The R Journal

Volume 17/1, March 2025

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

Contents

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

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

Email of editors and editorial board:
r-journal@R-project.org

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

Editorial

by Rob J Hyndman

Editorial changes

At the start of each year, one of the Executive Editors of the R Journal steps down and a new Executive Editor is appointed. This year, we say farewell to Simon Urbanek, and thank him for his service as Executive Editor over the period 2021-2024, including being Editor-in-Chief in 2023. And we welcome Emily Zabor, who joins the team of Executive Editors. Emily is an Associate Professor of Medicine at the Cleveland Clinic in Ohio, and is an active contributor to the R community responsible for several CRAN packages. She has been an Associate Editor of the R Journal since 2020, and has also been on the editorial boards of the Journal of Urology, European Urology, and the Journal of Clinical Oncology. We are delighted to have her step up to the role of Executive Editor for the next four years.

The other change at the start of each year is that the Editor-in-Chief changes. We thank Mark van der Loo for his service as Editor-in-Chief for 2024; he will continue as an Executive Editor for one more year. I will be the Editor-in-Chief for 2025, and I look forward to working with the editorial team to continue the development of the R Journal. The other continuing Executive Editor is Emi Tanaka.

We also welcome some new Associate Editors: Tomasz Woźniak, Alexander Kowarik, Thomas Nagler, Robin Lovelace, Valentin Todorov, Julia Wrobel, and David Ardia. We are grateful to them for their willingness to contribute to the R Journal. Several Associate Editors have stepped down, and we thank them for their contributions: Nicholas Tierney, Chris Brunsdon, Ivan Svetunkov and Romain Lesur. We also thank the many Associate Editors who continue to contribute to the R Journal.

Mitchell O'Hara-Wild has been providing valuable technical assistance to the R Journal for several years, and we have decided to formalize his role as Technical Editor of the journal for 2025. We are grateful to him for his ongoing contributions to the R Journal.

In this issue

On behalf of the editorial board, I am pleased to present Volume 17 Issue 1 of the R Journal. This issue features ten research articles, plus news from CRAN and the R Foundation.

The ten articles relate to R packages on the following diverse range of topics.

Forests and trees

- Random Forests for Time-Fixed and Time-Dependent Predictors: The DynForest R Package
- Structured Bayesian Regression Tree Models for Estimating Distributed Lag Effects: The R Package dlmtree

Longitudinal and panel data

- latrend: A Framework for Clustering Longitudinal Data
- panelPomp: Analysis of Panel Data via Partially Observed Markov Processes in R

Sampling tools

- CDsampling: An R Package for Constrained D-Optimal Sampling in Paid Research Studies
- LCCR: An R package for inference on latent class models for capture-recapture data with covariates

Spatial smoothing

- spheresmooth: An R Package for Penalized Piecewise Geodesic Curve Fitting on a Sphere
- Space-Time Smoothing of Survey Outcomes using the R Package SUMMER

Statistical simulation

- SimEngine: A Modular Framework for Statistical Simulations in R

Factor analysis

- SIREN: An Hybrid CFA-EFA R Package for Controlling Acquiescence in Restricted Factorial

All packages discussed are available on CRAN. Supplementary material with fully reproducible code is available for download from the Journal website.

Rob J Hyndman
Monash University

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

SIREN: A Hybrid CFA-EFA R Package for Controlling Acquiescence in Restricted Factorial Solutions

by David Navarro-Gonzalez, Pere J. Ferrando, Fabia Morales-Vives, Ana Hernandez-Dorado

Abstract The **siren** package implements a two-step procedure that allows restricted (confirmatory) factor analytic (FA) solutions to be fitted in data matrices that have been previously 'cleaned' of the biasing effects of acquiescent responding (AR) by using an unrestricted (exploratory) FA specification. So, the procedure which is implemented is hybrid: i.e. (a) an unrestricted acquiescence (ACQ) factor is first fitted to the data, (b) the residual data (or covariance) matrix after the impact of ACQ has been partialled-out is obtained, and (c) a restricted FA solution is fitted to the residual matrix. Although the basic foundations of the procedure are known, it contains new methodological developments that are, all of them, implemented in the package. So, provided that fully or partially balanced scales are available, the researcher will be able to: (a) calibrate a multidimensional CFA solution which is free from AR, (b) assess the goodness of model-data fit of this solution, and (c) obtain individual score estimates in the content as well as in the ACQ factors. The functioning of the program is assessed by means of a simulation study, and illustrated with a toy example. Its usefulness is also demonstrated by using an illustrative example in the personality domain. **siren** is submitted to be a valuable tool for use in item CFA applications when AR is expected to be operating.

1 Introduction

Valid interpretation of typical-response or non-cognitive (personality, attitude, interest, etc.) test scores requires that the item responses that are to be calibrated and scored meet a series of conditions. Of these, one of the more basic is that the responses truly reflect the influence of the content variables intended to be measured and are not affected by other systematic determinants unrelated to this content. In particular, this article is concerned with Acquiescent Responding (AR): the tendency to agree or endorse an item regardless of its content (Messick, 1966) as a response determinant. When AR is operating and is not properly controlled, a series of invalidating effects can be expected, both at the calibration level (biased item parameter estimates and spurious evidence of multidimensionality) and at the scoring level (scores that reflect a mixture of content and AR and so they cannot be univocally interpreted).

Test designers and practitioners are generally aware of the potential invalidating effects of AR, and use procedures for controlling them. Of these, the most common is to develop balanced scales. Provided that the content variables can be considered as continuous dimensions with two end poles, in a fully balanced scale half of the items are keyed toward one of the poles while the other half are keyed toward the opposite pole (Savalei and Falk, 2014; Vigil-Colet et al., 2020).

Statistical control of AR in test designs that use balanced scales is generally based on factor analytic (FA) procedures, and essentially entails explicitly modeling AR as an additional non-content factor. This FA-based control operates at two levels (see Ferrando et al., 2003): first, at the level of the factor structure obtained in the calibration stage (thus avoiding or minimizing the invalidating effects mentioned above); and second, at the level of the factor score estimates derived from the calibration structure (thus, providing "cleaner" content score estimates that have a more univocal interpretation).

Within the general FA modeling, two main approaches exist at present (Savalei and Falk, 2014; de la Fuente and Abad, 2020). The first is fully confirmatory, and the solution is identified by restricting all the loadings on the additional Acquiescence (ACQ) factor to

have the same unit value (Billiet and McClendon, 2000). The second is exploratory or semi-confirmatory ((Ferrando et al., 2003)): First, (a) an unrestricted ACQ factor with (possibly) different loadings and (b) an also unrestricted (EFA) direct "content" solution are obtained. Second, the direct content solution is either analytically rotated (fully exploratory solution) or rotated against a specified or semi-specified target (semi-confirmatory solution). The pros and cons of both approaches have been discussed and compared by Savalei and Falk (2014) and de la Fuente and Abad (2020). Both studies concluded that the confirmatory approach is more robust and user-friendly than the EFA with target rotation. However, it is also more sensitive to violation of the unit-weight loading assumption for the ACQ factor.

Within the general FA-based controlling procedure, two main approaches exist at present (Savalei and Falk, 2014; de la Fuente and Abad, 2020). The first is fully confirmatory, and, in it, the structural FA solution is identified by restricting all the loadings on the additional Acquiescence (ACQ) factor to have the same unit value (Billiet and McClendon, 2000). The second is exploratory or semi-confirmatory (Ferrando et al., 2003): First, an unrestricted ACQ factor with (possibly) different loadings together with a direct (i.e. non-rotated) unrestricted or exploratory (EFA) "content" solution are obtained. Second, the direct content solution is either analytically rotated (in a fully exploratory solution) or rotated against a specified or semi-specified target (in a semi-confirmatory solution). The pros and cons of both approaches have been discussed by Savalei and Falk (2014) and de la Fuente and Abad (2020). Both studies concluded that the confirmatory approach is more robust and user-friendly than the semi-confirmatory EFA with target rotation. However, it is also more sensitive to violation of the unit-weight loading assumption for the ACQ factor.

The aim of this paper is to propose and implement a "hybrid" approach, named SIREN, that combines features of both the CFA and EFA approaches. Furthermore, the procedure is comprehensive in that it is intended for fitting multiple content solutions and can also be used with scales that are not fully balanced (see below). Because we are using the same name for the proposed procedure and the package that implements it, in the remainder of the paper, we shall use the distinction "SIREN procedure" and "**siren** package" when necessary so as to avoid confusion.

Most of the basic foundations of SIREN have been discussed in the FA literature (e.g. Nunnally, 1978). Furthermore, the approach we shall propose is multi-step (see below), and the first step: estimating an unrestricted ACQ factor, is, essentially, the same as that used in the exploratory/semi-confirmatory approaches summarized below. So, this part of the proposal will not be discussed in detail but relevant references will be provided to the interested reader. On the other hand, the full proposal contains new developments, and these are the ones that will be discussed here in more detail.

1.1 Basic general results and rationale of the proposal

Consider a set of n items intended to measure p common content factors (e.g. personality dimensions). The basic FA model equation in the population is:

$$\mathbf{Z} = \mathbf{\Lambda}\boldsymbol{\theta} + \mathbf{\Psi}\mathbf{E} \tag{1}$$

where $\mathbf{Z}$ is an $n \times 1$ random vector of observed item scores; $\mathbf{\Lambda}$ is an $n \times p$ factor pattern matrix; $\boldsymbol{\theta}$ is an $p \times 1$ random vector of 'true' common factor scores; $\mathbf{\Psi}$ is an $n \times n$ diagonal matrix of unique-factor loadings, and $\mathbf{E}$ is an $n \times 1$ random vector of unique factor scores. With regards to scaling and assumed relations, the observed item scores are in reduced form (centered around the mean), the common factor scores and the unique scores are in standard scale (zero mean and unit variance), and, finally, the unique scores are assumed to be uncorrelated with the common factors and among them. In these conditions, the reproduced covariance matrix among the n item scores as implied by model (1) is given by the structural equation:

$$\mathbf{\Sigma} = \mathbf{\Lambda}\mathbf{\Phi}\mathbf{\Lambda}' + \mathbf{\Psi}^2 \tag{2}$$

where $\mathbf{\Phi}$ is $p \times p$ correlation matrix containing the correlations between the 'true' common factor scores. Generally, in the applications considered here, the $\mathbf{Z}$ scores will not only be mean-centered, but standardized scores, and so, the implied covariance matrix $\mathbf{\Sigma}$ in (2) will be a correlation matrix.

The main difference between an unrestricted (exploratory) and a restricted (confirmatory) solution within the general model (2) is in the constraints that are imposed to the pattern matrix $\mathbf{\Lambda}$. In an unrestricted solution, only minimal identification constraints are imposed, so that the common space: $\mathbf{\Lambda\Phi\Lambda'}$ in (2) is not restricted and multiple solutions of the same type, that fit all equally well, can be obtained from each other by rotation. In a restricted solution, the number of imposed restrictions makes the specified solution $\mathbf{\Lambda\Phi\Lambda'}$ unique, in the sense that it cannot be obtained by rotation of another solution (see Joreskog, 1969). Although a restricted solution can be obtained by using different sets of constraints, the most usual consist of imposing an independent-cluster structure (e.g. McDonald, 2000): each item has only a non-zero loading in one factor, having zero loadings in all the others.

At this point, we will start to develop a small, artificial toy example to help clarify the explanations that will follow. Suppose a questionnaire made up of 8 factorially simple items that measure two moderately correlated factors, so that the independent-cluster structure in the population is:

Table 1: Toy example: restricted CFA solution with two correlated content factors.

$$\mathbf{\Lambda} = \begin{bmatrix} 0.7 & 0.0 \\ -0.7 & 0.0 \\ 0.7 & 0.0 \\ -0.7 & 0.0 \\ 0.0 & 0.6 \\ 0.0 & -0.6 \\ 0.0 & 0.6 \\ 0.0 & -0.6 \end{bmatrix} \quad \mathbf{\Phi} = \begin{bmatrix} 1.0 & 0.3 \\ 0.3 & 1.0 \end{bmatrix}$$

A CFA estimation of this structure can be specified by constraining to zero the 8 elements of $\mathbf{\Lambda}$ that should be zero and freely estimating the remaining 8 loadings and the interfactor correlation based on the sample correlation matrix $\mathbf{R}$. Note that, for a solution of this type to be defined, the practitioner must be able to specify first, the number of content factors that the questionnaire intends to measure (two in the example), and second, the specific items that define each factor. Furthermore, the items are supposed to be all factorially simple, so that each item is a marker of the factor it measures and has negligible loadings on the remaining factors. These conditions are not easy to achieve, but can be feasible at advanced stages of test development.

Suppose now that the **content** structure of our example is that in table 1 but, at the same time, the item responses are also partly affected by AR, conceptualized as an additional non-content factor (see table 2 below). Now, even though the content structure was correct, if the specified two-factor structure above was directly fitted to $\mathbf{R}$, two types of distorted results would be expected. First, the goodness of model-data fit would not be good. Second, the loading and inter-factor correlation estimates would be biased with respect to the parameter values in (2) (see e.g. DeMars, 2014; Ferrando and Lorenzo-Seva, 2010).

The rationale of SIREN can now be explained from the results above. At the calibration level, the basic idea is to obtain a corrected or cleaned covariance/correlation matrix $\mathbf{R}_{corr}$ in which the impact of the ACQ factor has been partialled-out. If this is done correctly, a specified restricted CFA solution can be next fitted to $\mathbf{R}_{corr}$ instead of $\mathbf{R}$. This solution will now fit well, and the 'true' content parameters in table 1 will be well recovered. At the scoring level, once the ACQ and the content structures have been properly estimated, ACQ and content individual score estimates (i.e. factor scores) can be next obtained based on the unrestricted ACQ pattern and the restricted, CFA content solution.

As mentioned above, in order to control for the impact of AR, the items of the question-

naire have to be fully or partially balanced. In the present scenario, the condition of full balance implies that, within each factor, half of the items that define this factor are positively keyed and the other half are negatively keyed. The condition of partial balance implies here that all the factors contain positively and negatively keyed items, but that the number of positive and negative items is not the same at least in one factor (e.g. Lorenzo-Seva and Ferrando, 2009).

We shall now illustrate the points so far discussed with our toy example. Suppose now that the full content plus ACQ structure in the population is that in table 2. As the content pattern loadings show, however, the practitioner has done her work well and, within each factor, the items are fully balanced: within each content factor, half of the loadings are positive and half negative. As for the ACQ factor, (a) all the loadings are positive, and (b) they are smaller in magnitude than the content loadings. Both features are expected in empirical applications. First, AR is the tendency to agree with the item regardless of the direction in content (hence all loadings are expected to be positive). Second, in a well-designed measure, the item responses are expected to be far more determined by the content they measure than by ACQ.

Table 2: Toy example: complete solution when ACQ is operating. Balanced items.

$$\mathbf{\Lambda} = \begin{bmatrix} 0.7 & 0.0 & 0.1 \\ -0.7 & 0.0 & 0.2 \\ 0.7 & 0.0 & 0.3 \\ -0.7 & 0.0 & 0.3 \\ 0.0 & 0.6 & 0.3 \\ 0.0 & -0.6 & 0.3 \\ 0.0 & 0.6 & 0.2 \\ 0.0 & -0.6 & 0.1 \end{bmatrix} \quad \mathbf{\Phi} = \begin{bmatrix} 1.0 & 0.3 & 0.0 \\ 0.3 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Once a complete solution such as that in table 2 has been estimated, it can be next taken as a basis for obtaining ACQ and content score estimates (i.e. factor scores) for each individual. As discussed above, these scores will now be cleaner and have a more univocal interpretation.

1.2 General description of the procedure and relation with previous approaches

We propose a general multi-stage procedure in which the number of stages depends on whether the test is fully or only partially balanced. So, within each of the two conditions (full balance or partial balance), the stages will be described separately. Conceptually, however, it is useful to view the overall procedure as based on three general stages that are common in both conditions. In the first stage, an ACQ factor is estimated from the properties of the (partially or fully) balanced set of items, and the impact of this factor on the inter-item correlation matrix is partialled-out. In the second stage, a specified CFA solution is fitted to the 'cleaned' correlation matrix. Finally, in the third stage, individual score estimates are obtained from the hybrid solution (i.e. the unrestricted ACQ factor and the restricted content-factor solution).

Although the sequential rationale just described is conceptually the clearest, the structural solution at the second stage above can actually be specified and fitted in two ways. The first way directly follows from the corrected-correlation-matrix concept: to fit a CFA solution to a reduced correlation matrix which is free from ACQ. In the **siren** package, this way of fitting the model corresponds to the 'resid' method option (i.e. use the residual or corrected matrix as input for the CFA) as defined below. The second way is to take the estimated ACQ loadings obtained at the first stage as if they were fixed and known, and next to specify a full CFA solution that includes an additional ACQ factor with loadings fixed at the obtained values. In the **siren** package, this second choice corresponds to the 'fixed' method option defined below. As we shall see, the results from both approaches must

be the same. In either case, the corrected or residual correlation matrix is provided in the output of the SIREN package under the heading "rresidmatrix".

The explanation above allows the relation between SIREN and previous approaches to be discussed in more detail. If the solution is specified in the first way above ('resid' method option), SIREN can be viewed as a particular application of a residual covariance analysis approach, i.e. to initially correct a covariance or a correlation matrix for unwanted effects before it is used as input for further structural analyses (e.g. Andrews, 1984; Asparouhov and Muthen, 1984; DeCastellarnau and Saris, 2021; ten Berge, 2020).

If the second equivalent specification above: full CFA solution that includes the additional fixed ACQ factor ('fixed' method option), is used instead, then SIREN can be regarded as a modification of the CFA approach initially proposed by Billiet and McClendon (2000). In effect, in the latter, the ACQ loadings are fixed all of them to unity for identification purposes (which is generally unrealistic). In contrast, in SIREN these loadings are fixed at the (possibly different) values estimated at the first stage.

1.3 Background results and required general conditions

Consider again a fully-balanced questionnaire made up of n items (where n is even) that measure a set of (possibly related) traits $\theta_1 ... \theta_l ... \theta_m$, so that each item is a factorially pure measure of one of the m content factors plus of an acquiescence factor θ_a which is unrelated to the content factors. For an individual i that responds to an item j that measures content factor l, the structural model in a z-score metric (mean 0 and variance 1) is

$$z_{ij} = \lambda_{jl}\theta_{il} + \alpha_{ja}\theta_{ia} + \varepsilon_{ij} \tag{3}$$

This is a scalar specification of matrix equation (1) based on an independent-cluster pattern. Only a single content loading per item is specified because the remaining content loadings are zero. The α_{ja} loading is the loading item j has on the ACQ factor. Finally, the residual terms εs have zero means, and are uncorrelated with the factors or with one another.

The z item responses in (3) can be treated as categorical or (approximately) continuous. In the first case, the standardized scores would correspond to the continuous-unbounded "strength" latent variable that are assumed to underlie the observed scores and generate them according to a threshold mechanism (see Muthen, 1993). In the second case, they are directly the standardized item scores. From this general modeling, it follows that the inter-item correlations are polychoric correlations in the first case, and product-moment correlations in the second case (see e.g. Ferrando and Lorenzo-Seva, 2013, for further details). In the **siren** package, the user decides how the item responses are to be treated by using the 'corr' argument defined below, which has two options: "Pearson" (product-moment correlations) or "Polychoric" (Polychoric/Tetrachoric correlations). Once the correlation matrix has been obtained, the results that follow are common for both treatments.

Consider now the (polychoric or product-moment) reduced inter-item correlation matrix with communalities in the main diagonal (see Ferrando and Lorenzo-Seva, 2010). If (a) all the assumptions so far (independent-cluster structure, full balance within factors) were met, (b) the specified FA model was correct, and (c) the item communalities were known, then the first centroid loading (e.g. Lawley, 1960) for item j would correspond to the loading this item has on the ACQ factor (i.e. α_{ja}) in the population. In the unidimensional content case, further details on this result can be obtained from Ferrando et al. (2003), and, provided that full within-factor balance applies, the result also holds directly in the multidimensional case. Indeed, the conditions above are only approximately met at best (in particular, true communalities as assumed in the centroid method, are never known; (see McDonald, 1978). And, furthermore, the first centroid loading is a sample estimate. For these reasons, our choice in SIREN is to first obtain the first principal-axis or canonical factor by using an efficient EFA estimation procedure: Minimum Rank Factor Analysis (MRFA, ten Berge and Kiers, 1991), and next rotate this factor against the centroid vector that is used as a criterion (see Eysenck, 1950). The final factor so obtained is taken in SIREN as an estimate of the

ACQ factor. As for the MRFA choice, it is because of its robustness and because it provides estimates of the proportion of common variance accounted for by the different factors, and this information is useful for assessing the relevance of the ACQ factor in terms of explained common variance. The proportion of ACQ explained common variance is provided under the heading 'rACQvariance' in the SIREN package output.

