sets of weights are expected to be proportional, and the predictive power of the general score estimates and the primary score estimates to be the same. More in detail, (Ferrando and Lorenzo-Seva, 2019) proposed correcting the primary factor score estimates for measurement error, and then obtained single and multiple corrected correlation estimates whose expected values were the same under the null model above. Under the alternative hypothesis, on the other hand, the corrected multiple correlation (denoted by R_c) was expected to be larger than the single correlation based on the general scores (denoted by $\hat{\rho}_{\hat{\theta}_g y}$). The procedure implemented in **unival** for testing the null hypothesis of no incremental validity is to compute the difference $R_c - \hat{\rho}_{\hat{\theta}_g y}$, obtain the Bootstrap confidence interval for this difference, and check whether the zero value falls within the interval or not. If the null hypothesis is rejected, the alternative explanation (i.e. incremental validity) is the primary score estimates contain additional information allowing the multiple prediction based on them to be significantly better than the prediction based only on the general scores.

New methodological contributions

The present article extends the original proposal by (Ferrando and Lorenzo-Seva, 2019) in two directions. First, the procedure can now be correctly used with types of score estimate other than those considered initially. Second, an approximate procedure is proposed for testing essential unidimensionality against a solution in only two correlated factors.

As for the first point, the original proposal is based on factor score estimates behaving according to the assumptions derived from equation H.2.1. Appropriate scores of this type are mainly maximum-likelihood (ML) scores, which, in the linear FA model are known as Bartlett's (1937) scores (see Ferrando and Lorenzo-Seva, in press, for a discussion). However, other types of scores are in common use in FA applications. In particular, Bayes Expected-A-Posteriori (EAP) scores have a series of practical advantages in nonlinear FA applications (Bock and Mislevy, 1982) and are, possibly, the most commonly used scoring schema for this type of solution. EAP scores, however, are always inwardly biased (i.e. regressed towards the mean) and so do not fulfill the basic assumptions on which the original procedure was based.

Simple adaptations and corrections of the existing procedures can be obtained by viewing the EAP scores as the result of shrinking the ML scores towards the zero population mean so the shrinkage factor is the marginal reliability (Bock and Mislevy, 1982). By using this concept in the assessment of differential validity, it follows that the expected value of the raw correlation between the EAP score estimates for the k factor and y is given by

$$E(r_{(\hat{\theta}_{kEAP},y)}) = \frac{\rho_{(\theta_k,y)}}{\sqrt{1+E(Var(\varepsilon_{ik}|\theta_{ik}))}} \qquad \text{(H.3.1)}$$

Indeed, the conditional variances in the denominator of H.3.1 are not known, because they are based on the ML unbiased estimates. However, as the number of items increases, the posterior distribution approaches normality (Chang and Stout, 1993), and the posterior standard deviation (PSD) associated with the EAP estimate becomes equivalent to an asymptotic standard error (Bock and Mislevy, 1982). So, for factors defined, say, by 8 or more items, the following correction is expected to lead to appropriate disattenuated validity coefficients

$$\hat{\rho}_{(\theta_k,y)} = r_{(\hat{\theta}_{kEAP},y)}\sqrt{1+E(PSD^2(\theta_{ik}))}. \tag{H.3.2}$$

For very short item sets, the PSDs can be noticeably smaller than the standard errors because of the additional information contributed by the prior. The strategy proposed in this case is first to approximate the amounts of information from the PSDs by using the approximate relation (Wainer and Mislevy, 2000, p. 74)

$$PSD(\hat{\theta}) \cong \frac{1}{\sqrt{I(\hat{\theta}+1)}} \tag{H.3.3}$$

and then to use the modified correction

$$\hat{\rho}_{(\theta_k,y)} = r_{(\hat{\theta}_{kEAP},y)}\sqrt{1+E(\frac{1}{I(\hat{\theta}_{ik})})}. \tag{H.3.4}$$

Once the EAP-based disattenuated validity estimates have been obtained, they are used in the contrast H.2.4 in the same way as those derived from the ML scores.