Provided that the preliminary general conditions discussed above are met, the correct functioning of the SIREN procedure once the ACQ factor has been partialled-out depends on two main points. The first point is of a mathematical nature, and is critical if unbiased loadings (especially those of content factors) are to be obtained. The second is of a statistical nature, and is important if the goodness-of-fit assessment needs to be correct.

Obtaining unbiased loading estimates in SIREN mainly depends on achieving full balance (either for the total test or for a core balanced set), in principle, within each factor. In more detail, what is required is that the sum of content loadings within each factor be equal to zero. If it is, then the first centroid (or MRFA factor) will reflect only AR, so partializing it from the correlation matrix will remove only this response bias and leave a 'clean' corrected correlation that will reflect only content. However, if balance is not achieved, then, to a greater or lesser extent, the first centroid factor will reflect a mixture of ACQ and content. So, some content will be removed by partializing, and the resulting content loadings will be biased (see ten Berge, 2020). Having said that, however, we also note that the strict condition of full balance within each factor (or the core set within each factor) is probably too strong. Preliminary results by the writers suggest that, provided that the content CFA solution is correct and full balance holds for the entire set (or core set) of items, but not necessarily for each factor, then SIREN would still perform correctly in most cases. The extent to which the content loading estimates will degrade as imbalance increases is best addressed using simulation, and this will be done in the next section.

We turn now to the second, statistical point. Our proposal can be viewed as a particular application of what (Nunnally, 1978) called an ad-lib factorial process, in which successive factors are fitted to residual matrices by using different methods. We consider our proposal to be indeed legitimate, however, its multi-stage nature necessarily entails a loss of information and efficiency, because the successive estimates are taken as fixed and known, and their uncertainty is not taken into account. In agreement with authors such as (DeCastellarnau and Saris, 2021; Nunnally, 1978; ten Berge, 2020) and the empirical results provided by (Oberski and Satorra, 2013), however, we believe that the impact of the loss of efficiency discussed above on the point estimates and indices of goodness of fit will be relatively minor in practice provided that the proposed solution is correct and the basis conditions are reasonably met. The issue, however, needs to be, and will be, assessed by using simulation.

1.4 Multi-stage approach with fully balanced scales

The fulfillment of the full balance condition is checked automatically by the **siren** package by using the information provided in the 'target' argument (described below), in which the content pattern matrix with the signed dominant loading of each item on its corresponding factor is provided as input by the user. An example of a target would be the first two columns of $\mathbf{\Lambda}$ in table 2. The specific stages in this case are:

Stage 1: for the n test items, the ACQ factor loadings are estimated by: (a) obtaining the direct (canonical) MRFA solution from the inter-item correlation matrix, by specifying $m+1$ factors (b) obtaining the first centroid from the inter-item correlation matrix (to be used as criterion), and (c) rotating the MRFA solution to the position in which the first canonical MRFA factor has maximal congruence with respect to the criterion. Essentially, the process is a target rotation against a single target vector (the first centroid). However, because the canonical MRFA solution is orthogonal, the rotation is univocally defined. Finally, the criterion of maximal congruence is defined in least squares terms: i.e. the first rotated MRFA factor is that is the closest to the target centroid in the least-squares sense.

Stage 2: obtain the corrected (i.e. ACQ free) inter-item residual matrix as $\mathbf{R}_{corr} = \mathbf{R} - \boldsymbol{\alpha}\boldsymbol{\alpha}'$.

The $\mathbf{R}_{corr}$ matrix is, and should be treated as, a residual covariance matrix (not a correlation matrix).

Stage 3: The prescribed CFA solution can be specified and fitted in two alternative ways. The first is to input $\mathbf{R}_{corr}$ specified as a covariance matrix to the SEM program, and request a standardized solution. The output will consist of the standardized content pattern with loadings that are free of ACQ. The second way is to input the raw data to the SEM program, by specifying the prescribed CFA content solution, plus an additional ACQ factor in which all the loadings are specified as fixed and known. In this second specification, the ACQ loadings $\boldsymbol{\alpha}$ obtained in stage 2 cannot be imputed directly because $\mathbf{R}_{corr}$ is a covariance matrix (i.e. unstandardized) while the loadings on $\boldsymbol{\alpha}$ are standardized. The correct values to be imputed are thus those obtained by multiplying each standardized loading on $\boldsymbol{\alpha}$ by the corresponding item standard deviation: $\boldsymbol{\alpha}_x$(scaled)= $\boldsymbol{\alpha}_x \mathbf{sx}$. This scaling transforms the standardized loadings $\boldsymbol{\alpha}$ into unstandardized loadings (e.g. Bollen, 1989). As in the first approach, a standardized solution will next be requested in the output. If this is done, the output will now provide the standardized content pattern with loadings free from ACQ and an additional column containing the standardized ACQ loadings. Indeed, the standardized content pattern must be the same in both specifications. In the pkg siren programming, the CFA in stage 3 is done by using the cfa function from the **lavaan** and one of the two options described above. However, of the multiple choices that the program allows for estimating the structural item content parameters, we have chosen the one that is most congruent with the previous stages. Thus, whether the variables are treated as continuous or discrete, the estimation procedure is robust ULS, in agreement with the choice of MRFA-ULS for obtaining the ACQ estimates.

Stage 4: testing model-data fit at the structural level. Again, we have made choices within **lavaan** that are in accordance with the limited-information nature of the procedure as well as with the results of the simulation study. The chosen indices are: (a) the RMSR and GFI as overall measures of misfit (McDonald and Mok, 1995), (b) the RMSEA as a measure of relative fit with respect to the degrees of freedom (i.e. model complexity), and (c) the CFI as a measure of comparative fit with respect to the null independence model (see Tanaka, 1993, for points b and c). These indices are provided in the SIREN package output under the heading 'rfit_indices'.

Stage 5: obtaining individual score estimates. In the basic FA equation discussed above, the factor score estimates for each individual are the estimates of the 'true' scores $\boldsymbol{\theta}$ in (1), which, of course, are unknown. For both, the linear and the nonlinear models, the factor score estimates are Bayes Modal a Posteriori (MAP), which, in the continuous (linear) model, are known as regression estimates. In both cases, Bayesian scoring provides finite and plausible estimates for all the respondents under study (see Ferrando and Lorenzo-Seva, 2016). For each participant, the output information consists of the point estimate of his/her level on the content factors plus his/her factor score estimate on the ACQ factor. This last estimate can be interpreted as the predisposition of the individual to engage in AR. The factor score estimates are provided under the heading 'rp_factors' (see below).

Stages 1 to 4 in the fully-balanced procedure will be now illustrated with our toy example. Furthermore, the effects of ignoring the secondary ACQ factor will be illustrated by fitting directly the content solution in table 1 to the uncorrected (i.e observed) correlation matrix. To perform the illustration, we generated a random sample of $N = 500$ simulees from a population in which the complete solution in table 2 holds. The results are in table 3.

When an operating common factor is left unmodeled, both, biased structural estimates of the remaining parameters and deterioration of the GOF indicators are expected. So, in general terms, results in table 3 are predictable. Ferrando and Lorenzo-Seva (2010) and DeMars (2014) can be summarized as follows. With regards to bias, SIREN does a good job, and recovers quite acceptably (given both the sample and model size) all the parameters in the toy solution: content loadings, ACQ loadings, and inter-factor correlation. On the other hand, the loading estimates in the uncontrolled solution tend to be slightly more biased, and the inter-factor correlation slightly over-estimated. In terms of GOF, that of the SIREN solution is almost perfect by all standards, which is only to be expected, as the specified

solution is correct. The GOF of the uncorrected solution, however, clearly deteriorates. The amount of misfit is not terribly bad here, which is also expected in such a small model. In a larger-sized example, however, the deterioration of fit would have been much stronger.

Table 3: Toy Example Results

Siren Loadings	Direct Loadings
$\mathbf{\Lambda} = \begin{bmatrix} 0.758 & 0.000 & 0.118 \\ -0.664 & 0.000 & 0.175 \\ 0.620 & 0.000 & 0.311 \\ -0.723 & 0.000 & 0.257 \\ 0.000 & 0.605 & 0.382 \\ 0.000 & -0.583 & 0.272 \\ 0.000 & 0.681 & 0.137 \\ 0.000 & -0.582 & 0.165 \end{bmatrix}$	$\mathbf{\Lambda} = \begin{bmatrix} 0.761 & 0.000 \\ -0.693 & 0.000 \\ 0.631 & 0.000 \\ -0.727 & 0.000 \\ 0.000 & 0.549 \\ 0.000 & -0.552 \\ 0.000 & 0.720 \\ 0.000 & -0.586 \end{bmatrix}$
Siren Phi Matrix	**Direct Phi Matrix**
$\mathbf{\Phi} = \begin{bmatrix} 1 & .33 & 0 \\ .33 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\mathbf{\Phi} = \begin{bmatrix} 1 & .35 \\ .35 & 1 \end{bmatrix}$
Siren GOF	**Direct GOF**
CFI = 1	CFI = .922
GFI = .998	GFI = .974
RMSEA = 0	RMSEA = .088
SRMR = .024	SRMR = .049

Note: GOF=Goodness of Fit Indices

1.5 Multi-stage approach with partially balanced scales

The basic idea in this case is to first obtain a fully balanced sub-set of items, which we shall denote as the core set, and estimate the ACQ loadings of the items belonging to this core by using the procedure described above. Next, the ACQ loadings of the remaining items are estimated by using a type of extension analysis based on the method of moments. Once the ACQ loading estimates are available for all the test items, the rest of the procedure can be carried out exactly as in the fully balanced case. So, the only two points that require specific discussion are: first, how to determine which items will be included in the core set, and second, how the ACQ loadings of the remaining items will be determined.

Stage 1: Choosing the core set. Within each specified factor, the positive and negative items, as specified in the 'target' argument are separated into two groups, and a centroid FA is performed separately in each of the two resulting inter-item correlation matrices. The loadings in the smaller set (usually that containing the negative items) are taken as fixed, and each of them is paired with the positive loading with the most similar value. The aim is for the absolute value of the sums of the positive and negative loadings to be as similar as possible. The rationale is that the effect of ACQ will be in the same direction if items are all worded in the same direction (i. e. both the positive and the negative loadings will be upwardly biased).

Stage 2: For the n_c items in the core subset, the loadings on the ACQ factor are estimated using the procedure described in the fully balanced case.

Stage 3: Denote by X_o an item outside the core set, and let $j = 1, \cdots n_c$ be the items in the core. Let $\sum_{j=1}^{n_c} r_{oj}$ be the sum of the correlations of item X_o with the remaining items in

the core set (see (5) for deriving these correlations). If the core items are balanced, all the terms involving the sum of content loadings will vanish, and it will then follow that then follows that

$$\sum_{j=1}^{n_c} r_{oj} = \alpha_{oa} \sum_{j=1}^{n_c} \alpha_{ja} \tag{4}$$

So

$$\alpha_{oa} = \frac{\sum_{j=1}^{n_c} r_{oj}}{\sum_{j=1}^{n_c} \alpha_{ja}} \tag{5}$$

In words, if full balance holds for the core set, then the quotient between (a) the sum of correlations of item X_o with the remaining items in the core set, and (b) the sum of ACQ loadings in the core set provides a simple estimate of the loading of item X_o on the ACQ factor. Note that the sum in the denominator of (5) is taken as fixed and known and has been obtained in Stage 2 above. The estimate described above can be viewed as an extension-analysis estimate (e.g. McDonald, 1978) obtained by the method of moments.

The extension estimate (5) is computed on an item-by-item basis for each of the items outside the core set in Stage 3. So, at the end of this stage, ACQ loading estimates are available for all the test items under study. This is the same situation as at the end of Stage 1 in the fully-balanced-case approach. Therefore, from this point on, the procedure is the same in both cases.

In closing this section, it is important to note that the **siren** package automatically detects if the scales are only partially balanced through the information provided by the 'target' argument, and, if so, also automatically carries out the three-step procedure described in this section.

2 The siren package details

Available through CRAN, the **siren** package contains one main function (and additional internal functions) called `acquihybrid`, which implements the procedures described in the sections above.

The function usage is the following:

```
acquihybrid(x, content_factors, target, corr = "Pearson", raw_data=TRUE,
method="fixed", display = TRUE)
```

in which the arguments are:

`x`, raw sample item scores or a covariance/correlation matrix, as a data.frame or a numerical matrix,

`content_factors`, the number of content factors in the CFA solution. Each factor has to be defined by at least 3 items,

`target`, the pattern loading target matrix, which provides the signed dominant loading (higher in absolute value) of each item on its corresponding factor. The target is only used as a reference for assessing which items have significant loadings on which factors. The specific loading estimates are not used,

`corr`, determines the type of matrices to be used in the factor analysis. "Pearson": Computes Pearson inter-item correlation matrices (linear FA model); "Polychoric": Computes Polychoric/Tetrachoric inter-item correlation matrices (non-linear FA/graded model),

`raw_data`, logical argument, if TRUE, the entered data will be treated as raw item scores (default). If FALSE, the entered data will be treated as an inter-item covariance/correlation matrix,

`method`, two choices are provided: fixed, which uses the ACQ loadings obtained in the first step to specify the ACQ factor in the CFA solution based on the raw scores, and resid, which uses the ACQ-free corrected covariance matrix as input for the CFA,

`display`, determines if the output will be displayed in the console, TRUE by default. If it is TRUE, the output is printed in the console and if it is FALSE, the output is returned silently to the output variable.

The data provided should be a data frame or a numerical matrix for input vectors and matrices, character variables for `corr` and method arguments, and logical values for `raw_data` and `display` arguments.

The `acquihybrid` function returns a list variable, containing the following variables:

`rloadings`, the factor loadings for each content factor and acquiescence factor.

`rfactor_cor`, content factor correlations.

`rfit_indices`, a sub-list including a variety of popular fit indices as described above.

`rACQ_variance`, the amount of common variance explained by ACQ.

`rresid_matrix`, residual matrix after partialling-out for ACQ.

`rpfactors`, factor scores for each participant.

The package includes a detailed vignette titled "siren-vignette", which provides step-by-step explanations of how to use the package, as well as guidance on interpreting the data. The vignette uses the same dataset as the illustrative example below. The vignette is accessible through:

```
vignette("siren-vignette")
```

3 Simulation studies

To assess the behavior of the proposal under favorable conditions (correct population model) and its robustness against slight misspecifications, we conducted a simulation study which focused on both the recovery of the 'true' loadings in the ACQ and the content factors, and the goodness of fit results.

3.1 Method

A bidimensional content model with an additional ACQ factor (see Equation 1) was generated under the following specifications: (a) all the factors were orthogonal (this choice was made for simplicity); (b) the content factors contained positive and negative loadings (representing the positively and negatively keyed items); and (c) the loadings on the ACQ factor were all positive. Referring to Equation 1, the structure of the simulated model can be understood, as it defines how the observed scores are reproduced from the common factors and unique loadings, facilitating the interpretation of the factor analysis results. The number of items per factor was 10, and the sample size was fixed to 300, a value slightly higher than recommended to find accurate factor loadings estimates (Fabrigar et al., 1999). To help the reader to visually understand the design of the simulation study, we provide a simplified version of the path diagram in fig. 1.

In the content factors, the simulated loadings had an average value of .6 (in absolute value) and a standard deviation of 0.1. For the ACQ factor the mean loading value was .2. The behavior of **siren** was assessed under three general conditions: (1) type of item: ordinal (four categories; FA based on polychoric correlations), or continuous (FA based on Pearson correlations); (2) pattern of substantive loadings at three levels: (a) completely balanced, (b) 60% of positive items, and (c) 70% of positive items; and (3) ACQ pattern at three levels: (a) equal ACQ loadings, (b) low heterogeneity (standard deviation of .01), and (c) high heterogeneity (standard deviation of .1). Thus, a factorial design with 18 experimental conditions (2 x 3 x 3) was used. These conditions were chosen according to (a) the most problematic conditions for the alternative method discussed above, and (b) the degree of realism in the applied context.

For each experimental condition, 200 replicas were generated. A higher number of replicas would simply increase the estimation time without changing the results. All

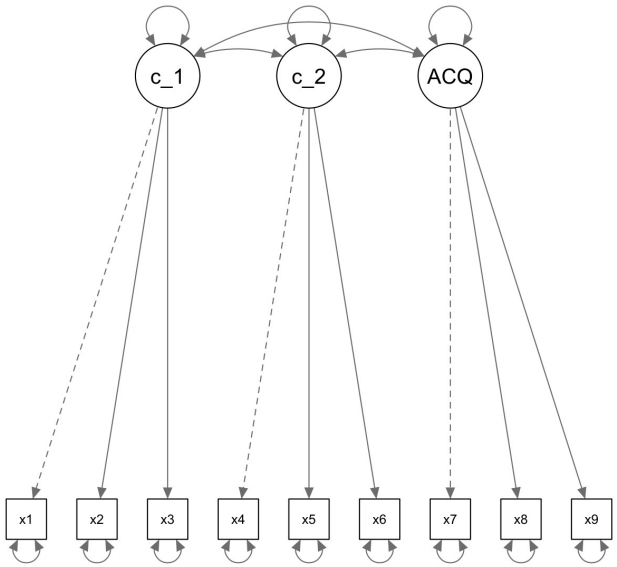


Figure 1: Simplified version of the path diagram of the simulated model.

analyses were conducted with R (R Core Team, 2024). The quality of the estimates was assessed using the average bias (6)

$$\text{Average Bias} = \frac{1}{n}\sum_{i=1}^{n}(\theta_i - \hat{\theta}_i) \tag{6}$$

where θ_i are the observed or true values, $\hat{\theta}_i$ are the predicted or estimated values, and n is the total number of observations; and the consistency of the model-data fit results was assessed using an analysis of variance (ANOVA) for each fit index considered in the study (CFI, GFI, RMSR and RMSEA).

3.2 Results

The results for the average bias were very stable in all conditions (see Tables 4 and 5). The loadings on content factors are recovered more accurately than the loadings on the ACQ factor. In the content factors, the bias is evenly distributed across both factors, and never exceeds .05 (continuous condition) or .04 (ordinal condition) in absolute value. In contrast, the average bias in the ACQ factor is greater in the ordinal case.

No significant changes are observed when imbalance increases. However, a slight increase in bias can be noticed in those conditions in which the ACQ pattern is more heterogeneous. This increase, however, does not substantially impact the average bias of the ACQ factor (table 5). The bias in ACQ remains at around .06, with a maximum of .077.

With regards to the model-data fit ANOVA results, finally, no significant effects were observed. However, when the ACQ patterns are very heterogeneous, the fit of the model tends to worsen, which suggests that **siren** is more sensitive to the pattern of ACQ loadings.

In closing, it should be noted that the simulated data are very favorable: each content factor is strong and well-defined, without the presence of correlated residuals or cross-loadings. In this framework, **siren** barely suffers from a lack of specification or bias when evaluated conditions are degraded. As ACQ loadings are not set to 1 (unlike the confirmatory method of Billiet and McClendon, 2000), these loadings are freely estimated, which is why the heterogeneity of the acquiescence pattern does not affect the estimation results.