We turn now to incremental validity. If EAP scores are used, the corrected estimate based on the general factor score estimates (denoted by $\hat{\rho}_{\hat{\theta}_g y}$) can be obtained as

$$\hat{\rho}_{(\theta_g,y)} = r_{(\hat{\theta}_{gEAP},y)}s_{(\hat{\theta}_{gEAP})}(1+E(\frac{1}{I(\hat{\theta}_{ik})})) \tag{H.3.5}$$

or, if the PSD approximation is used

$$\hat{\rho}_{(\theta_g,y)} = r_{(\hat{\theta}_{gEAP},y)}s_{(\hat{\theta}_{gEAP})}(1+E(PSD^2(\theta_{ik}))) \tag{H.3.6}$$

where $s_{(\hat{\theta}_{gEAP})}$ is the standard deviation of the EAP score estimates. As for the multiple estimate based on the primary factor scores (denoted by R_c), only the covariances between the score estimates and the criterion must be corrected when EAP estimates are used instead of ML estimates (see Ferrando and Lorenzo-Seva, 2019). EAP-based unbiased estimates of these covariances can be obtained as

$$\hat{C}ov_{\theta_k,y} = Cov_{(\hat{\theta}_{kEAP},y)}[1+E(PSD^2(\theta_{ik}))] \tag{H.3.7}$$

or, by using the PSD-to-Information transformation if the number of items is very small

$$\hat{C}ov_{\theta_k,y} = Cov_{(\hat{\theta}_{kEAP},y)}[1+E(\frac{1}{I(\hat{\theta}_{ik})})]. \tag{H.3.8}$$

Once the vector with the corrected covariances has been obtained, the rest of the procedure is the same as when it is based on ML score estimates.

Overall, the basis of the proposal so far discussed is to: (a) transform the EAP scores so they (approximately) behave as ML scores; (b) transform the PSDs so they will be equivalent to standard errors, and (c) use the transformed results as input in the standard procedure. The transformations are very simple, and the proposal is expected to work well in practical applications, as the simulation study below suggests. However, unstable or biased results might be obtained if the marginal reliability estimate used to correct for shrinkage was itself unstable or biased, or if the PSDs were directly used as if they were standard errors and the contribution of the prior was substantial.

This approximate procedure is expected to be useful in practice, because in many applications decisions must be taken about using one or two common factors. The problem in this case is a second-order solution can only be identified with three or more primary factors, and so, the initial proposal cannot be used in the bidimensional case. An approximate approach, however, can be used with the same rationale as in the original procedure.

Consider two matrices of factor score estimates (either ML or EAP): an $N \times 2$ matrix containing the estimates obtained by fitting the correlated two-factor solution, and an $N \times 1$ matrix containing the score estimates obtained by fitting the unidimensional (Spearman's) model to the item scores. Next, consider the following regression schemas in which the primary factor score estimates in the $N \times 2$ matrix are corrected for measurement error. The first regression is of the unidimensional score estimates on the corrected primary factor score estimates. The second is the regression of the criterion on the same corrected factor score estimates. Now, if the unidimensional solution is essentially correct in terms of validity, then the profiles of weights for predicting the general scores and those for predicting the criterion are expected to be the same except for a proportionality constant. Denoting by $\beta_g 1$ and $\beta_g 2$ the weights for predicting the general scores from the corrected primary estimates, and by $\beta_y 1$ and $\beta_y 2$ the corresponding weights for predicting the criterion, the contrast we propose for testing the null hypothesis no differential validity is

$$\frac{\beta_g 1}{\beta_y 1} = \frac{\beta_g 2}{\beta_y 2} \tag{H.3.9}$$

and is tested by using the same procedure as in equation H.2.4.

With regards to incremental validity, the null hypothesis of essential unidimensionality indicates both linear composites will predict the criterion equally well. So, if we denote by y'_g the composite based on the $\beta_g 1$ and $\beta_g 2$ weights, and by y'_y the composite based on the $\beta_y 1$ and $\beta_y 2$ weights, the test of no incremental validity is based on the contrast $r(y'_y, y) - r(y'_g, y)$, and is tested in the same way as the standard contrast above.