Table 4: Average biases of the content factor loadings

		factor 1											factor 2								
Continuous		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Bal	EQ	.047	.044	.045	.047	.045	.043	.045	.042	.04	.041	.044	.043	.042	.046	.045	.046	.041	.049	.042	.044
Bal	LH	.047	.044	.045	.047	.045	.043	.045	.042	.046	.041	.047	.043	.044	.043	.045	.043	.041	.046	.04	.041
Bal	HH	.046	.043	.047	.047	.045	.044	.045	.045	.043	.044	.045	.046	.04	.046	.043	.043	.044	.05	.045	.042
LU	EQ	.04	.044	.047	.044	.043	.046	.044	.042	.043	.043	.04	.043	.045	.045	.045	.042	.043	.044	.042	.043
LU	LH	.044	.046	.043	.044	.044	.043	.046	.043	.043	.045	.045	.041	.041	.039	.043	.042	.043	.043	.043	.042
LU	HH	.043	.043	.038	.043	.043	.046	.043	.042	.046	.05	.046	.044	.045	.044	.042	.044	.042	.043	.041	.039
HU	EQ	.044	.047	.041	.041	.048	.046	.045	.042	.049	.05	.044	.046	.044	.045	.044	.046	.046	.049	.041	.046
HU	LH	.043	.045	.046	.045	.045	.041	.045	.04	.043	.045	.048	.045	.046	.043	.048	.04	.043	.045	.043	.045
HU	HH	.05	.044	.046	.045	.047	.043	.044	.047	.05	.049	.046	.049	.042	.048	.046	.047	.049	.045	.044	.046
Ordinal		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Bal	EQ	.04	.036	.036	.039	.039	.038	.036	.037	.035	.041	.038	.039	.037	.038	.035	.035	.036	.04	.037	.031
Bal	LH	.04	.036	.036	.039	.039	.038	.036	.037	.035	.041	.039	.042	.037	.036	.035	.034	.038	.039	.037	.034
Bal	HH	.036	.036	.036	.038	.034	.037	.037	.037	.036	.036	.037	.035	.036	.04	.04	.037	.037	.038	.038	.037
LU	EQ	.039	.035	.037	.04	.034	.038	.035	.035	.033	.036	.032	.038	.038	.035	.036	.036	.04	.035	.037	.037
LU	LH	.039	.034	.038	.038	.037	.037	.037	.034	.04	.039	.04	.037	.035	.033	.039	.034	.036	.037	.039	.034
LU	HH	.037	.037	.039	.04	.036	.038	.034	.037	.038	.039	.035	.037	.037	.037	.04	.04	.037	.036	.037	.033
HU	EQ	.036	.036	.031	.039	.037	.037	.036	.041	.039	.037	.035	.04	.037	.033	.036	.035	.038	.044	.04	.042
HU	LH	.037	.038	.036	.04	.037	.035	.036	.036	.041	.038	.036	.038	.035	.035	.041	.037	.036	.036	.039	.039
HU	HH	.038	.037	.033	.042	.038	.038	.039	.036	.035	.038	.04	.04	.039	.038	.038	.043	.035	.038	0.037	.038

Note: Bal = Balanced; LU = Low Unbalanced; HU = High Unbalanced, EQ = equal; LH = Low Heterogeneity, HH = High Heteregeneity

ISSN 2073-4859

Table 5: Average biases of the ACQ factor loadings

		factor 1										ACQ factor								
Continuous		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Bal	EQ	.063	.066	.064	.074	.068	.064	.063	.066	.068	.064	.07	.068	.071	.064	.064	.075	.076	.074	.068	.068
Bal	LH	.07	.069	.072	.072	.066	.075	.063	.068	.072	.073	.068	.066	.063	.068	.068	.067	.065	.067	.068	.067
Bal	HH	.063	.057	.058	.06	.061	.061	.064	.062	.061	.06	.061	.063	.065	.065	.064	.064	.062	.063	.065	.066
LU	EQ	.058	.059	.059	.056	.058	.059	.058	.066	.064	.064	.061	.064	.058	.061	.064	.061	.068	.068	.063	.07
LU	LH	.059	.061	.06	.057	.066	.06	.061	.059	.062	.072	.066	.063	.06	.063	.065	.062	.061	.069	.066	.064
LU	HH	.067	.058	.063	.059	.056	.056	.059	.062	.061	.06	.061	.063	.056	.056	.059	.054	.063	.054	.056	.059
HU	EQ	.068	0.071	0.061	0.066	.065	.059	.063	0.062	.07	.07	.072	.066	.064	.067	.077	.076	.072	.066	.068	.066
HU	LH	.063	.058	.066	.063	.064	.061	.068	.071	.068	.075	.071	.068	.069	.063	.071	.068	.074	.064	.068	.067
HU	HH	.062	.062	.064	.068	.059	.062	.063	.067	.071	.065	.066	.071	.056	.056	.071	.065	.071	.067	.067	.059
Ordinal		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Bal	EQ	.069	.075	.072	.067	.074	.075	.079	.076	.074	.073	.071	.077	.078	.076	.069	.077	.074	.068	.074	.071
Bal	LH	.075	.076	.078	.07	.079	.08	.078	.079	.069	.083	.072	.077	.079	.072	.078	.076	.071	.076	.076	.071
Bal	HH	.059	.065	.068	.073	.072	.066	.065	.066	.07	.071	.065	.073	.067	.071	.068	.065	.069	.064	.07	.066
LU	EQ	.069	.062	.067	.07	.069	.072	.073	.073	.076	.07	.07	.066	.069	.068	.071	.072	.079	.072	.071	.072
LU	LH	.073	.061	.074	.071	.074	.073	.068	.076	.073	.065	.069	.076	.065	.073	.073	.074	.075	.072	.072	.075
LU	HH	.064	.069	.071	.065	.072	.072	.074	.068	.073	.076	.067	.068	.063	.068	.067	.07	.078	.06	.069	.064
HU	EQ	.076	.071	.067	.073	.07	.074	.073	.08	.069	.089	.081	.075	.078	.075	.073	.077	.076	.085	.074	.074
HU	LH	.072	.069	.07	.077	.078	.068	.068	.086	.079	.078	.078	.077	.074	.081	.074	.08	.077	.083	.082	.083
HU	HH	.06	.069	.073	.074	.07	.071	.076	.078	.077	.074	.077	.063	.076	.067	.067	.069	.08	.071	.076	.074

Note: Bal = Balanced; LU = Low Unbalanced; HU = High Unbalanced, EQ = equal; LH = Low Heterogeneity, HH = High Heteregeneity

ISSN 2073-4859

4 Illustrative example usage

To illustrate how the SIREN program works, we have used an existing dataset of 1309 participants (55.8% females) between 14 and 19 years old (M = 16.4, S.D. = 1.1) from three previous studies (Morales-Vives and Dueñas, 2018; Morales-Vives et al., 2020, in press). Therefore, further details about this data can be obtained from the original studies. Those participants with missing data were not included in the present illustrative analyses. All participants answered the Psychological Maturity Assessment Scale questionnaire (Morales-Vives et al., 2013, PSYMAS), which assesses the psychological maturity of adolescents, understood as the ability to take responsibility for one's own obligations, taking into account one's own characteristics and needs, without showing excessive dependence on others. It consists of 27 items with a five-point response format (1 = Completely disagree, 5 = Completely agree) and it assesses the following factors: work orientation, self-reliance, and identity. The study carried out by Morales-Vives et al. (2013), shows that (a) the content factors are correlated, and (b) some of the items are affected by the acquiescence response bias. This second feature is the reason why we chose the data from this questionnaire as an illustration of how **siren** works and how its outcomes are to be interpreted. In the current analysis, we have only used ten items from two of the subscales of this questionnaire (four items of self-reliance subscale and six items of identity subscale) so that within each subscale half of the items were in one direction (lack of maturity) and the other half in the opposite direction (high maturity). Self-reliance refers to willingness to take the initiative without allowing others to exercise excessive control, and Identity refers to knowledge about one's characteristics and needs. table 6 shows the contents of the used items. We would note that the dataset is available in the **siren** package, so that the interested reader can run the program and verify the results that are presented below.

The code required for running this illustrative example is the following:

```
psymas_target=cbind(c(-9,-9,0,0,0,9,0,0,9,0),c(0,0,-9,9,-9,0,9,-9,0,9))

acquihybrid(psymas, content_factors = 2, target = psymas_target,
  corr = "Polychoric", raw_data = TRUE, method = "fixed", display = TRUE)
```

Following the procedure explained above, the first step was to estimate the ACQ factor from the fully balanced set of items, in this case treating the variables as discrete (i.e. using the nonlinear model). As can be seen in table 7, the ACQ loading estimates ranged between .001 and .566, and the items with higher ACQ loadings were 3, 9 and 10. These results suggest that several items are affected by ACQ, as was expected, and justify the need to control for this response bias.

Table 6: Loading estimates in the Acquiescence factor obtained in the first step

	ACQ
Item 1. Consult the peer group before buying clothes	.001
Item 2. Friends' opinions determine what is considered wrong	.001
Item 6. Doesn't mind doing different things than friends	.231
Item 9. Facing consequences of one's own mistakes	.206
Item 3. Not showing the true self	.380
Item 4. Feeling accepted and valued	.079
Item 5. Feeling empty	.070
Item 7. Good self-knowledge	.156
Item 8. Others do not really know him/her	.338
Item 10. Feeling capable of doing many things well	.566

Note: ACQ = Acquiescence

We next fitted a CFA solution consisting of two correlated content factors with a full IC structure, in which each item only had a non-zero loading on its own factor, and an

additional ACQ factor in which the corresponding loading was fixed at the estimate obtained in table A (in the ordinal case there is no need to multiply this loading by the standard deviation as this has a unit value). The final ULS estimates for the full solution are in creftab8. As expected, the four items of self-reliance subscale loaded in one factor, and the six items of the identity subscale loaded in the other factor. Inspection of the signs of the loadings suggests that the full condition of balance within each factor is achieved. As for the strength of the content solution, items 2, 6, 9 had loadings of .40 or higher (in absolute value) on the self-reliance factor, while item 1 had the lowest loading, the same result obtained in the study carried out by Morales-Vives et al. (2013). All the items of identity had loadings on this factor higher than .40, being item 5 the item with the highest loading, which, again, agrees with the results by Morales-Vives et al. (2013). Overall, the procedures included in the **siren** program provide the expected results, which are congruent with those reported in the previous study, even though the latter included a greater number of items than in the present study. Furthermore, the correlation between the two factors is .436, as was expected, because the study carried out by Morales-Vives et al. (2013) already showed that these factors are positively correlated.

Table 7: Estimated loadings in the CFA solution

	Factor 1	Factor 2	ACQ
Item 1. Consult the peer group before buying clothes	.29	.00	.001
Item 2. Friends' opinions determine what is considered wrong	.53	.00	.001
Item 6. Doesn't mind doing different things than friends	-.56	.00	.231
Item 9. Facing consequences of one's own mistakes	-.40	.00	.206
Item 3. Not showing the true self	.00	.54	.380
Item 4. Feeling accepted and valued	.00	-.57	.079
Item 5. Feeling empty	.00	.66	.070
Item 7. Good self-knowledge	.00	-.53	.156
Item 8. Others do not really know him/her	.00	.43	.338
Item 10. Feeling capable of doing many things well	.00	-.47	.566

Note: ACQ = Acquiescence

The fit of the solution on table 7 was quite acceptable: GFI=.99, RMSR=.04, RMSEA=0.04, and CFI=0.96. This good fit suggests that, once ACQ is controlled, the structure of the PSYMAS item pool assessed here is remarkably simple and strong.

5 Concluding remarks

There are at present two factor-analytic approaches for calibrating and scoring typical-response measures after controlling for the biasing effects of AR. One of them is fully confirmatory, and the complete solution is identified by fixing all the ACQ loadings to the same value. The other is unrestricted (i.e. exploratory or semi-confirmatory). According to the literature, each of the two approaches has its pros and cons (Savalei and Falk, 2014; de la Fuente and Abad, 2020).

In this article we have proposed a hybrid EFA-CFA procedure, called SIREN, that tries to combine the best features of the two approaches above. Thus, in SIREN, the ACQ factor can be identified in a first step without the need to constrain all its loadings to have the same value. Next, once the ACQ factor is identified, a fully confirmatory (restricted) solution can be specified for the content factors at the second step. Finally, for both types of factors (ACQ and content), our proposed procedure allows factor score estimates for each individual to be obtained at the third step. The flexibility of what we propose widens the available options for assessing the structural properties of the typical-response measure under scrutiny, and also, for obtaining accurate score estimates for each individual. Regarding this last point, we would note that most existing factor-analytical developments designed for controlling

ACQ tend to focus solely on the structural properties of the instrument. However, accurate and "clean" individual score estimates might be highly relevant in further validity studies or if clinical decisions have to be taken on the basis of this instrument.

Apart from increased flexibility, the proposal has many features that considerably increase its range of application. To start with, it allows solutions to be fitted with the standard linear FA model or with the non-linear graded-response model. Second, the solution can be fitted using a "cleaned" residual covariance matrix (the standard approach to this type of problems) or directly fitted to the raw data using the ACQ loading estimates as fixed and known. This second option makes it possible to use a wide range of estimation procedures and goodness of fit measures for estimating and assessing model data fit.

The main theoretical and potential shortcoming of SIREN is the loss of efficiency caused by the sequential limited-information procedure which it uses. So far, the results of the simulation study suggest that this loss has little impact in practice. However, more extensive simulation is warranted. Although SIREN controls acquiescence bias, it is necessary to assess to what extent null or close-to-zero biases are detected.

The R program that implements SIREN (and which has the same name) has been designed to be as user-friendly as possible, and requires very few specifications from the user: essentially, the FA model of choice (linear or nonlinear) and a target matrix, which specifies the content factor on which each item is expected to load together with the expected sign of this loading. So, the program can be used by practitioners with minimal proficiency in FA. Furthermore, **siren** is extremely versatile, and provides a considerable amount of information in an output that is simple and clear to interpret. Even so, we plan to extend the calibration and the scoring choices of the program in future developments.

References

F. M. Andrews. Construct validity and error components of survey measures: A structural modeling approach. *Public opinion quarterly*, 2(48):409–442, 1984. URL https://doi.org/10.1086/268840. [p9]

T. Asparouhov and B. Muthen. Residual structural equation models. *Structural Equation Modeling*, 1(30):1–31, 1984. URL https://doi.org/10.1080/10705511.2022.2074422. [p9]

J. B. Billiet and M. J. McClendon. Modeling acquiescence in measurement models for two balanced sets of items. *Structural equation modeling*, 4(7):608–628, 2000. URL https://doi.org/10.1207/S15328007SEM0704_5. [p6, 9, 15]

K. A. Bollen. A new incremental fit index for general structural equation models. *Sociological methods & research*, 3(17):303–316, 1989. URL https://doi.org/10.1177/0049124189017003004. [p11]

J. de la Fuente and F. J. Abad. Comparing methods for modeling acquiescence in multidimensional partially balanced scales. *Psicothema*, 4(32):590–597, 2020. URL http://10.7334/psicothema2020.96. [p5, 6, 19]

A. DeCastellarnau and W. E. Saris. Correcting correlation and covariance matrices for measurement errors before further analysis. *Structural Equation Modeling: A Multidisciplinary Journal*, 4(28):572–581, 2021. URL https://doi.org/10.1080/10705511.2020.1870229. [p9, 10]

C. E. DeMars. An illustration of the effects of ignoring a secondary factor. *Applied Psychological Measurement*, 38(5):406–409, 2014. URL https://doi.org/10.1177/0146621614529360. [p7, 11]

H. J. Eysenck. Criterion analysis–an application of the hypothetico-deductive method to factor analysis. *Psychological Review*, 1(57):38–53, 1950. URL https://doi.org/10.1037/h0057657. [p9]

L. R. Fabrigar, D. T. Wegener, R. C. MacCallum, and E. J. Strahan. Evaluating the use of exploratory factor analysis in psychological research. *Psychological Methods*, 3(4):272–299, 1999. URL https://doi.org/10.1037/1082-989X.4.3.272. [p14]

P. J. Ferrando and U. Lorenzo-Seva. Acquiescence as a source of bias and model and person misfit: A theoretical and empirical analysis. *British Journal of Mathematical and Statistical Psychology*, 2(62):427–448, 2010. URL https://doi.org/10.1348/000711009X470740. [p7, 9, 11]

P. J. Ferrando and U. Lorenzo-Seva. Unrestricted item factor analysis and some relations with item response theory. Technical report, Department of Psychology, Universitat Rovira i Virgili, Tarragona, 2013. URL http://psico.fcep.urv.es/utilitats/factor. [p9]

P. J. Ferrando and U. Lorenzo-Seva. A note on improving eap trait estimation in oblique factor-analytic and item response theory models. *Psicologica*, 2(37):235–247, 2016. URL https://www.redalyc.org/articulo.oa?id=16946248007. [p11]

P. J. Ferrando, U. Lorenzo-Seva, and E. Chico. Unrestricted factor analytic procedures for assessing acquiescent responding in balanced, theoretically unidimensional personality scales. *Multivariate Behavioral Research*, 3(38):353–374, 2003. URL https://doi.org/10.1207/S15327906MBR3803_04. [p5, 6, 9]

K. G. Joreskog. A general approach to confirmatory maximum likelihood factor analysis. *Psychometrika*, 2(34):183–202, 1969. URL https://doi.org/10.1007/BF02289343. [p7]

D. Lawley. Approximate methods in factor analysis. *British Journal of Statistical Psychology*, 13(1):11–17, 1960. [p9]

U. Lorenzo-Seva and P. J. Ferrando. Acquiescent responding in partially balanced multidimensional scales. *British Journal of Mathematical and Statistical Psychology*, 2(62):319–326, 2009. URL https://doi.org/10.1348/000711007X265164. [p8]

R. P. McDonald. Some checking procedures for extension analysis. *Multivariate Behavioral Research*, 13(3):319–325, 1978. URL https://doi.org/10.1207/s15327906mbr1303_4. [p9, 13]

R. P. McDonald. A basis for multidimensional item response theory. *Applied Psychological Measurement*, 2(24):99–114, 2000. URL https://doi.org/10.1177/01466210022031552. [p7]

R. P. McDonald and M. M. Mok. Goodness of fit in item response models. *Multivariate Behavioral Research*, 1(30):23–40, 1995. URL https://doi.org/10.1207/s15327906mbr3001_2. [p11]

S. Messick. The psychology of acquiescence: an interpretation of research evidence 1. *ETS Research Bulletin Series*, 1966(1):i–44, 1966. [p5]

F. Morales-Vives and J. M. Dueñas. Predicting suicidal ideation in adolescent boys and girls: The role of psychological maturity, personality traits, depression and life satisfaction. *The Spanish journal of psychology*, 21:E10, 2018. URL https://doi.org/10.1017/sjp.2018.12. [p18]

F. Morales-Vives, E. Camps, and U. Lorenzo-Seva. Development and validation of the psychological maturity assessment scale (PSYMAS). *European Journal of Psychological Assessment*, 1(29):12–18, 2013. URL https://doi.org/10.1027/1015-5759/a000115. [p18, 19]

F. Morales-Vives, E. Camps, and J. M. Dueñas. Predicting academic achievement in adolescents: The role of maturity, intelligence and personality. *Psicothema*, 1(31):84–91, 2020. URL https://doi.org/10.1027/1015-5759/a000115. [p18]

F. Morales-Vives, P. Ferrando, J. M. Dueñas, S. Martín-Arbós, and E. Castarlenas. Are older teens more frustrated than younger teens by the COVID-19 restrictions? the role of psychological maturity, personality traits, depression and life satisfaction. *Current Psychology*, in press. URL https://doi.org/10.1007/s12144-023-04317-6. [p18]

B. Muthen. Goodness of fit with categorical and other non-normal variables. In K. Bollen and S. J. Long, editors, *Testing Structural Equation Models*, pages 205–243. Sage Publications, 1993. [p9]

J. C. Nunnally. An overview of psychological measurement. In B. Wolman, editor, *Clinical diagnosis of mental disorders*, pages 97–146. Springer, 1978. [p6, 10]

D. L. Oberski and A. Satorra. Measurement error models with uncertainty about the error variance. *Structural Equation Modeling: A Multidisciplinary Journal*, 3(20):409–428, 2013. URL https://doi.org/10.1080/10705511.2013.797820. [p10]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024. URL https://www.R-project.org/. ISBN 3-900051-07-0. [p15]

V. Savalei and C. F. Falk. Recovering substantive factor loadings in the presence of acquiescence bias: A comparison of three approaches. *Multivariate behavioral research*, 5(49): 407–424, 2014. URL https://doi.org/10.1037/1082-989X.4.3.272. [p5, 6, 19]

J. S. Tanaka. An overview of psychological measurement. In K. Bollen and S. J. Long, editors, *Testing Structural Equation Models*, pages 10–40. Sage Publications, 1993. [p11]

J. M. F. ten Berge. A legitimate case of component analysis of ipsative measures, and partialling the mean as an alternative to ipsatization. *Multivariate Behavioral Research*, 4 (34):89–102, 2020. URL https://doi.org/10.1207/s15327906mbr3401_4. [p9, 10]

J. M. F. ten Berge and H. A. L. Kiers. A numerical approach to the approximate and the exact minimum rank of a covariance matrix. *Psychometrika*, 2(56):309–215, 1991. URL https://doi.org/10.1007/BF02294464. [p9]

A. Vigil-Colet, D. Navarro-Gonzalez, and F. Morales-Vives. To reverse or to not reverse likert-type items: That is the question. *Psicothema*, 1(32):108–114, 2020. URL https://doi.org/10.7334/psicothema2019.286. [p5]

David Navarro-Gonzalez
Department of Psychology
University of Lleida
Spain
0000-0002-9843-5058
david.navarro@udl.cat

Pere J. Ferrando
Department of Psychology
University Rovira i Virgili
Spain
0000-0002-3133-5466
perejoan.ferrando@urv.cat

Fabia Morales-Vives
Department of Psychology
University Rovira i Virgili
Spain

0000-0002-2095-0244
fabia.morales@urv.cat

Ana Hernandez-Dorado
Department of Psychology
University Rovira i Virgili
Spain
0000-0001-9502-9735
ahernandezd@professor.universidadviu.com

Random Forests for Time-Fixed and Time-Dependent Predictors: The DynForest R Package

by Anthony Devaux, Cécile Proust-Lima, and Robin Genuer

Abstract The R package DynForest implements random forests for predicting a continuous, a categorical, or a (multiple causes) time-to-event outcome based on time-fixed and time-dependent predictors. The main originality of DynForest is that it handles time-dependent predictors that can be endogenous (i.e., impacted by the outcome process), measured with error, and measured at subject-specific times. At each recursive step of the tree building process, the time-dependent predictors are internally summarized into individual features on which the split can be done. This is achieved using flexible linear mixed models (thanks to the R package lcmm), whose specification is pre-specified by the user. DynForest returns the mean for a continuous outcome, the category with a majority vote for a categorical outcome, or the cumulative incidence function over time for a survival outcome. DynForest also computes variable importance and minimal depth to inform on the most predictive variables or groups of variables. This paper aims to guide the user with step-by-step examples for fitting random forests using DynForest.

1 Introduction

Random forests are a non-parametric, powerful method for prediction purposes. Introduced by Breiman (Breiman, 2001) for classification (categorical outcome) and regression (continuous outcome) frameworks, random forests are particularly designed to tackle modeling issues in high-dimensional contexts ($n << p$). They can also easily take into account complex associations between the outcome and the predictors without any pre-specification, where regression models are rapidly limited.

Recently, this methodology was extended to survival data (Ishwaran et al., 2008) and competing events (Ishwaran et al., 2014). Random forests were implemented in several R (R Core Team, 2019) packages such as **randomForestSRC** (Ishwaran and Kogalur, 2022), **ranger** (Wright and Ziegler, 2017) or **xgboost** (Chen and Guestrin, 2016), among others. However, these packages are all limited to time-fixed predictors. Yet, in many applications, it may be relevant to include predictors that are repeatedly measured at multiple occasions (regular or irregular times) with measurement errors to more accurately predict the outcome. This is the case, in particular, in health research where a health outcome is to be predicted according to the history of individual information.

We developed an original random forests methodology to tackle this issue and incorporate longitudinal predictors that may be prone to error and possibly intermittently measured (Devaux et al., 2023). The present paper aims to describe the **DynForest** R package associated with this methodology, allowing the prediction of a continuous, categorical, or survival outcome using multivariate time-dependent predictors.

In section 2, we briefly present **DynForest** methodology through its algorithm. In section 3, we present the different functions of **DynForest**, and we illustrate them in section 4 for a survival outcome and in section 5 for a categorical outcome. An illustration for a continuous outcome can be found in the **DynForest** vignette. To conclude, we discuss in section 6 the limitations and future improvements.

2 DynForest principle

DynForest is a random forest methodology which can include both time-fixed predictors of any nature and time-dependent predictors possibly measured at irregular times. The

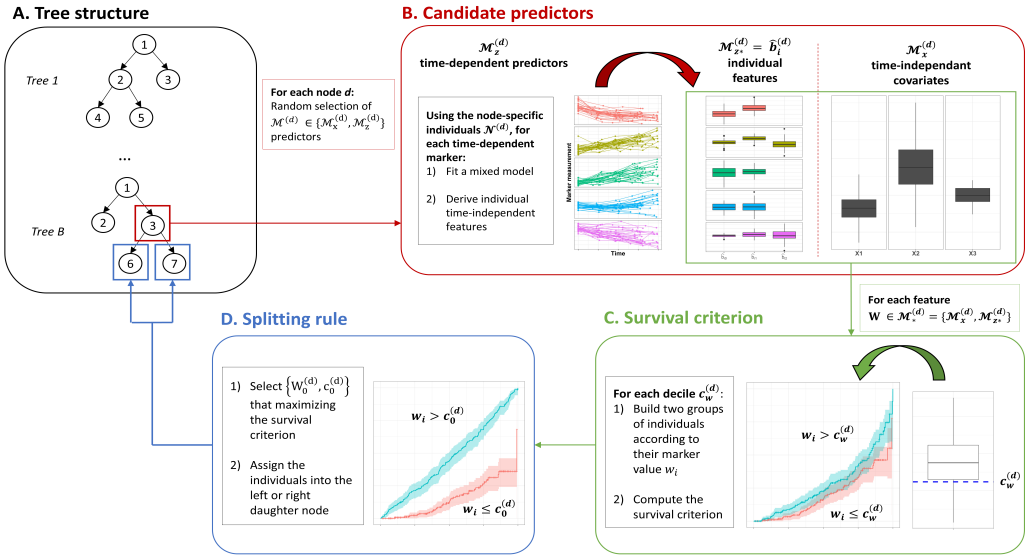


Figure 1: Overall scheme of the tree building in DynForest with (A) the tree structure, (B) the node-specific treatment of time-dependent predictors to obtain time-fixed features, (C) the dichotomization of the time-fixed features, (D) the splitting rule.

purpose of **DynForest** is to predict an outcome which can be categorical, continuous or survival (with possibly competing events).

The random forest should first be built on a learning dataset of N subjects including: Y the outcome; $\mathcal{M}_x$ an ensemble of P time-fixed predictors; $\mathcal{M}_z$ an ensemble of Q time-dependent predictors. The random forest consists of an ensemble of B trees which are grown as detailed below.

2.1 The tree building

The tree building process, summarized in figure 1, aims to recursively partition the subjects into groups/nodes that are the most homogeneous regarding the outcome Y.

For each tree b $(b = 1, ..., B)$, we first draw a bootstrap sample from the original dataset of N subjects (N draws among the N subjects with replacement). The subjects excluded by the bootstrap constitute the out-of-bag (OOB) sample, noted OOB^b for tree b. At each node $d \in \mathcal{D}^b$ of the tree, we recursively repeat the following steps using the $N^{(d)}$ subjects located at node d:

1. An ensemble of $\mathcal{M}^{(d)} = \{\mathcal{M}_x^{(d)}, \mathcal{M}_z^{(d)}\}$ candidate predictors is randomly selected among $\{\mathcal{M}_x, \mathcal{M}_z\}$ (see figure 1B). The size of $\mathcal{M}^{(d)}$ is defined by the hyperparameter $mtry$.

2. For each time-dependent predictor in $\mathcal{M}_z^{(d)}$:

    a. We independently model the trajectory of the predictor using a flexible linear mixed model (Laird and Ware, 1982) according to time (the specification of the model is defined by the user). It is defined as: $Z_{ij} = X1_{ij}(t_{ij})\beta + X2_{ij}(t_{ij})b_i + \epsilon_{ij}$ where Z_{ij} is the value of predictor Z for subject i at occasion j, $X1_{ij}$ and $X2_{ij}$ are vectors of time functions and covariates to be specified by the user and t_{ij} is the time at occasion j for subject i. β are population effects, and b_i individual random effects which follow a multivariate normal distribution. ϵ_{ij} are the zero-mean independent Gaussian errors of measurement.
    b. We predict the vector of random-effects $\hat{b}_i$ using the available information of individual $i = 1, ..., N^{(d)}$. Predictions $\hat{b}_i$ constitute time-independent features

summarizing each time-dependent predictor. We thus derive the ensemble $\mathcal{M}_{z\star}^{(d)}$ for all the variables in $\mathcal{M}_z^{(d)}$.

3. We define $\mathcal{M}_\star^{(d)} = \{\mathcal{M}_x^{(d)}, \mathcal{M}_{z\star}^{(d)}\}$ as our new ensemble of time-independent candidate features.

4. For each candidate feature $W \in \mathcal{M}_\star^{(d)}$:

   a. We build a series of splits $c_W^{(d)}$ according to the feature values if continuous, or subsets of categories otherwise (see figure 1C), leading each time to two groups.
   b. We quantify the distance between the two groups according to the nature of Y:
      - If Y continuous: we compute the weighted within-group variance with the proportion of subjects in each group as weights
      - If Y categorical: we compute the weighted within-group Shannon entropy (Shannon, 1948) (i.e., the amount of uncertainty) with the proportion of subjects in each group as weights
      - If Y survival without competing events: we compute the log-rank statistic test (Peto and Peto, 1972)
      - If Y survival with competing events: we compute the Fine & Gray statistic test (Gray, 1988)

5. We split the subjects into the two groups that minimize (for continuous and categorical outcome) or maximize (for survival outcome) the quantity defined previously. We denote $\{W_0^d, c_0^d\}$ as the optimal couple used to split the subjects and assign them to the left and right daughter nodes $2d$ and $2d+1$, respectively (see figure 1D and A).

6. Step 1 to 5 are iterated on the daughter nodes until stopping criteria are met.

We define two stopping criteria: `nodesize`, the minimal number of subjects in a node required to reiterate the split, and `minsplit`, the minimal number of events required to split the node. `minsplit` is only defined with a survival outcome. In the following, we call leaves the terminal nodes.

In each leaf $h \in \mathcal{H}^b$ of tree b, a summary π^{h^b} is computed using the individuals belonging to the leaf. The leaf summary is defined according to the outcome:

- the mean, for Y continuous
- the category with the highest probability, for Y categorical
- the cumulative incidence function over time computed using the Nelson-Aalen cumulative hazard function estimator (Nelson, 1969; Aalen, 1976), for Y single cause time-to-event
- the cumulative incidence function over time computed using the non-parametric Aalen-Johansen estimator (Aalen and Johansen, 1978), for Y time-to-event with multiple causes

NOTE: Step 2 involves the estimation of a parametric mixed model for each time-dependent predictor. The specification of this model should be carefully determined in preliminary analyses by the user, keeping in mind that there should be a trade-off between goodness-of-fit and parameter parsimony. The critical point is to adequately specify the trajectory shape over time at the population and individual levels. Different bases of time functions can be considered (e.g., fractional polynomials, splines, adhoc) and compared in terms of fit.

2.2 Individual prediction of the outcome

Out-Of-Bag individual prediction

The overall OOB prediction $\hat{\pi}_\star$ for a subject $\star$ can be computed by averaging the tree-based predictions of $\star$ over the random forest as follows:

$$\hat{\pi}_\star = \frac{1}{|\mathcal{O}_\star|} \sum_{b \in \mathcal{O}_\star} \hat{\pi}^{h^b_\star} \tag{1}$$

where $\mathcal{O}_\star$ is the ensemble of trees where $\star$ is *OOB* and $|\mathcal{O}_\star|$ denotes its cardinality. The prediction $\hat{\pi}^{h^b_\star}$ is obtained by dropping down subject $\star$ along tree b. At each node $d \in \mathcal{D}^b$, the subject $\star$ is assigned to the left or right node according to his/her data and the optimal couple $\{W^d_0, c^d_0\}$. W^d_0 is a random-effect feature, its value for $\star$ is predicted from the individual repeated measures using the estimated parameters from the linear mixed model.

Individual dynamic prediction from a landmark time

With a survival outcome, the OOB prediction described in the previous paragraph can be extended to compute the individual probability of event from a landmark time s by exploiting the repeated measures of subject $\star$ only until s. For a new subject $\star$, we thus define the individual probability of event, noted $\hat{\pi}_\star(s, s+t)$, as the probability of experiencing the event by time $s+t$ given the information prior to landmark time s:

$$\hat{\pi}_\star(s, s+t) = \frac{1}{B} \sum_{b=1}^{B} \hat{\pi}^{h^b_\star}(s, s+t) \tag{2}$$

where $\hat{\pi}^{h^b_\star}(s, s+t)$ is the tree-based probability of event at time $s+t$ computed by dropping down $\star$ along the tree by considering longitudinal predictors collected until s and time-fixed predictors. Note that any horizon t can be considered provided $s+t$ remains in the time window on which the random forest was trained. In the **DynForest** package, by default, the probability is computed at all the observed event times after the landmark time.

2.3 Out-Of-Bag prediction error

Using the OOB individual predictions, an OOB prediction error can be internally assessed. The OOB prediction error quantifies the difference between the observed and the predicted values. It is defined according to the nature of Y as:

- for Y continuous, the mean square error (MSE) defined by:

$$errOOB = \frac{1}{N} \sum_{i=1}^{N} (\hat{\pi}_i - \pi^0_i)^2 \tag{3}$$

- for Y categorical, the misclassification error defined by:

$$errOOB = \frac{1}{N} \sum_{i=1}^{N} 1_{(\hat{\pi}_i \neq \pi^0_i)} \tag{4}$$

- for Y survival, the Integrated Brier Score (IBS) (Sène et al., 2016) between τ_1 and τ_2 defined by:

$$errOOB = \int_{\tau_1}^{\tau_2} \frac{1}{N} \sum_{i=1}^{N} \hat{\omega}_i(t) \Big\{ I(T_i \leq t, \delta_i = k) - \hat{\pi}_{ik}(t) \Big) \Big\}^2 dt \tag{5}$$

with T the time-to-event, k the cause of interest and $\hat{\omega}(t)$ the estimated weights using Inverse Probability of Censoring Weights (IPCW) technique that accounts for censoring (Gerds and Schumacher, 2006).

The OOB error of prediction is used, in particular, to tune the random forest by determining the hyperparameters (i.e., `mtry`, `nodesize` and `minsplit`) which give the smallest OOB prediction error.

2.4 Explore the most predictive variables

Variable importance

The variable importance (VIMP) measures the loss of predictive performance (Ishwaran et al., 2008) when removing the link between a predictor and the outcome. The link is removed by permuting the predictor values at the subject level for time-fixed predictors or at the observation level for time-dependent predictors. A large VIMP value indicates a good predictive ability for the predictor.

However, in the case of correlated predictors, the VIMP may not properly quantify the variable importance (Gregorutti et al., 2017) as the information of the predictor may still be present. To better handle situations with highly correlated predictors, the grouped variable importance (gVIMP) can be computed indirectly. It consists of simultaneously evaluating the importance of a group of predictors defined by the user. The computation is the same as for the VIMP except the permutation is performed simultaneously on all the predictors of the group. A large gVIMP value indicates a good predictive ability for the group of predictors.

Minimal depth

The minimal depth is another statistic to quantify the importance of a variable. It assesses the distance between the root node and the first node for which the predictor is used to split the subjects (1 for first level, 2 for second level, 3 for third level, ...). This statistic can be computed at the predictor level or at the feature level, allowing us to fully understand the tree building process.

We strongly advise computing the minimal depth with the `mtry` hyperparameter chosen at its maximum to ensure that all predictors are systematically among candidate predictors for splitting the subjects.

3 The DynForest R package

DynForest methodology was implemented in the R package **DynForest** (Devaux, 2024) freely available on The Comprehensive R Archive Network (CRAN) to users.

The package includes two main functions: `dynforest()` and `predict()` for the learning and the prediction steps. These functions are fully described in section 3.1 and 3.2. Other functions available are briefly described in the table below. These functions are illustrated in examples, one for a survival outcome and one for a categorical outcome.

Function	Description
Learning and prediction steps	
`dynforest()`	Function that builds the random forest
`predict()`	Function for S3 class `dynforest` predicting the outcome on new subjects using the individual-specific information
Assessment function	

Function	Description
`compute_ooberror()`	Function that computes the Out-Of-Bag error to be minimized to tune the random forest
Exploring functions	
`compute_vimp()`	Function that computes the importance of variables
`compute_gvimp()`	Function that computes the importance of a group of variables
`compute_vardepth()`	Function that extracts information about the tree building process
plot() functions for S3 class:	
`dynforest`	Plot the estimated CIF for given tree nodes or subjects
`dynforestpred`	Plot the predicted CIF for the cause of interest for given subjects
`dynforestvimp`	Plot the importance of variables by value or percentage
`dynforestgvimp`	Plot the importance of a group of variables by value or percentage
`dynforestvardepth`	Plot the minimal depth by predictors or features
Other functions	
`summary()`	Function for class S3 `dynforest` or `dynforestoob` displaying information about the type of random forest, predictors included, parameters used, Out-Of-Bag error (only for `dynforestoob` class) and brief summaries about the leaves
`print()`	Function to print object of class `dynforest`, `dynforestoob`, `dynforestvimp`, `dynforestgvimp`, `dynforestvardepth` and `dynforestpred`
`get_tree()`	Function that extracts the tree structure for a given tree
`get_treenode()`	Function that extracts the terminal node identifiers for a given tree

3.1 `dynforest()` function

`dynforest()` is the function to build the random forest. The call of this function is:

```
dynforest(
  timeData = NULL, fixedData = NULL, idVar = NULL,
  timeVar = NULL, timeVarModel = NULL, Y = NULL,
  ntree = 200, mtry = NULL, nodesize = 1, minsplit = 2, cause = 1,
  nsplit_option = "quantile", ncores = NULL,
  seed = 1234, verbose = TRUE
)
```

Arguments

`timeData` is an optional argument that contains the dataframe in longitudinal format (i.e., one observation per row) for the time-dependent predictors. In addition to time-dependent predictors, this dataframe should include a unique identifier and the measurement times. This

argument is set to NULL if no time-dependent predictor is included. Argument fixedData contains the dataframe in wide format (i.e., one subject per row) for the time-fixed predictors. In addition to time-fixed predictors, this dataframe should also include the same identifier as used in timeData. This argument is set to NULL if no time-fixed predictor is included. Argument idVar provides the name of the identifier variable included in timeData and fixedData dataframes. Argument timeVar provides the name of the time variable included in timeData dataframe. Argument timeVarModel contains as many lists as time-dependent predictors defined in timeData to specify the structure of the mixed models assumed for each predictor. For each time-dependent predictor, the list should contain a fixed and a random argument to define the formula of a mixed model to be estimated with **lcmm** R package (Proust-Lima et al., 2017). fixed defines the formula for the fixed-effects and random for the random-effects (e.g., list(Y1 = list(fixed = Y1 ~ time, random = ~ time)). Argument Y contains a list of two elements type and Y. Element type defines the nature of the outcome (surv for survival outcome with possibly competing causes, numeric for continuous outcome and factor for categorical outcome) and element Y defines the dataframe which includes the identifier (same as in timeData and fixedData dataframes) and outcome variables.

Arguments ntree, mtry, nodesize and minsplit are the hyperparameters of the random forest. Argument ntree controls the number of trees in the random forest (200 by default). Argument mtry indicates the number of variables randomly drawn at each node (square root of the total number of predictors by default). Argument nodesize indicates the minimal number of subjects allowed in the leaves (1 by default). Argument minsplit controls the minimal number of events required to split the node (2 by default).

For a survival outcome, argument cause indicates the event of interest. Argument nsplit_option indicates the method to build the two groups of individuals at each node. By default, we build the groups according to deciles (quantile option) but they could be built according to random values (sample option).

Argument ncores indicates the number of cores used to grow the trees in parallel mode. By default, we set the number of cores of the computer minus 1. Argument seed specifies the random seed. It can be fixed to replicate the results. Argument verbose allows to display a progression bar during the execution of the function.