The unival package details

The current version of the (**unival**) package, which is available through CRAN, contains one main function (and additional internal functions) for implementing the procedures described in the sections above. Further details on the theoretical bases of **unival** are provided in (Ferrando and Lorenzo-Seva, 2019). The function usage is as follows.

```
unival(y, FP, fg, PHI, FA_model = 'Linear', type, SEP, SEG, relip, relig,
percent = 90, display = TRUE)
```

- `y`, the related external variable,
- `FP`, the primary factor score estimates,
- `fg`, the general or second-order factor score estimates. This argument is optional except when two primary factors are specified. In this case, second-order general score estimates cannot be obtained,
- `PHI`, inter-factor correlation matrix,
- `FA_model`, Which FA-model was used for calibration and scoring. Available options are: "Linear" (by default) or "Graded". The Graded option refers to the nonlinear FA model, in which item scores are treated as ordered-categorical variables, and includes binary scores as a specific case,
- `type`, Which type of factor score estimates were used in FP and fg. The two available options are: "ML" or "EAP" scores. If not specified, ML estimation will be assumed,
- `SEP`, Standard Errors (ML scores) or PSDs (EAP scores) for primary factor scores (only required when the "Graded" option is used),
- `SEG`, Standard Errors (ML scores) or PSDs (EAP scores) for the general factor (only required when the "Graded" option is used),
- `relip`, the marginal reliabilities of the primary factor scors estimates. Optional when three or more primary factors are specified; otherwise, the user should provide them,
- `relig`, the marginal reliability of the general factor score estimates (optional).

The data provided should be a data frame or a numerical matrix for input vectors and matrices, and numerical values for the arguments containing a single element, like `relig`. The package imports three additional packages: **stats** (R Core Team, 2018), **optimbase** (Bihorel and Baudin, 2014) and **psych** (Revelle, 2018), for internal calculations (e.g. using the '`fa`' function from **psych** package for performing the FA calibration).

Since the function requires the factor score estimates as input, these estimates must be obtained from the raw data (i.e. the raw item scores) before **unival** is used. We recommend the non-commercial FACTOR program (Lorenzo-Seva and Ferrando, 2013) to obtain EAP estimates under the linear and the graded FA model, or the **mirt** R package (Chalmers, 2012) to obtain ML and EAP estimates for both models. FACTOR also provides PSDs for the EAP scores. Finally, both programs provide marginal reliability estimates for the chosen factor scores.

Simulation studies

The sensitivity of the procedures proposed in **unival**, for both differential and incremental validity, depends on two main factors. The first is the relative strength of the relations between (a) the general factor scores and the external variables, and (b) the primary factor scores and the external variable. The second is the extent of the agreement between the relations between the unique parts of the primary factor and the external variables and the relations between the primary factor scores and the general factor. In summary, differential and incremental validity are expected to be clearly detected when (a) the primary factor scores are more strongly related to the external variable than to the general scores, and (b) the relation between the unique parts of the primary scores and the external variables is the opposite of the relation between the corresponding factors and the general factor. The opposite condition: (a) a general, dominant factor relates more strongly to the external variable than the primary factors do; and (b) a similar profile of relations in which the primary factors relate to the external variable in the same way as they do with the general factor, is very difficult to distinguish from the null hypothesis on which the procedures are based.

Ferrando and Lorenzo-Seva (2019) undertook a general simulation study in which the determinants above were manipulated as independent variables together with sample and model size. The study was based on the linear FA model and Bartlett's ML score estimates. In this article we replicated the study above but we discretized the continuous item responses in five response categories (i.e. a typical Likert score) and fitted the data using the non-linear FA model, thus treating the item scores are ordered-categorical variables. In addition, the factor score estimates were Bayes EAP scores. The present study, then, considers the second potential FA model that can be used in **unival**, and assesses the behavior of some of the new developments proposed in the article (the use of Bayes scores instead of ML scores). Because the design and conditions of the study were the same as those in Ferrando and Lorenzo-Seva (2019) the results are only summarized here. Details and tables of results can be obtained from the authors. The results generally agreed quite well with those obtained in the original study except for the (unavoidable) loss of power due to categorization. More in detail, in the study under the null model, neither spurious differential nor incremental validity was detected in any of the conditions.

In the studies in which the alternative model was correct, the following results were obtained. Differential validity was correctly detected except in the least favorable cells: dominant general-factor relations and profile agreement. As for incremental validity, the loss of power was more evident, and the procedure was less sensitive than in the continuous case: when the profiles of relations agreed (i.e. when the primary factors related to the external variable in the same way as they related to the general factor), **unival** failed to detect the increments in predictive power. This result, which, to a lesser extent, had already been obtained in the original study, suggests the unique relations have already been taken into account by the general factor score estimates. So, the multiple-regression linear composite, with weights very similar to those of the general factor score composite, does not substantially add to the prediction of the external variable. Overall, then, the results of the study suggest that in low-sensitivity conditions the **unival** outcome leads to the unidimensional model being chosen even when unique relations with the criterion do in fact exist. This choice, however, is probably not a practical limitation, as in these conditions the unidimensional model is more parsimonious and can explain the validity relations well. Finally, as for the differences with the previous study, the results suggest the **unival** procedures also work well with the non-linear FA model and Bayes scores. However, as expected, the categorization of the responses leads to a loss of information which, in turn, results in a loss of sensitivity and power. The most reasonable way to compensate for this loss would probably be to use a larger number of items.

Illustration with real data

The **unival** package contains an example dataset – SAS3f – which is a matrix containing a criterion (marks on a final statistics exam), the primary factor score estimates and the general factor score estimates in a sample of 238 respondents. Both the primary and general scores were EAP estimates obtained with the FACTOR (Lorenzo-Seva and Ferrando, 2013) program.

The instrument under scrutiny is the Statistical Anxiety Scale (SAS, Vigil-Colet et al., 2008) a 24-item instrument which was initially designed to assess three related dimensions of anxiety: Examination anxiety (EA), asking for help anxiety (AHA) and interpretation anxiety (IA). Previous studies have obtained a clear solution in three highly-related factors but have also found an essentially unidimensional solution is tenable. So, the problem is to decide whether it is more appropriate to use only single-factor scores measuring an overall dimension of statistical anxiety or it is preferable (and more informative) to use the factor score estimates in each of the three dimensions.

The only remaining argument for running **unival** with minimal input requests is the inter-factor correlation matrix between the primary factors. The example should be specified as follows:

```
> PHI = cbind(c(1,0.408,0.504),c(0.408,1,0.436),c(0.504,0.436,1))
> y = SAS3f[,1]
> FP = as.matrix(SAS3f[,2:4])
> fg = SAS3f[,5]
> unival(y = y, FP = FP, fg = fg, PHI = PHI, type = 'EAP')
```

The output from the above command is:

```
Unival: Assessing essential unidimensionality using external validity information

Differential validity assessment:

0.6012 (0.4615 - 0.7311)
0.2362 (0.0280 - 0.4172)
0.3635 (0.2390 - 0.5035)

Maximum difference

0.2377 (0.0891 - 0.3587) *

Incremental validity assessment:

0.3164 (0.2328 - 0.3944)
0.4107 (0.3362 - 0.4720)

Incremental value estimate

0.0943 (0.0203 - 0.1492) **

* Some factors are more strongly or weakly related to the criterion that can be
 predicted from their relations to the general factor
** There is a significant increase in accuracy between the prediction based on the
 primary factor score estimates and that based on the general factor score  estimates.
```

Overall, the results seem to be clear. In differential validity terms, the confidence intervals for the first and second factors do not overlap, and the zero value falls outside the maximum-difference confidence interval. The interpretation is the primary factors relate to the criterion in ways that cannot be predicted from their relations with the general factor. More specifically, the first factor (AHA) seems to be more strongly related, and the second factor (IA) more weakly related to the criterion than could be predicted by their relations with the general factor.