Values

dynforest() function returns an object of class dynforest containing several elements:

- data a list with longitudinal predictors (Longitudinal element), continuous predictors (Numeric element) and categorical predictors (Factor element)
- rf is a dataframe with one column per tree containing a list with several elements, which includes:
  - leaves the leaf identifier for each subject used to grow the tree
  - idY the identifiers for each subject used to grow the tree
  - V_split the split summary (more detailed below)
  - Y_pred the estimated outcome in each leaf
  - model_param the estimated parameters of the mixed model for the longitudinal predictors used to split the subjects at each node
  - Ytype, hist_nodes, Y, boot and Ylevels internal information used in other functions
- type the nature of the outcome
- times the event times (only for survival outcome)
- cause the cause of interest (only for survival outcome)
- causes the unique causes (only for survival outcome)
- Inputs the list of predictors names for Longitudinal (longitudinal predictor), Continuous (continuous predictor) and Factor (categorical predictor)

- `Longitudinal.model` the mixed model specification for each longitudinal predictor
- `param` a list of hyperparameters used to grow the random forest
- `comput.time` the computation time

The main information returned by `rf` is the `V_split` element which can also be extracted using the `get_tree()` function. This element contains a table sorted by the node/leaf identifier (`id_node` column) with each row representing a node/leaf. Each column provides information about the splits:

- `type`: the nature of the predictor (`Longitudinal` for longitudinal predictor, `Numeric` for continuous predictor or `Factor` for categorical predictor) if the node was split, `Leaf` otherwise;
- `var_split`: the predictor used for the split defined by its order in `timeData` and `fixedData`;
- `feature`: the feature used for the split defined by its position in random statistic;
- `threshold`: the threshold used for the split (only with `Longitudinal` and `Numeric`). No information is returned for `Factor`;
- `N`: the number of subjects in the node/leaf;
- `Nevent`: the number of events of interest in the node/leaf (only with survival outcome);
- `depth`: the depth level of the node/leaf.

Additional information about the dependencies

`dynforest()` function internally calls other functions from related packages to build the random forest:

- `hlme()` function (from **lcmm** package (Proust-Lima et al., 2017)) to fit the mixed models for the time-dependent predictors defined in `timeData` and `timeVarModel` arguments
- `Entropy()` function (from base package) to compute the Shannon entropy
- `survdiff()` function (from **survival** package (Therneau, 2022)) to compute the log-rank statistic test
- `crr()` function (from **cmprsk** package (Gray, 2020)) to compute the Fine & Gray statistic test

3.2 `predict()` function

`predict()` is the S3 function for class `dynforest` to predict the outcome on new subjects. Landmark time can be specified to consider only longitudinal data collected up to this time to compute the prediction. The call of this function is:

```
predict(object, timeData = NULL, fixedData = NULL, idVar, timeVar, t0 = NULL)
```

Arguments

Argument `object` contains a `dynforest` object resulting from the `dynforest()` function. Argument `timeData` contains the dataframe in longitudinal format (i.e., one observation per row) for the time-dependent predictors of new subjects. In addition to time-dependent predictors, this dataframe should also include a unique identifier and the time measurements. This argument can be set to `NULL` if no time-dependent predictor is included. Argument `fixedData` contains the dataframe in wide format (i.e., one subject per row) for the time-fixed predictors of new subjects. In addition to time-fixed predictors, this dataframe should also include a unique identifier. This argument can be set to `NULL` if no time-fixed predictor is included. Argument `idVar` provides the name of the identifier variable included in `timeData` and `fixedData` dataframes. Argument `timeVar` provides the name of the time-measurement variable included in `timeData` dataframe. Argument `t0` defines the landmark time; only the longitudinal data collected up to this time are to be considered. This argument should be set to `NULL` to include all longitudinal data.

Values

`predict()` function returns several elements:

- `t0` the landmark time defined in the argument (`NULL` by default)
- `times` times used to compute the individual predictions (only with survival outcome). The times are defined according to the time-to-event subjects used to build the random forest.
- `pred_indiv` the predicted outcome for the new subject. With survival outcome, predictions are provided for each time defined in the `times` element.
- `pred_leaf` a table giving for each tree (in column) the leaf in which each subject is assigned (in row)
- `pred_indiv_proba` the proportion of the trees leading to the category prediction for each subject (only with categorical outcome)

4 How to use DynForest R package with a survival outcome?

4.1 Illustrative dataset: pbc2 dataset

The `pbc2` dataset (Murtaugh et al., 1994) is loaded with the package **DynForest** to illustrate its functionalities. `pbc2` data come from a clinical trial conducted by the Mayo Clinic between 1974 and 1984 to treat primary biliary cholangitis (PBC), a chronic liver disease. A total of 312 patients were enrolled in a clinical trial to evaluate the effectiveness of D-penicillamine compared to a placebo to treat PBC and were followed until the clinical trial ended, leading to a total of 1945 observations. During the follow-up, several clinical continuous markers were collected over time such as: the level of serum bilirubin (`serBilir`), the level of serum cholesterol (`serChol`), the level of albumin (`albumin`), the level of alkaline (`alkaline`), the level of aspartate aminotransferase (`SGOT`), platelets count (`platelets`), and the prothrombin time (`prothrombin`). Four non-continuous time-dependent predictors were also collected: the presence of ascites (`ascites`), the presence of hepatomegaly (`hepatomegaly`), the presence of blood vessel malformations in the skin (`spiders`), and the edema levels (`edema`). These time-dependent predictors were recorded according to the `time` variable. In addition to these time-dependent predictors, a few predictors were collected at enrollment: sex (`sex`), age (`age`), and the drug treatment (`drug`). During the follow-up, 140 patients died before transplantation, 29 patients were transplanted, and 143 patients were censored alive (`event`). The time of the first event (censored alive or any event) was considered as the event time (`years`)

```
library("DynForest")
data(pbc2)
pbc2[1:5, c(
  "id", "time", "serBilir", "SGOT", "albumin", "alkaline",
  "age", "drug", "sex", "years", "event"
)]
```

```
#>    id      time serBilir  SGOT albumin alkaline      age      drug    sex
#> 1   1 0.0000000     14.5 138.0    2.60     1718 58.76684 D-penicil female
#> 2   1 0.5256817     21.3   6.2    2.94     1612 58.76684 D-penicil female
#> 3  10 0.0000000     12.6 147.3    2.74      918 70.56182   placebo female
#> 4 100 0.0000000      2.3 178.3    3.00      746 51.47027   placebo   male
#> 5 100 0.4681853      2.5 189.1    2.94      836 51.47027   placebo   male
#>       years event
#> 1 1.0951703     2
#> 2 1.0951703     2
#> 3 0.1396342     2
#> 4 1.5113350     2
```

```
#> 5 1.5113350     2
```

For the illustration, 4 time-dependent predictors (`serBilir`, `SGOT`, `albumin` and `alkaline`) and 3 predictors measured at enrollment (`sex`, `age` and `drug`) were considered. We aim to predict death without transplantation in patients suffering from primary biliary cholangitis (PBC) using clinical and socio-demographic predictors, considering transplantation as a competing event.

4.2 Data management

To begin, we split the subjects into two datasets: (i) one dataset to train the random forest using 2/3 of patients; (ii) one dataset to predict on the other 1/3 of patients. The random seed is set to 1234 for replication purposes.

```
set.seed(1234)
id <- unique(pbc2$id)
id_sample <- sample(id, length(id) * 2 / 3)
id_row <- which(pbc2$id %in% id_sample)
pbc2_train <- pbc2[id_row, ]
pbc2_pred <- pbc2[-id_row, ]
```

Then, we build the dataframe `timeData_train` in the longitudinal format (i.e., one observation per row) for the longitudinal predictors including: `id` the unique patient identifier; `time` the observed time measurements; `serBilir`, `SGOT`, `albumin` and `alkaline` the longitudinal predictors. We also build the dataframe `fixedData_train` with the time-fixed predictors including: `id` the unique patient identifier; `age`, `drug` and `sex` predictors measured at enrollment. The nature of each predictor needs to be properly defined with the `as.factor()` function for categorical predictors (e.g., `drug` and `sex`).

```
timeData_train <- pbc2_train[,
  c("id", "time", "serBilir", "SGOT", "albumin", "alkaline")
]
fixedData_train <- unique(pbc2_train[, c("id", "age", "drug", "sex")])
```

4.3 Specification of the models for the time-dependent predictors

The first step of the random forest building consists of specifying the mixed model of each longitudinal predictor through a list containing the fixed and random formula for the fixed effect and random effects of the mixed models, respectively. Here, we assume a linear trajectory for `serBilir`, `albumin` and `alkaline`, and a quadratic trajectory for `SGOT`. Although we restricted this example to linear and quadratic functions of time, we note that any function can be considered, including splines.

```
timeVarModel <- list(
  serBilir = list(fixed = serBilir ~ time, random = ~time),
  SGOT = list(fixed = SGOT ~ time + I(time^2), random = ~ time + I(time^2)),
  albumin = list(fixed = albumin ~ time, random = ~time),
  alkaline = list(fixed = alkaline ~ time, random = ~time)
)
```

For this illustration, the outcome object contains a list with `type` set to `surv` (for survival data) and `Y` contains a dataframe in wide format (one subject per row) with: `id` the unique patient identifier; `years` the time-to-event data; `event` the event indicator.

```
Y <- list(type = "surv", Y = unique(pbc2_train[, c("id", "years", "event")]))
```

4.4 Random forest building

We build the random forest using `dynforest()` function with the following code:

```
res_dyn <- dynforest(
  timeData = timeData_train,
  fixedData = fixedData_train,
  timeVar = "time", idVar = "id",
  timeVarModel = timeVarModel, Y = Y,
  ntree = 200, mtry = 3, nodesize = 2, minsplit = 3,
  cause = 2, ncores = 7, seed = 1234
)
```

In a survival context with multiple events, it is necessary to specify the event of interest with the argument `cause`. We thus fix `cause` = 2 to specify the event of interest (i.e., the death event). For the hyperparameters, we arbitrarily chose `mtry` = 3, `nodesize` = 2, and `minsplit` = 3, and we will discuss this point in section 4.8.

Overall information about the random forest can be output with the `summary()` function as displayed below for our example:

```
summary(res_dyn)
```

```
#> dynforest executed for survival (competing risk) outcome
#>  Splitting rule: Fine & Gray statistic test
#>  Out-of-bag error type: Integrated Brier Score
#>  Leaf statistic: Cumulative incidence function
#> ----------------
#> Input
#>  Number of subjects: 208
#>  Longitudinal: 4 predictor(s)
#>  Numeric: 1 predictor(s)
#>  Factor: 2 predictor(s)
#> ----------------
#> Tuning parameters
#>  mtry: 3
#>  nodesize: 2
#>  minsplit: 3
#>  ntree: 200
#> ----------------
#> ----------------
#> dynforest summary
#>  Average depth per tree: 6.62
#>  Average number of leaves per tree: 27.68
#>  Average number of subjects per leaf: 4.78
#>  Average number of events of interest per leaf: 1.95
#> ----------------
#> Computation time
#>  Number of cores used: 7
#>  Time difference of 3.18191 mins
#> ----------------
```

We executed the `dynforest()` function for a survival outcome with competing events. In this mode, we use the Fine & Gray statistic test as the splitting rule and the cumulative incidence function (CIF) as the leaf statistic. To build the random forest, we included 208 subjects with 4 longitudinal (`Longitudinal`), 1 continuous (`Numeric`) and 2 categorical (`Factor`) predictors. The `summary()` function returns some statistics about the trees. For

instance, we have on average 4.8 subjects and 1.9 death events per leaf. The number of subjects per leaf should always be higher than the `nodesize` hyperparameter. OOB error should first be computed using the `compute_ooberror()` function (see section 4.5) to be displayed on summary output.

To further investigate the tree structure, the split details can be output using the `get_tree()` function with the following code (for tree 1):

```
head(get_tree(dynforest_obj = res_dyn, tree = 1))
```

```
#>           type id_node var_split feature     threshold   N Nevent depth
#> 1 Longitudinal       1         3       1 -1.272629e-01 129     51     1
#> 2      Numeric       2         1      NA  4.138210e+01  39     27     2
#> 3 Longitudinal       3         4       1  1.459346e+02  90     24     2
#> 4 Longitudinal       4         3       1  3.608271e-11   8      3     3
#> 5 Longitudinal       5         2       1  5.924123e+01  31     24     3
#> 6 Longitudinal       6         1       1  2.786575e-01  63     12     3
```

```
tail(get_tree(dynforest_obj = res_dyn, tree = 1))
```

```
#>            type id_node var_split feature     threshold N Nevent depth
#> 50         Leaf     174        NA      NA            NA 2      2     8
#> 51 Longitudinal     175         2       1 -1.850322e-10 4      4     8
#> 52         Leaf     250        NA      NA            NA 5      1     8
#> 53         Leaf     251        NA      NA            NA 4      2     8
#> 54         Leaf     350        NA      NA            NA 2      2     9
#> 55         Leaf     351        NA      NA            NA 2      2     9
```

Looking at the head of the `get_tree()` function output, we see that subjects were split at node 1 (`id_node`) using the first random-effect (`feature` = 1) of the third `Longitudinal` predictor (`var_split` = 3) with `threshold` = -0.1273. `var_split` = 3 corresponds to `albumin`, so subjects at node 1 with `albumin` values below -0.1273 are assigned to node 2, otherwise to node 3. The last rows of the random forest given by the tail of `get_tree()` function output provide the leaves descriptions. For instance, in row 53, 4 subjects are included in leaf 251, and 2 subjects have the event of interest.

Estimated cumulative incidence function (CIF) within each leaf of a tree can be displayed using the `plot()` function. For instance, the CIF of the cause of interest for leaf 251 in tree 1 can be displayed using the following code:

CIF of a single tree is not meant to be interpreted alone. The CIF should be averaged over all trees of the random forest. For a subject, the estimated CIF over the random forest is obtained by averaging all the tree-specific CIFs of the tree-leaf where the subject belongs. This can be done with the `plot()` function such as:

```
plot(res_dyn, id = 104, max_tree = 9)
```

In this example, we display in figure 2 for subject 104 the tree-specific CIFs for the first 9 trees where this subject is used to grow the trees. This figure shows how the estimated CIF can differ across the trees and requires to be averaged as each is calculated from information of the few subjects belonging to a leaf.

4.5 Out-Of-Bag error

The Out-Of-Bag error (OOB) aims at assessing the predictive abilities of the random forest. With a survival outcome, the OOB error is evaluated using the Integrated Brier Score (IBS) (Gerds and Schumacher, 2006). It is computed using the `compute_ooberror()` function with an object of class `dynforest` as the main argument, such as:

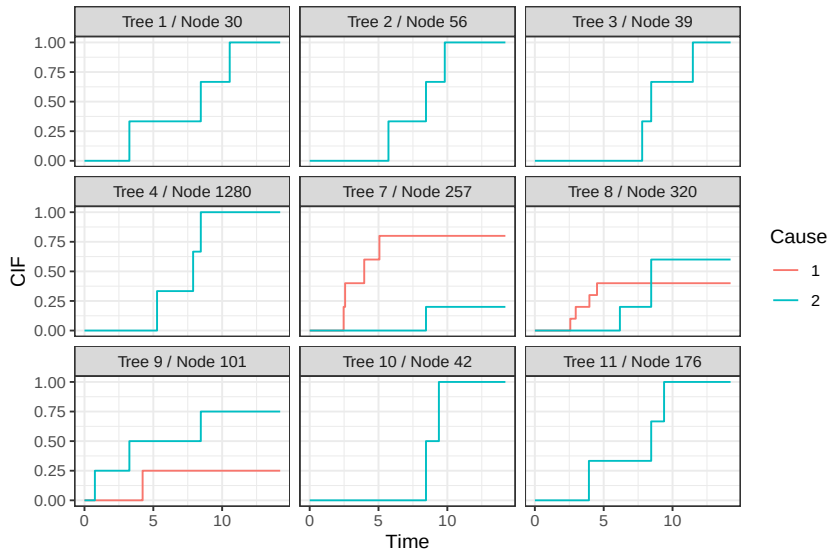


Figure 2: Estimated cumulative incidence functions for subject 104 over 9 trees.

```
res_dyn_OOB <- compute_ooberror(dynforest_obj = res_dyn)
```

`compute_ooberror()` returns the OOB errors by individual. The overall OOB error for the random forest is obtained by averaging the individual specific OOB errors, and can be displayed using `print()` or directly by calling the object.

```
res_dyn_OOB
```

```
#> [1] 0.1265053
```

We obtain an IBS of 0.127 computed from time 0 to the maximum event time. The time range can be modified using `IBS.min` and `IBS.max` arguments to define the minimum and maximum, respectively. To maximize the predictive ability of the random forest, the hyperparameters can be tuned, that is, chosen as those that minimize the OOB error (see section 4.8).

4.6 Individual prediction of the outcome

The `predict()` function allows prediction of the outcome for a new subject using the trained random forest. The function requires the individual data: time-dependent predictors in `timeData` and time-fixed predictors in `fixedData`. For a survival outcome, dynamic predictions can be computed by fixing a prediction time (called landmark time, argument `t0`) from which the prediction is made. In this case, only the history of the individual up to this landmark time (including the longitudinal and time-fixed predictors) will be used. In particular, if the landmark time is fixed to 0, only the information at time 0 will be considered for predicting the outcome.

For the illustration, we only select the subjects still at risk at the landmark time of 4 years. We build the dataframe for those subjects and we predict the individual-specific CIF using the `predict()` function as follows:

```
id_pred <- unique(pbc2_pred$id[which(pbc2_pred$years > 4)])
pbc2_pred_tLM <- pbc2_pred[which(pbc2_pred$id %in% id_pred), ]
timeData_pred <- pbc2_pred_tLM[,
  c("id", "time", "serBilir", "SGOT", "albumin", "alkaline")
]
fixedData_pred <- unique(pbc2_pred_tLM[, c("id", "age", "drug", "sex")])
```

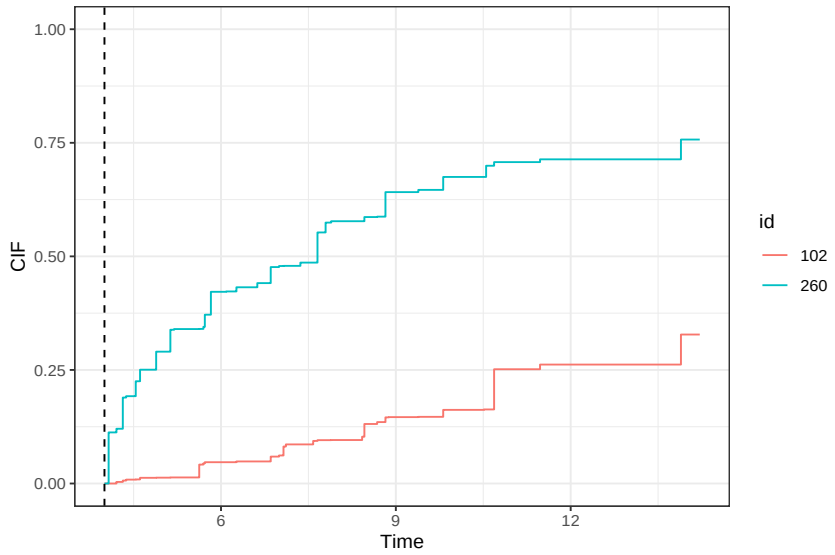


Figure 3: Predicted cumulative incidence function for subjects 102 and 260 from landmark time of 4 years (represented by the dashed vertical line)

```
pred_dyn <- predict(
  object = res_dyn,
  timeData = timeData_pred,
  fixedData = fixedData_pred,
  idVar = "id",
  timeVar = "time",
  t0 = 4
)
```

The `predict()` function provides several elements as described in section 3.2. In addition, the `plot()` function can be used to display the CIF of the outcome (here death before transplantation) for subjects indicated with argument `id`. For instance, we compute the CIF for subjects 102 and 260 with the following code and display them in figure 3.

In the first year after the landmark time (at 4 years), we observe a rapid increase in the risk of death for subject 260 compared to subject 102. We also notice that after 10 years from the landmark time, subject 260 has a probability of death almost three times higher than that of subject 102.

4.7 Predictiveness of the variables

Variable importance

The main objective of the random forest is to predict an outcome. But usually, we are interested in identifying which predictors are the most predictive. The VIMP statistic (Ishwaran et al., 2008) can be computed using the `compute_vimp()` function. This function returns the VIMP statistic for each predictor with the `$Importance` element. These results can also be displayed using the `plot()` function, either in absolute value by default or in percentage with the `PCT` argument set to `TRUE`.

```
res_dyn_VIMP <- compute_vimp(dynforest_obj = res_dyn, seed = 123)

p1 <- plot(res_dyn_VIMP, PCT = TRUE)
```

The VIMP results are displayed in figure 4A. The most predictive variables are `serBilir`, `albumin` and age with the largest VIMP percentage. By removing the association between `serBilir` and the event, the OOB error was increased by 30%.