Incremental-validity results are also clear: the prediction of the criterion based on the primary factor estimates clearly outperforms the prediction that can be made from the general factor score estimates when the regressions are corrected for measurement error. Note in particular the zero value falls well outside the confidence interval of the incremental validity estimate. To sum up, it is clear both information and predictive power will be lost in this example if the single or general factor score estimates are used as a summary of the estimates based on the three anxiety factors. So, in terms of validity, the FA solution in three correlated factors seems to be preferable.

Concluding remarks

In the FA literature, several authors (e.g. Carmines and Zeller, 1991; Floyd and Widaman, 1995; Goldberg, 1972; Mershon and Gorsuch, 1988) have pointed out the dimensionality of a set of item scores cannot be decided solely in internal terms. Rather, the ultimate criterion for judging what the most appropriate solution is should be how the scores derived from this solution relate to relevant external variables. In spite of this, however, external information is rarely used in FA-based assessments. One explanation for this state of affairs is, indeed, the difficulty of collecting additional relevant external measures. Apart from this, however, clear and rigorous procedures on how to carry out this assessment have only been proposed recently and, so far, have not been implemented in non-commercial software. For this reason, we believe **unival** is a useful additional tool for researchers who use FA in psychometric applications.

unival has been designed to work with scores derived from an FA solution rather than from raw item scores, and this has both shortcomings and advantages. Thus, at the minimal-input level, potential users of the program have to be able to carry out factor analyses with other programs, and, particularly, to obtain factor score estimates. Furthermore, they need to know what types of score have been computed by the program. More advanced **unival** usages require users to know how to obtain marginal reliability estimates for the factor scores or how to perform second-order factor analysis. To sum up, the program is designed for practitioners with some level of proficiency in FA. In principle, this is a potential shortcoming but does not restrict the usefulness of the program. As described above, all the input required by **unival** can be obtained from non-commercial FA packages, some of which are also quite user friendly.

The choice of the factor scores as input, on the other hand, makes the program extremely flexible and versatile. **unival** can work with scores derived from standard linear FA solutions or from non-linear solutions (which include the multidimensional versions of the graded-response and the two-parameter IRT models). Furthermore, users can choose to provide the minimal input options, or can tailor the input by choosing the type of marginal reliability estimate to be used in the error corrections or the general factor score estimates on which the analyses are based (second-order factor scores or scores derived from directly fitting the unidimensional model). No matter how complex the model or input choices are, however, the output provided by **unival** is extremely simple and clear to interpret, as the illustrative example shows.

Acknowledgments

This project has been made possible by the support of the Ministerio de Economía, Industria y Competitividad, the Agencia Estatal de Investigación (AEI) and the European Regional Development Fund (ERDF) (PSI2017-82307-P).

Bibliography

M. S. Bartlett. The statistical conception of mental factors. *British Journal of Psychology*, 28: 97–104, 1937. URL https://doi.org/10.1111/j.2044-8295.1937.tb00863.x. [p429]

A. Beauducel, C. Harms, and N. Hilger. Reliability estimates for three factor score estimators. *International Journal of Statistics and Probability*, 5(6):94–107, 2016. URL https://doi.org/10.5539/ijsp.v5n6p943. [p427]

S. Bihorel and M. Baudin. *optimbase: R port of the Scilab optimbase module*, 2014. URL https://CRAN.R-project.org/package=optimbase. R package version 1.0-9. [p431]

R. D. Bock and R. J. Mislevy. Adaptive eap estimation of ability in a microcomputer environment. *Applied Psychological Measurement*, 6(4):431–444, 1982. URL https://doi.org/10.1177/014662168200600405. [p429]

E. G. Carmines and R. A. Zeller. *Reliability and Validity Assessment*, volume 17. SAGE, 1991. ISBN 9780803913714. [p433]

R. P. Chalmers. mirt: A multidimensional item response theory package for the R environment. *Journal of Statistical Software*, 48(6):1–29, 2012. doi:10.18637/jss.v048.i06. [p431]

H. Chang and W. Stout. The asymptotic posterior normality of the latent trait in an irt model. *Psychometrika*, 58(1):37–52, 1993. URL https://doi.org/10.1007/BF02294469. [p429]

P. J. Curran, V. T. Cole, D. J. Bauer, W. A. Rothenberg, and A. M. Hussong. Recovering predictor–criterion relations using covariate-informed factor score estimates. *Structural Equation Modeling: A Multidisciplinary Journal*, 25(6):860–875, 2018. URL https://doi.org/10.1080/10705511.2018.1473773. [p427]

P. J. Ferrando and U. Lorenzo-Seva. Program factor at 10: origins, development and future directions. *Psicothema*, 29:236–241, 2017. URL https://doi.org/10.7334/psicothema2016.304. [p427]

P. J. Ferrando and U. Lorenzo-Seva. Assessing the quality and appropriateness of factor solutions and factor score estimates in exploratory item factor analysis. *Educational and Psychological Measurement*, 78(5):762–780, 2018. URL https://doi.org/10.1177/0013164417719308. [p427]

P. J. Ferrando and U. Lorenzo-Seva. An external validity approach for assessing essential unidimensionality in correlated-factor models. *Educational and Psychological Measurement*, 2019. URL https://doi.org/10.1177/0013164418824755. [p427, 428, 429, 430, 431, 432]

P. J. Ferrando and U. Lorenzo-Seva. On the added value of multiple factor score estimates in essentially unidimensional models. *Educational and Psychological Measurement*, in press. URL https://doi.org/10.1177/0013164418773851. [p427, 428, 429]

P. J. Ferrando and D. Navarro-Gonzalez. Assessing the quality and usefulness of factor-analytic applications to personality measures: A study with the statistical anxiety scale. *Personality and Individual Differences*, 123(1):81–86, 2018. URL https://doi.org/10.1016/j.paid.2017.11.014. [p428]

F. J. Floyd and K. F. Widaman. Factor analysis in the development and refinement of clinical assessment instruments. *Psychological assessment*, 7(3):286–299, 1995. URL https://doi.org/10.1037/1040-3590.7.3.286. [p427, 433]

A. Furnham. The development of single trait personality theories. *Personality and Individual Differences*, 11(9):923–929, 1990. URL https://doi.org/10.1016/0191-8869(90)90273-T. [p427]

L. R. Goldberg. Parameters of personality inventory construction and utilization: A comparison of prediction strategies and tactics. *Multivariate Behavioral Research Monographs*, 72(2):59, 1972. [p433]

S. J. Haberman. When can subscores have value? *Journal of Educational and Behavioral Statistics*, 33(2):204–229, 2008. URL https://doi.org/10.3102/1076998607302636. [p427]

U. Lorenzo-Seva and P. J. Ferrando. Factor 9.2: A comprehensive program for fitting exploratory and semiconfirmatory factor analysis and irt models. *Applied Psychological Measurement*, 37(6): 497–498, 2013. URL https://doi.org/10.1177/0146621613487794. [p427, 431, 432]

B. Mershon and R. L. Gorsuch. Number of factors in the personality sphere: Does increase in factors increase predictability of real-life criteria? *Journal of Personality and Social Psychology*, 55(4): 675–680, 1988. URL https://doi.org/10.1037/0022-3514.55.4.675. [p433]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL https://www.R-project.org/. [p431]

T. Raykov and G. A. Marcoulides. On studying common factor dominance and approximate unidimensionality in multicomponent measuring instruments with discrete items. *Educational and Psychological Measurement*, 78(3):504–516, 2018. URL https://doi.org/10.1177/0013164416678650. [p427]

S. P. Reise, W. E. Bonifay, and M. G. Haviland. Scoring and modeling psychological measures in the presence of multidimensionality. *Journal of personality assessment*, 95(2):129–140, 2013. URL https://doi.org/10.1080/00223891.2012.725437. [p427]

S. P. Reise, K. F. Cook, and T. M. Moore. Evaluating the impact of multidimensionality on unidimensional item response theory model parameters. In *Handbook of item response theory modeling*, pages 13–40. Routledge, 2015. [p427]