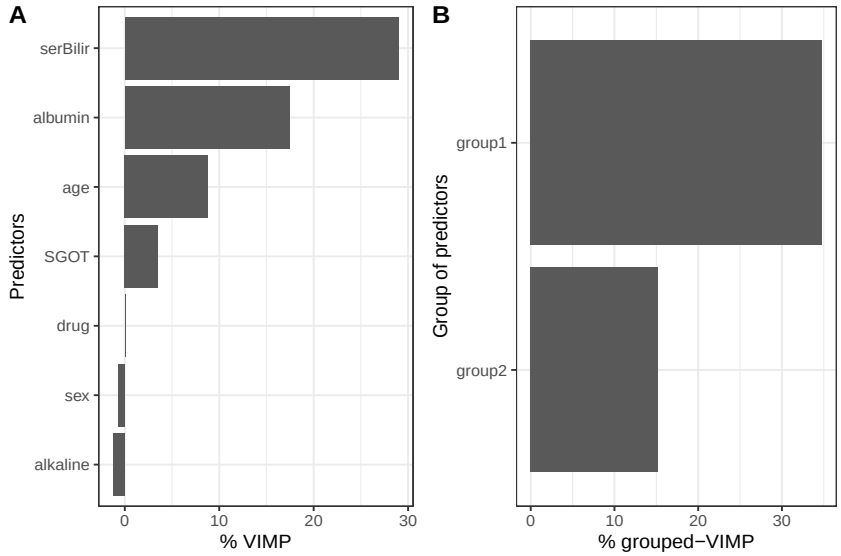


Figure 4: (A) VIMP statistic and (B) grouped-VIMP statistic displayed as a percentage of loss in OOB error of prediction. group1 includes serBilir and SGOT; group2 includes albumin and alkaline.

In the case of correlated predictors, the predictors can be regrouped into dimensions and the VIMP can be computed at the dimension group level with the gVIMP statistic. Permutation is done for each variable of the group simultaneously. The gVIMP is computed with the `compute_gvimp()` function in which the `group` argument defines the group of predictors as a list. For instance, with two groups of predictors (named group1 and group2), the gVIMP statistic is computed using the following code:

```
group <- list(
  group1 = c("serBilir", "SGOT"),
  group2 = c("albumin", "alkaline")
)
res_dyn_gVIMP <- compute_gvimp(dynforest_obj = res_dyn, group = group, seed = 123)

p2 <- plot(res_dyn_gVIMP, PCT = TRUE)
```

Similar to the VIMP statistic, the gVIMP results can be displayed using the `plot()` function. The figure 4B shows that group1 has the highest gVIMP percentage with 34%.

```
plot_grid(p1, p2, labels = c("A", "B"))
```

To compute the gVIMP statistic, the groups can be defined regardless of the number of predictors. However, the comparison between the groups may be harder when group sizes are very different.

Minimal depth

To go further into the understanding of the tree building process, the `compute_vardepth()` function extracts information about the average minimal depth by feature (`$min_depth`), the minimal depth for each feature and each tree (`$var_node_depth`), and the number of times that the feature is used for splitting for each feature and each tree (`$var_count`).

Using an object from the `compute_vardepth()` function, the `plot()` function allows us to plot the distribution of the average minimal depth across the trees. The `plot_level` argument defines how the average minimal depth is plotted, by predictor or feature.

The distribution of the minimal depth level is displayed in figure 5 by predictor and feature. Note that the minimal depth level should always be interpreted with the number of

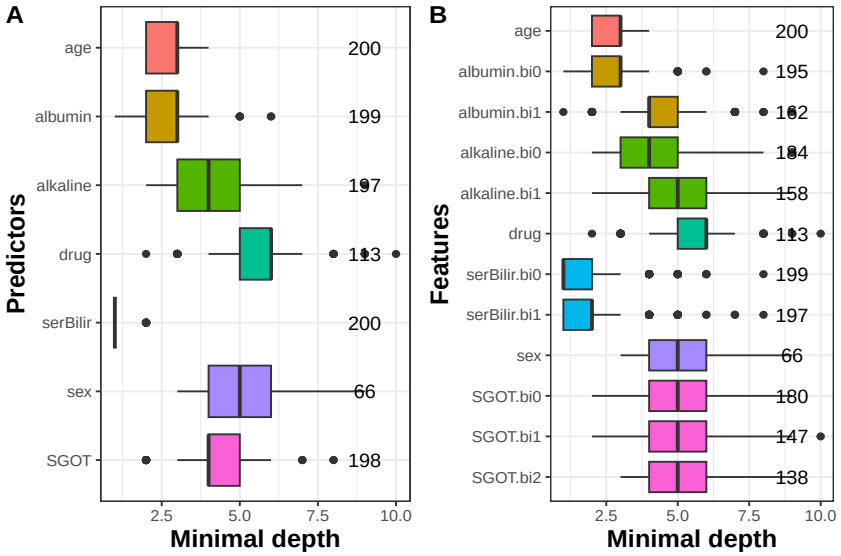


Figure 5: Average minimal depth level by predictor (A) and feature (B).

trees where the predictor/feature is found. Indeed, to accurately appreciate the importance of a variable's minimal depth, the variable has to be systematically part of the candidates at each node. This is why we strongly advise computing the minimal depth on a random forest with the `mtry` hyperparameter chosen at its maximum (as done below).

```
res_dyn_max <- dynforest(
  timeData = timeData_train,
  fixedData = fixedData_train,
  timeVar = "time", idVar = "id",
  timeVarModel = timeVarModel, Y = Y,
  ntree = 200, mtry = 7, nodesize = 2, minsplit = 3,
  cause = 2, ncores = 7, seed = 1234
)

depth_dyn <- compute_vardepth(dynforest_obj = res_dyn_max)
p1 <- plot(depth_dyn, plot_level = "predictor")

p2 <- plot(depth_dyn, plot_level = "feature")

plot_grid(p1, p2, labels = c("A", "B"))
```

In our example, we ran a random forest with the mtry hyperparameter set to its maximum (i.e., `mtry` = 7) and we computed the minimal depth on this random forest. We observe that `serBilir`, `albumin` and age have the lowest minimal depth, indicating these predictors are used to split the subjects at early stages in 200 out of 200 trees, i.e., 100% for `serBilir`, age and in 199 out of 200 for `albumin` (figure 5A). The minimal depth level by feature (figure 5B) provides more advanced details about the tree building process. For instance, we can see that the random effects of `serBilir` (indicated by bi0 and bi1 in the graph) are the earliest features used on 199 and 197 out of 200 trees, respectively.

4.8 Guidelines to tune the hyperparameters

The predictive performance of the random forest strongly depends on the hyperparameters `mtry`, `nodesize` and `minsplit`. They should therefore be chosen carefully. The `nodesize` and `minsplit` hyperparameters control the tree depth. The trees need to be deep enough to ensure that the predictions are accurate. By default, we fixed `nodesize` and `minsplit` at

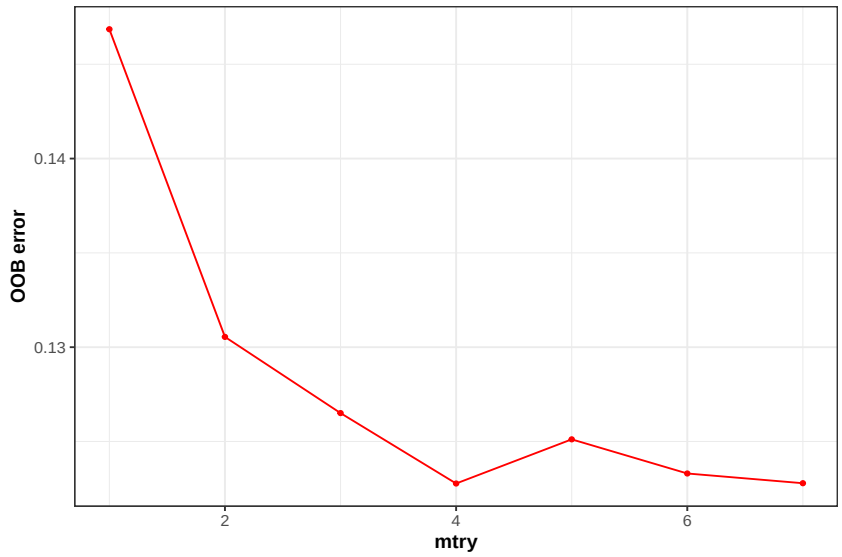


Figure 6: OOB error according to mtry hyperparameter. The optimal value was found for the maximum value mtry = 7.

the minimum, that is `nodesize` = 1 and `minsplit` = 2. However, with a large number of individuals, the tree depth could be slightly decreased by increasing these hyperparameters in order to reduce the computation time.

The `mtry` hyperparameter defines the number of predictors randomly drawn at each node. By default, we chose `mtry` equal to the square root of the number of predictors as usually recommended (Bernard et al., 2009). However, this hyperparameter should be carefully tuned with possible values between 1 and the number of predictors. Indeed, the predictive performance of the random forest is highly related to this hyperparameter.

In the illustration, we tuned `mtry` for every possible value (1 to 7). Figure 6 displays the OOB error according to the `mtry` hyperparameter.

We can see in this figure large OOB error differences according to the `mtry` hyperparameter. In particular, we observe the worst predictive performance for `mtry` = 1. The minimum OOB error value is obtained for `mtry` = 7 although differences from `mtry` = 3 to `mtry` = 7 seem relatively minimal. Moreover, a larger `mtry` value has a significant impact on the computation time. Using 7 cores in this example, the `dynforest()` function has been executed in 1.7, 2.8, 3.8, 4.7, 5.4, 6.2 and 7.2 minutes, for `mtry` values ranging from 1 to 7, respectively. The number of subjects, time measurements, trees, and their depths may also contribute to longer computation time.

5 How to use DynForest R package with a categorical outcome?

In this section, we use **DynForest** in a classification perspective using `pbc2` data. For illustration purposes, we want to predict death between 4 and 10 years on subjects still at risk at 4 years from the repeated data up to 4 years. Note that this is only for illustrative purposes as this technique does not handle the censoring of death times correctly.

5.1 Data management

For the illustration, we select patients still at risk at 4 years and we recode the `event` variable with `event` = 1 for subjects who died between 4 years and 10 years, whereas subjects with transplantation were recoded `event` = 0, as were the subjects still alive. We split the subjects into two datasets: (i) one dataset to train the random forest using 2/3 of patients; (ii) one dataset to predict on the other 1/3 of patients.

```
pbc2 <- pbc2[which(pbc2$years > 4 & pbc2$time <= 4), ]
pbc2$event <- ifelse(pbc2$event == 2, 1, 0)
pbc2$event[which(pbc2$years > 10)] <- 0
set.seed(1234)
id <- unique(pbc2$id)
id_sample <- sample(id, length(id) * 2 / 3)
id_row <- which(pbc2$id %in% id_sample)
pbc2_train <- pbc2[id_row, ]
pbc2_pred <- pbc2[-id_row, ]
```

We use the same strategy as in the survival context (section 4) to build the random forest, with the same predictors and the same association for time-dependent predictors.

```
timeData_train <- pbc2_train[,
  c("id", "time", "serBilir", "SGOT", "albumin", "alkaline")
]
timeVarModel <- list(
  serBilir = list(fixed = serBilir ~ time, random = ~time),
  SGOT = list(fixed = SGOT ~ time + I(time^2), random = ~ time + I(time^2)),
  albumin = list(fixed = albumin ~ time, random = ~time),
  alkaline = list(fixed = alkaline ~ time, random = ~time)
)
fixedData_train <- unique(pbc2_train[, c("id", "age", "drug", "sex")])
```

With a categorical outcome, the definition of the output object is slightly different. We should specify `type="factor"` to define the outcome as categorical, and the dataframe in `Y` should contain only 2 columns, the variable identifier `id` and the outcome `event`.

```
Y <- list(
  type = "factor",
  Y = unique(pbc2_train[, c("id", "event")])
)
```

5.2 The random forest building

We executed the `dynforest()` function to build the random forest with hyperparameters `mtry` = 7 and `nodesize` = 2 as follows:

```
res_dyn <- dynforest(
  timeData = timeData_train,
  fixedData = fixedData_train,
  timeVar = "time", idVar = "id",
  timeVarModel = timeVarModel,
  mtry = 7, nodesize = 2,
  Y = Y, ncores = 7, seed = 1234
)
```

5.3 Out-Of-Bag error

With a categorical outcome, the OOB prediction error is evaluated using a misclassification criterion. This criterion can be computed with the `compute_ooberror()` function and the results of the random forest can be displayed using `summary()`:

```
res_dyn_OOB <- compute_ooberror(dynforest_obj = res_dyn)

summary(res_dyn_OOB)
```

```
#> dynforest executed for categorical outcome
#>  Splitting rule: Minimize weighted within-group Shannon entropy
#>  Out-of-bag error type: Missclassification
#>  Leaf statistic: Majority vote
#> ----------------
#> Input
#>  Number of subjects: 150
#>  Longitudinal: 4 predictor(s)
#>  Numeric: 1 predictor(s)
#>  Factor: 2 predictor(s)
#> ----------------
#> Tuning parameters
#>  mtry: 7
#>  nodesize: 2
#>  ntree: 200
#> ----------------
#> ----------------
#> dynforest summary
#>  Average depth per tree: 5.89
#>  Average number of leaves per tree: 16.81
#>  Average number of subjects per leaf: 5.72
#> ----------------
#> Out-of-bag error based on Missclassification
#>  Out-of-bag error: 0.2333
#> ----------------
#> Computation time
#>  Number of cores used: 7
#>  Time difference of 2.87888 mins
#> ----------------
```

In this illustration, we built the random forest using 150 subjects because we only kept the subjects still at risk at the landmark time at 4 years and split the dataset in 2/3 for training and 1/3 for testing. We have on average 5.7 subjects per leaf, and the average depth level per tree is 5.9. This random forest predicted the wrong outcome for 23% of the subjects. The random forest performance can be optimized by choosing the `mtry` and `nodesize` hyperparameters that minimize the OOB misclassification.

5.4 Prediction of the outcome

We can predict the probability of death between 4 and 10 years for subjects still at risk at the landmark time at 4 years. In classification mode, the predictions are performed using the majority vote. The prediction over the trees is thus a category of the outcome along with the proportion of the trees that lead to this category. Predictions are computed using the `predict()` function, then a dataframe can be easily built from the returning object to get the prediction and probability of the outcome for each subject:

```
timeData_pred <- pbc2_pred[,
  c("id", "time", "serBilir", "SGOT", "albumin", "alkaline")
]
fixedData_pred <- unique(pbc2_pred[, c("id", "age", "drug", "sex")])
pred_dyn <- predict(
  object = res_dyn,
  timeData = timeData_pred,
  fixedData = fixedData_pred,
  idVar = "id", timeVar = "time",
  t0 = 4
```

```
)

head(data.frame(
  pred = pred_dyn$pred_indiv,
  proba = pred_dyn$pred_indiv_proba
))

#>     pred proba
#> 101    0 0.945
#> 104    0 0.790
#> 106    1 0.600
#> 108    0 0.945
#> 112    1 0.575
#> 114    0 0.650
```

As shown in this example, some predictions are made with varying confidence from 57.5% for subject 112 to 94.5% for subject 101. We predict, for instance, no event for subject 101 with a probability of 94.5% and an event for subject 106 with a probability of 60.0%.

5.5 Predictiveness variables

Variable importance

The most predictive variables can be identified using the `compute_vimp()` function and displayed using the `plot()` function as follows:

```
res_dyn_VIMP <- compute_vimp(dynforest_obj = res_dyn_OOB, seed = 123)
plot(res_dyn_VIMP, PCT = TRUE)
```

Again, we found that the most predictive variable is `serBilir`. When perturbing `serBilir`, the OOB prediction error was increased by 15%.

Minimal depth

The minimal depth is computed using the `compute_vardepth()` function and is displayed at predictor and feature levels using the `plot()` function. The results are displayed in figure 7 using the random forest with maximal `mtry` hyperparameter value (i.e., `mtry` = 7) for a better understanding.

```
depth_dyn <- compute_vardepth(dynforest_obj = res_dyn_OOB)
p1 <- plot(depth_dyn, plot_level = "predictor")

p2 <- plot(depth_dyn, plot_level = "feature")

plot_grid(p1, p2, labels = c("A", "B"))
```

We observe that `serBilir` and `albumin` have the lowest minimal depth and are used to split the subjects in almost all the trees (199 and 196 out of 200 trees, respectively) (figure 7A). Figure 7B provides further results. In particular, this graph shows that the random intercept (indicated by bi0) of `serBilir` and `albumin` are the earliest predictors used to split the subjects and are present in 192 and 191 out of 200 trees, respectively.

6 Discussion

The **DynForest** R package provides an easy-to-use random forests methodology for predictors that may contain longitudinal variables possibly measured irregularly with error. Note

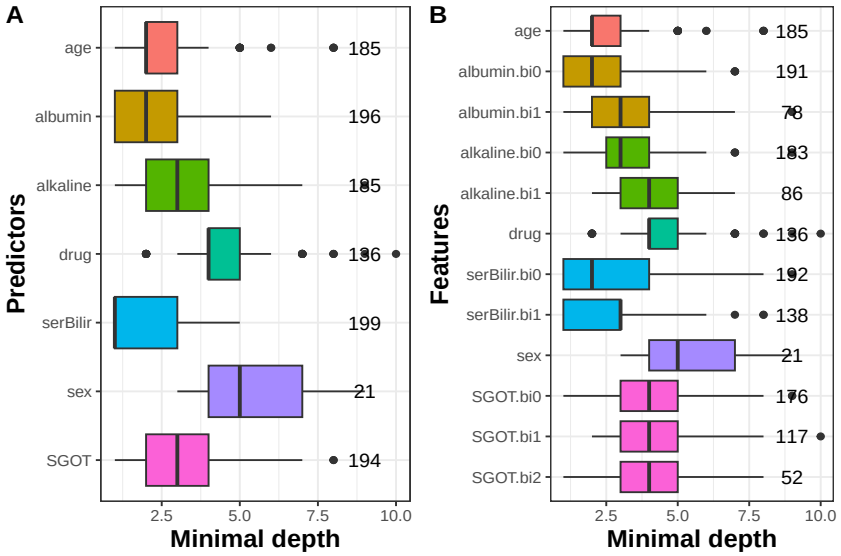


Figure 7: Average minimal depth by predictor (A) and feature (B).

that the method can also be used without any longitudinal predictors, as in other random forests packages.

We implemented several statistics to identify the predictive ability of each variable with the VIMP, gVIMP and average minimal depth. For the survival outcome, compared to the **randomForestSRC** R package, we considered two different stopping criteria `nodesize` and `minsplit` to favor the deepest forests possible and avoid suboptimal splits. We designed **DynForest** to be as user-friendly as possible. To achieve that, we implemented various functions to summarize and display the results, and provided a step-by-step analysis in survival and categorical modes. As **DynForest** involves mixed models to be updated within each tree building, the program can be numerically demanding despite efforts to speed up computations (parallel computing, history of previous estimations). A substantial computational burden is especially expected when the sample is very large and/or a very large set of time-dependent predictors is considered.

Several improvements could be considered in the future. We used linear mixed models for longitudinal continuous outcomes, but alternative strategies could be considered such as the PACE algorithm (Yao et al., 2005) based on functional data analysis. We could also consider different natures of longitudinal predictors (e.g., binary) for which generalized linear mixed models could be used. **DynForest** currently handles continuous, categorical and survival (with possibly competing events) outcomes. But other outcomes could be envisaged such as curves, recurrent events or interval-censored time-to-events. We leave these perspectives for future releases.

Computational details

The results in this paper were obtained using R 4.4.1 with the **DynForest** 1.2.0 package on a virtualized Windows Server 2016 Remote Desktop Server with 48GB RAM. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`.

Acknowledgments

We thank Dr. Louis Capitaine for FrechForest R code used in **DynForest**.

This work was funded by the French National Research Agency (ANR-18-CE36-0004-01 for project DyMES), and the French government in the framework of the PIA3 ("Investment

for the future") (project reference 17-EURE-0019) and in the framework of the University of Bordeaux's IdEx "Investments for the Future" program / RRI PHDS.