W. Revelle. *psych: Procedures for Psychological, Psychometric, and Personality Research*. Northwestern University, Evanston, Illinois, 2018. URL https://CRAN.R-project.org/package=psych. R package version 1.8.10. [p431]

A. Rodriguez, S. P. Reise, and M. G. Haviland. Evaluating bifactor models: Calculating and interpreting statistical indices. *Psychological Methods*, 21(3):137–150, 2016a. URL https://doi.org/10.1037/met0000045. [p427]

A. Rodriguez, S. P. Reise, and M. G. Haviland. Applying bifactor statistical indices in the evaluation of psychological measures. *Journal of personality assessment*, 98(3):223–237, 2016b. URL https://doi.org/10.1080/00223891.2015.1089249. [p427]

A. Vigil-Colet, U. Lorenzo-Seva, and L. Condon. Development and validation of the statistical anxiety scale. *Psicothema*, 20(1):174–180, 2008. URL https://doi.org/10.1037/t62688-000. [p432]

H. Wainer and R. J. Mislevy. Item response theory, item calibration and proficiency estimations. In H. Wainer, editor, *Computerized Adaptive Testing: A Primer*, pages 61–101. LEA, 2000. [p430]

Pere J. Ferrando
Department of Psychology
University Rovira i Virgili
Spain
0000-0002-3133-5466
perejoan.ferrando@urv.cat

Urbano Lorenzo-Seva
Department of Psychology
University Rovira i Virgili
Spain
0000-0001-5369-3099
urbano.lorenzo@urv.cat

David Navarro-Gonzalez
Department of Psychology
University Rovira i Virgili
Spain
0000-0002-9843-5058
david.navarro@urv.cat

R Foundation News

by Torsten Hothorn

Donations and members

Membership fees and donations received between 2019-01-07 and 2019-09-04.

Donations

Web Hosting Buddy (United States) ilustat (Portugal) Mike Foster (United Kingdom) Dreamz Inc. (Sweden) Dotcom-Monitor (United States) Loadview-Testing (United States) J. BRIAN LORIA (United States) Security Guard Training Central (United States) Driven Coffee Roasters (United States) WebHostingProf (United States) Bill Pikounis (United States) Daniel Wollschläger (Germany) Direction départementale des finances publiques des Yvelines, Versailles (France) Merck Research Laboratories, Kenilwort (United States) Novartis Pharma AG, Basel (Switzerland)

Supporting benefactors

INWT Statistics GmbH (Germany) Mirai Solutions GmbH, Zürich (Switzerland)

Supporting institutions

University of Iowa, Iowa City (United States)

Supporting members

Ashanka Beligaswatte (Australia) Chris Billingham (United Kingdom) Michael Blanks (United States) Robert Carnell (United States) Henry Carstens (United States) Gerard Conaghan (United Kingdom) Robin Crockett (United Kingdom) Michael Dorman (Israel) Andreas Eckner (United States) Gerrit Eichner (Germany) Johan Eklund (Sweden) Martin Elff (Germany) Mitch Eppley (United States) Spyridon Fortis (United States) Jan Marvin Garbuszus (Germany) J. Antonio García Ramírez (Mexico) Hlynur Hallgrímsson (Iceland) Martin Haneferd (Norway) ken ikeda (Japan) Christian Kampichler (Netherlands) Curtis Kephart (United States) Gavin Kirby (United Kingdom) David Knipping (United States) Sebastian Koehler (Germany) HOONJEONG KWON (Korea, Republic of) Luca La Rocca (Italy) Adrien Le Guillou (France) Sharon Machlis (United States) Michal Majka (Austria) Ernst Molitor (Germany) David Monterde (Spain) Jens Oehlschlägel (Germany) francis pampush (United States) Stavros Panidis (Greece) Gopal Peddinti (Finland) Fergus Reig Gracia (Spain) Stefano Rezzonico (Canada) Cristián Rizzi (Argentina) Adriaan Rowan (South Africa) Henrik Schirmer (Norway) Robert Selden (United States) Harald Sterly (Germany) Arthur Szasz (Brazil) Robert van den Berg (Austria) Mark van der Loo (Netherlands) Earo Wang (Australia) Roger Watson (United Kingdom) Klaus Wiese (Honduras) Rahadian Zulfadin (Indonesia)