References

O. Aalen. Nonparametric inference in connection with multiple decrement models. *Scandinavian Journal of Statistics*, 3(1):15–27, 1976. ISSN 0303-6898. URL https://www.jstor.org/stable/4615603. [p26]

O. O. Aalen and S. Johansen. An empirical transition matrix for non-homogeneous markov chains based on censored observations. *Scandinavian Journal of Statistics*, 5(3):141–150, 1978. ISSN 0303-6898. URL https://www.jstor.org/stable/4615704. [p26]

S. Bernard, L. Heutte, and S. Adam. Influence of hyperparameters on random forest accuracy. In *International Workshop on Multiple Classifier Systems*, pages 171–180. Springer, 2009. doi: 10.1007/978-3-642-02326-2_18. [p40]

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. [p24]

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016. doi: 10.1145/2939672.2939785. [p24]

A. Devaux. *DynForest: Random Forest with Multivariate Longitudinal Predictors*, 2024. URL https://CRAN.R-project.org/package=DynForest. R package version: 1.1.3. [p28]

A. Devaux, C. Helmer, R. Genuer, and C. Proust-Lima. Random survival forests with multivariate longitudinal endogenous covariates. *Statistical Methods in Medical Research*, 32(12):2331–2346, 2023. doi: 10.1177/09622802231206477. [p24]

T. A. Gerds and M. Schumacher. Consistent estimation of the expected brier score in general survival models with right-censored event times. *Biometrical Journal*, 48(6):1029–1040, 2006. ISSN 1521-4036. doi: 10.1002/bimj.200610301. [p28, 35]

B. Gray. *cmprsk: Subdistribution Analysis of Competing Risks*, 2020. URL https://CRAN.R-project.org/package=cmprsk. R package version 2.2-10. [p31]

R. J. Gray. A class of k-sample tests for comparing the cumulative incidence of a competing risk. *The Annals of Statistics*, 16(3):1141–1154, 1988. ISSN 0090-5364. URL https://www.jstor.org/stable/2241622. [p26]

B. Gregorutti, B. Michel, and P. Saint-Pierre. Correlation and variable importance in random forests. *Statistics and Computing*, 27(3):659–678, 2017. doi: 10.1007/s11222-016-9646-1. [p28]

H. Ishwaran and U. B. Kogalur. *Fast Unified Random Forests for Survival, Regression, and Classification (RF-SRC)*, 2022. URL https://cran.r-project.org/package=randomForestSRC. R package version 3.1.1. [p24]

H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3):841–860, 2008. ISSN 1932-6157. doi: 10.1214/08-AOAS169. [p24, 28, 37]

H. Ishwaran, T. A. Gerds, U. B. Kogalur, R. D. Moore, S. J. Gange, and B. M. Lau. Random survival forests for competing risks. *Biostatistics*, 15(4):757–773, 2014. ISSN 1468-4357, 1465-4644. doi: 10.1093/biostatistics/kxu010. [p24]

N. M. Laird and J. H. Ware. Random-effects models for longitudinal data. *Biometrics*, 38(4): 963–974, 1982. ISSN 0006-341X. doi: 10.2307/2529876. [p25]

P. A. Murtaugh, E. R. Dickson, G. M. Van Dam, M. Malinchoc, P. M. Grambsch, A. L. Langworthy, and C. H. Gips. Primary biliary cirrhosis: Prediction of short-term survival based on repeated patient visits. *Hepatology*, 20(1):126–134, 1994. ISSN 0270-9139, 1527-3350. doi: 10.1002/hep.1840200120. [p32]

W. Nelson. Hazard plotting for incomplete failure data. *Journal of Quality Technology*, 1(1): 27–52, 1969. ISSN 0022-4065. doi: 10.1080/00224065.1969.11980344. [p26]

R. Peto and J. Peto. Asymptotically efficient rank invariant test procedures. *Journal of the Royal Statistical Society: Series A (General)*, 135(2):185–198, 1972. doi: 10.2307/2344317. [p26]

C. Proust-Lima, V. Philipps, and B. Liquet. Estimation of extended mixed models using latent classes and latent processes: The R package lcmm. *Journal of Statistical Software*, 78 (2):1–56, 2017. doi: 10.18637/jss.v078.i02. [p30, 31]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL https://www.R-project.org/. [p24]

M. Sène, J. M. G. Taylor, J. J. Dignam, H. Jacqmin-Gadda, and C. Proust-Lima. Individualized dynamic prediction of prostate cancer recurrence with and without the initiation of a second treatment: Development and validation. *Statistical Methods in Medical Research*, 25 (6):2972–2991, 2016. doi: 10.1177/0962280214535763. [p27]

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. [p26]

T. M. Therneau. *A Package for Survival Analysis in R*, 2022. URL https://CRAN.R-project.org/package=survival. R package version 3.4-0. [p31]

M. N. Wright and A. Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1), 2017. ISSN 1548-7660. doi: 10.18637/jss.v077.i01. [p24]

F. Yao, H.-G. Müller, and J.-L. Wang. Functional data analysis for sparse longitudinal data. *Journal of the American Statistical Association*, 100(470):577–590, 2005. ISSN 0162-1459, 1537-274X. doi: 10.1198/016214504000001745. [p44]

Anthony Devaux
Inserm U1219 - Bordeaux Population Health - Université de Bordeaux - The George Institute for Global Health - UNSW Sydney - School of Population Health - UNSW Sydney
Bordeaux (France) - Sydney (Australia)
ORCiD: 0000-0002-8862-4218
anthony.devauxbarault@gmail.com

Cécile Proust-Lima
Inserm U1219 - Bordeaux Population Health - Université de Bordeaux
Bordeaux (France)
ORCiD: 0000-0002-9884-955X

Robin Genuer
Inserm U1219 - Bordeaux Population Health - Université de Bordeaux
Bordeaux (France)
ORCiD: 0000-0002-0981-3943

LCCR: An R Package for Inference on Latent Class Models for Capture-Recapture Data with Covariates

by Francesco Bartolucci and Antonio Forcina

Abstract A detailed description of the R package LCCR for the analysis of capture-recapture data relating to a closed population is provided. The data that can be analyzed consist of the full capture history for each sample unit and may possibly include individual covariates. The package allows the specification and estimation of latent class models in which the distribution of the capture history conditional on the latent class and covariates follows either a log-linear model in which bivariate interactions are allowed, or a logit model for the conditional probability of capture at each occasion given the previous capture history. Alternatively, the conditional distribution of each capture occasion can be formulated by a logit model that may account for the effect of previous capture occasions. Apart from the conditional distribution of the capture history, the covariates can also affect the distribution of the class weights. Estimation is based on the unconditional maximum likelihood method, suitably extended to account for the presence of covariates by including unit-specific weights, which are commonly used in empirical likelihood methods. The package also allows simulation of capture-recapture data from a specified model and the computation of the profile confidence interval for the population size. For illustration, we use a data set about meningitis cases in an Italian region.

1 Introduction

In the statistical literature on capture-recapture data relating to closed populations, see Böhning et al. (2018) for a recent overview, several models are now available, which represent a development of traditional models such as those illustrated in Otis et al. (1978). Here we focus on the typical case in which, for each unit captured at least once, the full capture history is available. In this case, the most recent model formulations can address both observed heterogeneity, accounted for by individual covariates, and unobserved heterogeneity, using random effects or latent variables having a continuous or discrete distribution. Moreover, maximum likelihood estimation may be formulated in two different ways; see Sanathanan (1972) for a fundamental contribution in this regard. The simplest method is based on maximizing the conditional likelihood function of the observed data given that sample units have been captured at least once. This function depends on the model parameters only, which are estimated prior to the population size. In contrast, the unconditional maximum likelihood (UML) method is based on a target function which is jointly maximized with respect to the population size and model parameters.

In this paper, we focus on a class of models that may address both observed and unobserved heterogeneity, by the inclusion of individual covariates and latent variables having a discrete distribution, and which are described in Bartolucci and Forcina (2024); this class of models is closely related to the one proposed in Bartolucci and Forcina (2006) and Bartolucci and Pennoni (2007). In particular, we describe the R package **LCCR** (Bartolucci and Forcina, 2025) that contains functions to formulate and estimate these models. Estimation is based on the UML method described in Bartolucci and Forcina (2024) that takes inspiration from the formulation based on empirical likelihood (Owen, 2001), developed by Liu et al. (2017). The functions in package **LCCR** may also be used to simulate data from an assumed model and to obtain a profile confidence interval for the population size, which has interesting properties with respect to standard confidence intervals based on the standard error for this parameter.

In R, other packages are available to deal with capture-recapture data. Here we mention two among the most popular packages which are available in CRAN and are related to the

present package **LCCR**. Package **Rcapture** (Baillargeon and Rivest, 2007) provides functions to fit log-linear models for capture-recapture data for closed and open populations, together with tools for preliminary analysis of these data based on descriptive statistics and for model selection and diagnostics. For closed populations, in particular, the models considered may be specified in different ways to also include behavioral and heterogeneity effects (see also Rivest and Baillargeon, 2007), while parameter estimation is based on a conditional maximum likelihood method. Inference on the population size is also based on the profile likelihood method of Cormack (1992). Another R package that may be used to fit capture-recapture models for closed populations is **BBRecapture**. The models considered in the package allow for accounting for behavioral effects in a flexible way; see also Alunni Fegatelli and Tardella (2016). The inference on these models is based on a Bayesian approach under a set of prior distributions on the parameters, although the package also contains functions for UML estimation. A related package is **LCMCR** (Manrique-Vallier, 2023) that, by a Markov chain Monte Carlo algorithm, fits a Bayesian non-parametric latent class model for capture-recapture data that does not require specifying the number of classes because it relies on a Dirichlet process, as described in Manrique-Vallier (2016). It is worth noting that, differently from the previous packages, the **LCCR** package described here allows for individual covariates conditionally on the latent class, apart from permitting the formulation of models with a variety of effects of the type conceptualized by Otis et al. (1978).

The remainder of the paper is organized as follows. In the next section, we provide an overview of the statistical approach in terms of model assumptions and inferential methods, summarizing some concepts introduced in Bartolucci and Forcina (2024). Then, we describe a set of preliminary functions of package **LCCR** to formulate models, simulate data from the assumed model, and manipulate the available data. A further section is devoted to the main R functions to estimate model parameters and the population size, also by a profile confidence interval. The paper concludes with an example based on Italian meningitis data and a brief summary.

2 Approach overview

Let J denote the number of capture occasions generating configurations denoted by $\boldsymbol{r} = (r_1, \ldots, r_J)'$, with $r_j = 0, 1$, $j \in \mathcal{J}$, where $\mathcal{J} = \{1, \ldots, J\}$. The set of all possible capture configurations $\boldsymbol{r}$ apart from $\mathbf{0}$ is denoted by $\mathcal{R}$, where in general $\mathbf{0}$ denotes a column vector of zeros of a suitable dimension; when this set includes the $\mathbf{0}$ configuration, it is denoted by $\tilde{\mathcal{R}}$. In the present framework, the units captured at least once, whose number is denoted by n, may be collected in I strata with each stratum $i \in \mathcal{I}$, $\mathcal{I} = \{1, \ldots, I\}$, having a corresponding vector of c covariates $\boldsymbol{w}_i$. We also consider the possible presence of vectors of covariates $\boldsymbol{x}_{i,j}$, of dimension d, which are also specific to the capture occasion. The observed capture-recapture configurations are represented by the column vectors $\boldsymbol{y}_i$ having $2^J - 1$ elements $y_{i,\boldsymbol{r}}$, $\boldsymbol{r} \in \mathcal{R}$, corresponding to the frequency of the capture configuration $\boldsymbol{r}$ in stratum i. The sum of these frequencies is denoted by n_i, with all the n_i collected in the vector $\boldsymbol{n}$. Note that, with individual data, each vector $\boldsymbol{y}_i$ has only one element equal to 1 and all other elements equal to 0, so that $n_i = 1$ for all i.

2.1 Model assumptions

The approach proposed in Bartolucci and Forcina (2024) is based on a latent class (LC) model with H classes and in which the class weights $\pi_{h,i}$, $h \in \mathcal{H}$, $i \in \mathcal{I}$, with $\mathcal{H} = \{1, \ldots, H\}$, may depend on the vectors of covariates $\boldsymbol{w}_i$. The class weights for the same stratum i are collected in the column vector $\boldsymbol{\pi}_i$. Moreover, the conditional probability of the capture configuration $\boldsymbol{r}$ given the latent class h and that the individual is in stratum i, denoted by $q_{h,i,\boldsymbol{r}}$, may depend on the vectors of covariates $\boldsymbol{x}_{i,j}$. These probabilities are collected in the column vector $\boldsymbol{q}_{h,i}$ for all $\boldsymbol{r} \in \mathcal{R}$ in lexicographical order. When also the capture-recapture configuration $\mathbf{0}$ is included, the probability vector at issue will be denoted by $\tilde{\boldsymbol{q}}_{h,i}$ that then has elements $q_{h,i,\boldsymbol{r}}$ for all $\boldsymbol{r} \in \tilde{\mathcal{R}}$. Two different approaches are adopted to model this vector of probabilities:

the first is based on a log-linear parametrization and the second is based on a recursive logit parametrization.

Finally, we recall that the probability of capture configuration $\boldsymbol{r}$ for stratum h is equal to

$$p_{i,\boldsymbol{r}} = \sum_{h \in \mathcal{H}} \pi_{h,i} q_{h,i,\boldsymbol{r}};$$

a related expression may be used to compute the overall vector of such probabilities, $\boldsymbol{p}_i$ (or $\tilde{\boldsymbol{p}}_i$ when also the elements corresponding to $\boldsymbol{r} = \boldsymbol{0}$ are included), on the basis of vectors $\boldsymbol{q}_{h,i}$ (or $\tilde{\boldsymbol{q}}_{h,i}$) and $\boldsymbol{\pi}_i$.

Before describing in detail the single model components, it is worth recalling that the LC approach to dealing with unobserved heterogeneity is rather common in the capture-recapture literature. Among the most recent papers dealing with this approach, that by Aleshin-Guendel et al. (2024) focuses on identifiability issues, although with reference to simpler models than the ones presented here.

Distribution of the latent classes

For the conditional distribution of the latent classes given the covariates, we assume a multinomial logit model based on the assumption that

$$\pi_{h,i} = \frac{1}{1 + \sum_{l \in \bar{\mathcal{H}}} \exp(\alpha_l + \boldsymbol{w}_i' \boldsymbol{\beta}_l)} \begin{cases} 1, & h = 1, \\ \exp(\alpha_h + \boldsymbol{w}_i' \boldsymbol{\beta}_h), & h \in \bar{\mathcal{H}}, \end{cases}$$

where $\bar{\mathcal{H}} = \{2, \ldots, H\}$. Using matrix notation, the overall vector of class weights may be expressed as

$$\boldsymbol{\pi}_i = \frac{\exp(\boldsymbol{W}_i \boldsymbol{\beta})}{\boldsymbol{1}' \exp(\boldsymbol{W}_i \boldsymbol{\beta})},$$

for a suitable design matrix $\boldsymbol{W}_i$ depending on $\boldsymbol{w}_i$ and where $\boldsymbol{1}$ denotes a column vector of ones of a suitable dimension, while $\boldsymbol{\beta}$ is the vector of all previous parameters.

Conditional response probabilities

The user can choose between two ways of modeling capture probabilities conditionally on the latent class. The first model formulation is based on a log-linear parametrization and assumes that

$$q_{h,i,\boldsymbol{r}} = \frac{\exp\left(\sum_{j \in \mathcal{J}} r_j \gamma_{h,j} + \sum_{j \in \mathcal{J}} r_j \boldsymbol{x}_{i,j}' \boldsymbol{\delta}_{h,j} + \sum_{(j_1,j_2) \in \mathcal{B}} r_{j_1} r_{j_2} \eta_{h,j_1,j_2}\right)}{\sum_{\boldsymbol{u} \in \tilde{\mathcal{R}}} \exp\left(\sum_{j \in \mathcal{J}} u_j \gamma_{h,j} + \sum_{j \in \mathcal{J}} u_j \boldsymbol{x}_{i,j}' \boldsymbol{\delta}_{h,j} + \sum_{(j_1,j_2) \in \mathcal{B}} u_{j_1} u_{j_2} \eta_{h,j_1,j_2}\right)}, \tag{1}$$

where $\gamma_{h,j}$ are main effects, $\boldsymbol{\delta}_{h,j}$ are regression coefficients for the covariates in each vector $\boldsymbol{x}_{i,j}$, and η_{h,j_1,j_2} are bivariate interactions. Moreover, $\mathcal{B}$ is a set of pairs of indices of type (j_1, j_2) of the bivariate interactions, which is possibly empty when these interactions are not used and then the third sum at the numerator and denominator of equation (1) disappears. The number of these interactions is denoted by $b = |\mathcal{B}|$. Obviously, when covariates are not included, even the second sum disappears from the previous equation. All the parameters in (1) are collected in the vector $\tilde{\boldsymbol{\lambda}}$ and suitable constraints, corresponding to the linear form $\tilde{\boldsymbol{\lambda}} = \boldsymbol{L}\boldsymbol{\lambda}$ for a suitable matrix $\boldsymbol{L}$, can be defined in order to make the model more parsimonious. The types of constraint will be discussed in the following when the package is illustrated.

For the overall vector of probabilities we have

$$\tilde{\boldsymbol{q}}_{h,i} = \frac{\exp(\boldsymbol{M}_{h,i} \boldsymbol{\lambda})}{\boldsymbol{1}' \exp(\boldsymbol{M}_{h,i} \boldsymbol{\lambda})}, \tag{2}$$

based on design matrices $\boldsymbol{M}_{h,i}$ depending on $\boldsymbol{L}$ and having a number of rows equal to 2^J and a number of columns equal to that of parameters in (1), which are collected in $\boldsymbol{\lambda}$. In practice, $\boldsymbol{M}_{h,i}\boldsymbol{\lambda}$ contains the terms of the exponential function at the numerator of (1) for all $\boldsymbol{r} \in \tilde{\mathcal{R}}$.

The second model formulation is based on a recursive logit model according to which

$$q_{h,i,\boldsymbol{r}} = \frac{\exp[r_j(\gamma_{h,1} + \boldsymbol{x}'_{i,j}\boldsymbol{\delta}_{h,1})]}{1+\exp(\gamma_{h,1} + \boldsymbol{x}'_{i,j}\boldsymbol{\delta}_{h,1})} \prod_{j=2}^{J} \frac{\exp[r_j(\gamma_{h,j} + \boldsymbol{x}'_{i,j}\boldsymbol{\delta}_{h,j} + f(\boldsymbol{r}_{j-1})\eta_{h,j})]}{1+\exp(\gamma_{h,j} + \boldsymbol{x}'_{i,j}\boldsymbol{\delta}_{h,j} + f(\boldsymbol{r}_{j-1})\eta_{h,j})}, \tag{3}$$

where $f(\boldsymbol{r}_{j-1})$ is a function of the past capture history. In particular, we consider the case in which this function is equal to $I(\boldsymbol{1}'\boldsymbol{r}_{j-1} > 0)$, so that it is equal to 1 if the individual has already been marked and 0 otherwise, or simply $f(\boldsymbol{r}_{j-1}) = \boldsymbol{1}'\boldsymbol{r}_{j-1}$. We also consider the case $f(\boldsymbol{r}_{j-1}) = r_{j-1}$, so that only the previous response variable is relevant. Also in this case, we can assume specific constraints that will be described in the following. The parameters in (3) are still collected in the vector $\tilde{\boldsymbol{\lambda}}$ on which the constraint $\tilde{\boldsymbol{\lambda}} = \boldsymbol{L}\boldsymbol{\lambda}$ is assumed, so that the free parameter vector is $\boldsymbol{\lambda}$.

A convenient way to express this parameterization is as

$$\tilde{\boldsymbol{q}}_{h,i} = \exp\{\boldsymbol{A}\boldsymbol{M}_{h,i}\boldsymbol{\lambda} - \boldsymbol{B}\log[\boldsymbol{1} + \exp(\boldsymbol{M}_{h,i}\boldsymbol{\lambda})]\}, \tag{4}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are suitable matrices having dimension $2^J \times (2^J - 1)$ and with all elements equal to 0 or 1. Moreover, the $\boldsymbol{M}_{h,i}$ are design matrices such that $\boldsymbol{M}_{h,i}\boldsymbol{\lambda}$ is a vector containing the logits corresponding to the single terms in (3).

2.2 Unconditional maximum likelihood inference

Inference on the models described above follows the approach proposed by Liu et al. (2017) and further developed in Bartolucci and Forcina (2024). In summary, we associate a weight τ_i to each stratum i under the constraint that $\sum_{i\in\mathcal{I}} \tau_i = 1$, so that the probability of not being captured is equal to

$$\phi = \sum_{i\in\mathcal{I}} \tau_i \phi_i,$$

where $\phi_i = p_{i,\boldsymbol{0}}$. All such probabilities will be collected in vector $\boldsymbol{\phi}$.