Torsten Hothorn
Universität Zürich, Switzerland Torsten.Hothorn@R-project.org

R News

by R Core Team

CHANGES IN R 3.6.1

INSTALLATION on a UNIX-ALIKE

- The default detection of the shell variable ‘libNN’ is overridden for derivatives of Debian Linux, some of which have started to have a ‘/usr/lib64’ directory. (E.g. Ubuntu 19.04.) As before, it can be specified in ‘config.site’.

UTILITIES

- `R CMD` knows the values of AR and RANLIB, often set for LTO builds.

BUG FIXES

- On Windows, GUI package installation via menuInstallPkgs() works again, thanks to Len Weil’s and Duncan Murdoch’s PR#17556.

- `quasi(*,variance = list(..))` now works more efficiently, and should work in all cases fixing PR#17560. Further, `quasi(var = mu(1-mu)) and quasi(var = "mu 3̂")` now work, and `quasi(variance = "log(mu)")` now gives a correct error message.

- Creation of lazy loading database during package installation is again robust to ‘Rprofile’ changing the current working directory (PR#17559).

- `boxplot(y   f,horizontal=TRUE)` now produces correct x- and y-labels.

- `rbind.data.frame()` allows to keep ‘`<NA>`’ levels from factor columns (PR#17562) via new option factor.exclude.

  Additionally, it works in one more case with matrix-columns which had been reported on 2017-01-16 by Krzysztof Banas.

- Correct messaging in C++ pragma checks in tools code for `R CMD check`, fixing PR#17566 thanks to Xavier Robin.

- print()ing and auto-printing no longer differs for functions with a user defined print.function, thanks to Bill Dunlap’s report.

- On Windows, ‘`writeClipboard(..,format = <n>)`’ now does correctly pass format to the underlying C code, thanks to a bug report (with patch) by Jenny Bryan.

- `as.data.frame()` treats 1D arrays the same as vectors, PR#17570.

- Improvements in `smoothEnds(x,*)` working with NAs (towards `runmed()` working in that case, in the next version of R).

- `vcov(glm(<quasi>),dispersion = *)` works correctly again, fixing PR#17571 thanks to Pavel Krivitsky.

- `R CMD INSTALL` of binary packages on Windows now works also with per-directory locking.

- `R CMD INSTALL` and `install.packages()` on Windows are now more robust against a locked file in an earlier installation of the package to be installed. The default value of option install.lock on Windows has been changed to TRUE.

- On Unix alikes (when readline is active), only expand tilde ‘`( )`’ file names starting with a tilde, instead of almost all tildes.

OTHER RECENT SIGNIFICANT CHANGES IN R

There were two important, user-visible changes in version 3.6.0:

- Serialization format version 3 becomes the default for serialization and saving of the workspace (`save(),serialize(),saveRDS(),compiler::cmpfile()`). Serialized data in format 3 cannot be read by versions of R prior to version 3.5.0. Serialization format version 2 is still supported and can be selected by version = 2 in the save/serialization functions. The default can be changed back for the whole R session by setting environment variables R_DEFAULT_SAVE_VERSION and R_DEFAULT_SERIALIZE_VERSION to 2. For maximal back-compatibility, files 'vignette.rds' and 'partial.rdb' generated by R CMD build are in serialization format version 2, and resave by default produces files in serialization format version 2 (unless the original is already in format version 3).

- The default method for generating from a discrete uniform distribution (used in `sample()`, for instance) has been changed. This addresses the fact, pointed out by Ottoboni and Stark, that the previous method made `sample()` noticeably non-uniform on large populations. See PR#17494 for a discussion. The previous method can be requested using `RNGkind()` or `RNGversion()` if necessary for reproduction of old results. Thanks to Duncan Murdoch for contributing the patch and Gabe Becker for further assistance.

  The output of `RNGkind()` has been changed to also return the `kind` used by `sample()`.