We base inference on the log-likelihood function

$$\ell(\boldsymbol{\psi}) = \log\frac{\Gamma(N+1)}{\Gamma(N-n+1)} + (N-n)\log(\phi) + \sum_{i\in\mathcal{I}}(\boldsymbol{y}'_i \log \boldsymbol{p}_i + \log\tau_i),$$

where $\boldsymbol{\psi}$ denotes the overall vector of parameters including N. In order to maximize this function, we rely on an algorithm that is based on the following steps:

1. for fixed $\boldsymbol{\tau}$ and N, maximize $\ell(\boldsymbol{\psi})$ with respect to $\boldsymbol{\beta}$ and $\boldsymbol{\lambda}$. This amounts to maximizing

$$\ell_1(\boldsymbol{\beta},\boldsymbol{\lambda}) = (N-n)\log(\phi) + \sum_{i=1}^{s} \boldsymbol{y}'_i \log(\boldsymbol{p}_i) \tag{5}$$

   with respect to these parameters, as will be clarified below;

2. for fixed $\boldsymbol{\beta}$, $\boldsymbol{\lambda}$, and N, maximize $\ell(\boldsymbol{\psi})$ with respect to $\boldsymbol{\tau}$. This amounts to maximizing

$$\ell_2(\boldsymbol{\tau}) = (N-n)\log(\phi) + \sum_{i=1}^{s} \log\tau_i$$

   and may be based on a simple updating rule consisting in computing

$$\boldsymbol{\tau} = \frac{1}{N}\left[\boldsymbol{n} + \frac{N-n}{\phi}\text{diag}(\boldsymbol{\phi})\boldsymbol{\tau}\right],$$

where the previous $\boldsymbol{\tau}$ and ϕ are included at the right-hand side; see Bartolucci and Forcina (2024) for details;

3. for fixed $\boldsymbol{\beta}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\tau}$, maximize $\ell(\boldsymbol{\psi})$ with respect to N. This amounts to maximizing

$$\ell_3(N) = \log \frac{\Gamma(N+1)}{\Gamma(N-n+1)} + (N-n)\log(\phi)$$

with respect to N, which may be accomplished by simple Newton-Raphson steps.

Maximization of $\ell_1(\boldsymbol{\beta}, \boldsymbol{\lambda})$ in (5) is based on an Expectation-Maximization (EM) algorithm (Dempster et al., 1977) that has a different implementation according to whether a log-linear or a recursive logit parametrization is adopted, while in Bartolucci and Forcina (2024) a Fisher-scoring maximization algorithm is suggested. The EM algorithm alternates two steps until convergence:

- **E-step**: for each stratum i, the expected value of the missing frequency $y_{i,\mathbf{0}}$ is computed given the current value of the parameters and the observed data as

$$\hat{y}_{i,\mathbf{0}} = (N-n)\frac{\phi_i \tau_i}{\phi}.$$

Then, all observed and predicted frequencies are split among classes, obtaining

$$\tilde{y}_{h,i,\boldsymbol{r}} = y_{i,\boldsymbol{r}} \frac{\pi_{h,i} q_{h,i,\boldsymbol{r}}}{p_{i,\boldsymbol{r}}}, \quad h \in \mathcal{H},\ i \in \mathcal{I},\ \boldsymbol{r} \in \tilde{\mathcal{R}},$$

with $y_{i,\mathbf{0}} \equiv \hat{y}_{i,\mathbf{0}}$, so that $\hat{N}_i = \hat{y}_{i,\mathbf{0}} + n_i$ is the predicted size for this stratum at the population level.

- **M-step**: the parameters $\boldsymbol{\beta}$ are updated by maximizing the log-likelihood

$$\tilde{\ell}_1(\boldsymbol{\beta}) = \sum_{i \in \mathcal{I}} \tilde{\boldsymbol{n}}_i' \log \boldsymbol{\pi}_i,$$

where $\tilde{\boldsymbol{n}}_i$ is the column vector with elements $\tilde{n}_{h,i} = \sum_{\boldsymbol{r}} \tilde{y}_{h,i,\boldsymbol{r}}$, $h = 1, \ldots, H$. Within this step, the parameters $\boldsymbol{\lambda}$ are updated by maximizing

$$\tilde{\ell}_2(\boldsymbol{\lambda}) = \sum_{h \in \mathcal{H}} \sum_{i \in \mathcal{I}} \tilde{\boldsymbol{y}}_{h,i}' \log \tilde{\boldsymbol{q}}_{h,i},$$

where $\tilde{\boldsymbol{y}}_{h,i}$ is the column vector with elements $\tilde{y}_{h,i,\boldsymbol{r}}$ for all $\boldsymbol{r} \in \tilde{\mathcal{R}}$. The derivatives of functions $\tilde{\ell}_1(\boldsymbol{\beta})$ and $\tilde{\ell}_2(\boldsymbol{\lambda})$ are provided in Appendix depending on the type of model adopted.

The EM algorithm must be properly initialized in order to increase the chance of getting the global maximum of the overall log-likelihood function. As will be illustrated in the following, this may be based on the joint use of deterministic and random starting values; more details are provided in Appendix. We also warn the reader that, as is well known, the EM algorithm may require many iterations as a counterpart to its stability.

Within package **LCCR**, the estimation function also computes the standard errors for the parameter estimates, including the population size N. This is based on first obtaining the observed information matrix as minus the numerical derivative of the score vector with respect to all model parameters, with the exception of parameters $\boldsymbol{\tau}$ that are updated on the basis of the value of the other parameters and are treated as nuisance parameters. Then, this information matrix is dealt with in the usual way to obtain the standard errors. We refer to Bartolucci and Forcina (2024) for details about the computation of the score.

3 Preliminary functions in the package

The approach discussed in this paper is based on a variety of parametrizations regarding the conditional response probabilities given the latent class. Before illustrating the functions in package **LCCR** that may be used for estimation, it is then important to illustrate how to specify the model of interest and the functions that build the design matrices used to define the log-linear and recursive logit parametrizations in (2) and (4). We also illustrate other functions for data simulation and for data manipulation. In this regard, it is important to recall that we consider two possible data formats. The first is adopted with individual data, so that $n_i = 1$, $i \in \mathcal{I}$, and then the observed capture configurations may be equivalently represented by the vectors $\boldsymbol{r}_i$ specified for all individuals. The second format is with aggregated data in which every n_i may be greater than 1 and the observed capture configurations are represented by the frequency vectors $\boldsymbol{y}_i$.

3.1 Model specification

We consider first the log-linear parametrization. The model is formulated by specifying the constraints regarding the intercepts $\gamma_{h,j}$, the vector of regression coefficients $\boldsymbol{\delta}_{h,j}$, and the bivariate interaction parameters η_{h,j_1,j_2}. The possible constraints are listed below.

1. Regarding the intercepts $\gamma_{h,j}$, it is possible to formulate the following assumptions:
   - standard LC model with a separate intercept $\gamma_{h,j}$ for each class and capture occasion (overall HJ parameters);
   - an LC model with the same intercept for each capture occasion, so that $\gamma_{h,j}$ does not depend on j (overall H parameters);
   - a Rasch LC model in which $\gamma_{h,j}$ is decomposed as the sum of a parameter for each latent class, measuring the tendency to be captured, and a parameter for the capture occasion (overall $H - 1 + J$ parameters taking identifiability constraints into account).
2. Regarding the regression coefficients, the following restrictions are considered:
   - same regression coefficients across classes and capture occasions, that is, $\boldsymbol{\delta}_{h,j}$ does not depend either on h or j (overall d parameters, where d is the number of covariates in $\boldsymbol{x}_{i,j}$);
   - regression coefficients that are class specific, that is, $\boldsymbol{\delta}_{h,j}$ does not depend on j (overall dH parameters);
   - regression coefficients that are occasion specific, that is, $\boldsymbol{\delta}_{h,j}$ does not depend on h (overall dJ parameters);
   - free regression coefficients for each latent class and capture occasion (overall dHJ parameters).
3. Regarding the bivariate interactions, the possible constraints are:
   - same interaction for all latent classes, that is, η_{h,j_1,j_2} is constant with respect to j_1 and j_2 (overall 1 parameter);
   - interactions that are latent class specific, that is, η_{h,j_1,j_2} depends only on h (overall H parameters);
   - interactions that are specific for each element in $\mathcal{B}$, that is, η_{h,j_1,j_2} does not depend on h (overall b parameters);
   - free interactions η_{h,j_1,j_2} (overall bH parameters).

The function that builds matrices $\boldsymbol{M}_{h,i}$ in (2) depending on the specified model is

```
design_matrix_loglin(J, H = 1, main = c("LC", "same", "Rasch"), X = NULL,
                     free_cov = c("no", "class", "resp", "both"),
                     biv = NULL, free_biv = c("no", "class", "int", "both"))
```

with the following input arguments:

- `J`: number of capture occasions;
- `H`: number of latent classes;
- `main` to specify the constraints on the intercepts with possible values: `LC` for the LC specification; `same` for the same effect for each capture occasion; and `Rasch` for additive effect of class and capture occasion;
- `X`: array containing covariate vectors $\boldsymbol{x}_{h,i}$ with dimension $S \times d \times J$;
- `free_cov` to specify the constraints on the regression parameters $\boldsymbol{\delta}_{h,i}$ with possible values: `no` for the constant effect with respect to class and capture occasion; `class` for free effects with respect to the class; `resp` for free effects with respect to the capture occasion; `both` for free effect with respect to the class and capture occasion;
- `biv`: matrix of dimension $b \times 2$ containing the list of bivariate interactions;
- `free_biv` to specify the constraint on the interaction parameters δ_{h,j_1,j_2} with possible values: `no` for constant effect with respect to the class and interaction; `class` for free effect with respect to the class; `int` for free effect with respect to interaction; and `both` for free effects with respect to the class and interaction.

Just to clarify the use of this function, suppose that there are 3 capture occasions and a single group covariate equal to 0 and 1. Then, by the following commands we can obtain the design matrices for an LC model with two classes and the effect of the covariate:

```
X = array(c(0,1),c(2,1,3))
M = design_matrix_loglin(3,2,X=X)$M
```

In this way we obtain an array `M` having dimension $8 \times 7 \times 2 \times 2$, where 8 is the number of response configurations, 7 is the number of parameters involved in the conditional distribution of the capture configurations given the latent class, and 2 is both the number of strata and classes.

Regarding the logit recursive model parameters in (3), we may assume similar constraints as above for the parameters $\boldsymbol{\delta}_{h,j}$ and $\eta_{h,j}$. Regarding these parameters, we may assume:

- the same parameter for all latent classes, that is, $\boldsymbol{\delta}_{h,j}$ is constant with respect to h and j (overall d parameters);
- lagged effect that is latent class specific, that is, $\eta_{h,j}$ is independent of j (overall H parameters);
- lagged effect that is capture occasion specific, that is, $\eta_{h,j} = \eta_j$ depends only on j (overall $J-1$ parameters);
- free lagged effect parameters $\eta_{h,j}$ (overall $H(J-1)$ parameters).

In this way, we also properly account for behavioral effects; see also Alunni Fegatelli and Tardella (2016).

Regarding the covariates, it is important to note that when there is only one covariate affecting the class weights, the I values of this covariate in the input argument `X` can be given as a vector, a matrix, or an array, whereas when there are more covariates `X` can be a matrix or an array. When `X` is a vector rather than an array, then it is interpreted as containing the

values of a single covariate that are replicated for all capture occasions. Similarly, when X is a matrix of dimension $I \times d$, then its rows are replicated for all capture occasions.

The function in the package that builds the design matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{M}_{h,i}$ used in (4) is the following:

```
design_matrix_logit(J, H = 1, main = c("LC", "same", "Rasch"), X = NULL,
                    free_cov = c("no", "class", "resp", "both"),
                    flag = c("no", "prev", "sum", "atleast"),
                    free_flag = c("no", "class", "resp", "both"))
```

with the same input arguments as function `design_matrix_loglin()` defined above, apart from the following:

- `flag` to specify the dependence on the lagged responses: `no` for absence of dependence; `prev` for dependence on the previous response; `sum` for dependence on the sum of the previous response variables; and `atleast` to introduce a dummy variable equal to 1 if there is at least one capture in the past;
- `free_flag` to specify the constraints on the parameters for the lagged responses with values: `no` for using only one parameter; `class` when these parameters are free with respect to the latent class; `resp` when they are free with respect to the capture occasion; and `both` when they are free with respect to the class and capture occasion.

3.2 Simulation

The following function may be used to simulate data from one of the models formulated in the section about model assumptions:

```
simLCCR(H, J, be, la, N, model = c("loglin", "logit"), Wc = NULL, Xc = NULL, biv = NULL,
        flag = c("no", "prev", "sum", "atleast"),
        main = c("LC", "same", "Rasch"),
        free_cov = c("no", "class", "resp", "both"),
        free_biv = c("no", "class", "int", "both"),
        free_flag = c("no", "class", "resp", "both"))
```

The input arguments are the following:

- `H`: number of latent classes;
- `J`: number of capture occasions;
- `be`: parameter vector on the class weights;
- `la`: parameter vector on the conditional response probabilities;
- `N`: population size (with individual data) or vector of the size of every stratum in the population (with aggregated data);
- `model`: to specify the model formulation with options `loglin` or `logit`;
- `Wc`: matrix with rows corresponding to the vectors $\boldsymbol{w}_i$ for all sampled individuals;
- `Xc`: array of covariates in $\boldsymbol{x}_{h,i}$ organized as in the input of functions `design_matrix_loglin()` and `design_matrix_logit()` for all sampled individuals;
- `biv`, `main`, `free_cov`, `free_biv`: see the same arguments of function `design_matrix_loglin()` when the log-linear parametrization is adopted;
- `flag`, `main`, `free_cov`, `free_flag`: see the same arguments of function `design_matrix_logit()` when the recursive logit parametrization is adopted.

3.3 Data manipulation

Typically, individual data are available in the form of vectors $\boldsymbol{r}_i$ collected in the matrix $\boldsymbol{R}$. To use the estimation function that will be illustrated below, it is then necessary to transform each of these capture configurations into a frequency vector of type $\boldsymbol{y}_i$ having dimension 2^J. For this aim, with package `LCCR` it is possible to use function

```
freq_data(R,count=rep(1,nrow(R)))
```

that accepts as input the matrix of individual capture configurations and, as an optional argument, a vector of counts for each of these configurations, and provides the matrix of the corresponding frequencies.

Furthermore, it may be convenient to transform individual data into aggregated data, and for this aim it is possible to use function

```
aggr_data(Y,W=NULL,X=NULL)
```

with the following input arguments:

- `Y`: matrix of capture configurations in frequency format;
- `W`: matrix of covariates affecting the class weights;
- `X`: array of covariates affecting the conditional response probabilities given the latent class.

The same arguments are provided in output in aggregated form and called as `Ya`, `Wa`, and `Xa`.

4 Model estimation within the package

4.1 Point estimation

Within the package, the estimation function is

```
estLCCR(Y, H, model = c("loglin", "logit"), W = NULL, X = NULL, N = NULL, biv = NULL,
        flag = c("no", "prev", "sum", "atleast"),
        main = c("LC", "same", "Rasch"),
        free_cov = c("no", "class", "resp", "both"),
        free_biv = c("no", "class", "int", "both"),
        free_flag = c("no", "class", "resp", "both"),
        N0 = NULL, be0 = NULL, la0 = NULL, control = list(),
        verb = TRUE, init_rand = FALSE, se_out = FALSE)
```

The main input arguments are:

- `Y`: matrix of dimension $I \times (2^J - 1)$ with rows corresponding to the observed vectors $\boldsymbol{y}_i$;
- `H`: number of latent classes;
- `model`: to specify the type of parametrization of the capture probabilities given the latent class;
- `W`: matrix with rows corresponding to the vectors $\boldsymbol{w}_i$;
- `X`: array of covariates in $\boldsymbol{x}_{h,i}$ organized as in the input of functions `design_matrix_loglin()` and `design_matrix_logit()`;
- `N`: fixed population size;

- `biv`, `main`, `free_cov`, `free_biv`: see the same arguments of function `design_matrix_loglin()` when the log-linear parametrization is adopted;
- `flag`, `main`, `free_cov`, `free_flag`: see the same arguments of function `design_matrix_logit()` when the recursive logit parametrization is adopted.

The main output arguments are:

- `be`: estimate of the parameters in $\boldsymbol{\beta}$;
- `la`: estimate of the parameters in $\boldsymbol{\lambda}$;
- `lk`: final log-likelihood;
- `N`: estimate of N;
- `AIC`: value of the Akaike Information Criterion (AIC) for model selection Akaike (1973);
- `BIC`: value of the Bayesian Information Criterion (BIC) for model selection Schwarz (1978);
- `tauv`: estimate of the vector of weights of each stratum;
- `phiv`: estimate of the vector probabilities of being never captured for each stratum;
- `Piv`: matrix of the probabilities $\pi_{h,i}$ of the latent classes for each stratum;
- `Q`: array of the conditional probabilities $q_{h,i,\mathbf{r}}$ of the capture configurations given each latent class and stratum;
- `seN`: standard error for the estimate of N;
- `sebe`: vector of standard errors for the estimate of $\boldsymbol{\beta}$;
- `sela`: vector of standard errors for the estimate of $\boldsymbol{\lambda}$.

4.2 Confidence interval on N

One interesting feature of package **LCCR** is the possibility to obtain a profile confidence interval for the population size that is based on the asymptotic results proposed in Liu et al. (2017) and Bartolucci and Forcina (2024). For this aim, it is possible to use the method

```
confint(object, parm = list(), level = 0.95, ...)
```

where:

- `object`: output from function `estLCCR`;
- `parm`: a list containing control arguments for the step length of the N values, range of N values in terms of distance of the log-likelihood from its maximum, and maximum value of this grid as a multiple of the estimate of this parameter;
- level: required confidence level.

The output of this function can be suitably represented as will be shown in the following section about an illustrative example. We warn the user that, given the approach followed to obtain confidence intervals, this function could require a long computing time. However, by suitably setting the arguments in `parm`, it is possible to speed up the process to obtain the confidence interval.

5 Example about victims of trafficking

For the first illustration of the package, which is more focused on data preparation, we make use of data taken from Silverman (2020) about victims of trafficking in the UK in 2013; see also Silverman (2014) for details. Five lists are considered: *LA* (local authorities); NG (non-government organizations); *PFNCA* (police forces and National Crime Agency); *GO* (government organizations); *GP* (general public).

In the format reporting the frequency of every observed capture configuration $\boldsymbol{r}$, the data can be entered as follows:

```
UKdat5 = data.frame(LA = c(1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1),
                    NG = c(0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1),
                 PFNCA = c(0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1),
                    GO = c(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1),
                    GP = c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0),
                      count = c(54, 463, 995, 695, 316, 15, 19, 3, 62, 19,
                                1, 76, 11, 8, 1, 1, 4, 1))
```

Note that the resulting data frame has 18 rows, corresponding to the number of distinct configurations, and 6 columns, as the last column is that of the frequencies. In order to convert these data into the format used in package **LCCR**, based on vectors of type $\boldsymbol{y}_i$ having as elements the frequencies of all possible capture configurations for stratum i, we can proceed as follows:

```
# load package
require(LCCR)

# prepare data
Y = freq_data(as.matrix(UKdat5[,1:5]),UKdat5[,6])
Ya = aggr_data(Y)$Ya
```

In this way we obtain the following matrix of a single row, as there is only one stratum, and 32 columns equal to the number of capture configurations, that is, 2^5:

```
> Ya
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
[1,]    0  316  695    8  995   11   76    0  463     1    19     0    62     0     4     0
   [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30]
[1,]    54     0     3     0    19     0     0     0    15     0     1     0     1     0
     [,31] [,32]
[1,]     1     0
```

These configurations are arranged in lexicographical order, and then, for instance, 316 is the frequency of the second configuration equal to $\boldsymbol{r} = (0,0,0,0,1)'$.

To fit the simple independence model M_t on these data, we can use the following command:

```
est1 = estLCCR(Ya,H=1,se_out=TRUE)
```

obtaining the following output:

```
> summary(est1)

Estimation of latent class models for capture-recapture data

Call:
estLCCR(Y = Ya, H = 1, se_out = TRUE)
```

```
Available objects:
 [1] "beta"      "lambda"    "lk"        "N"         "np"        "AIC"
 [7] "BIC"       "M"         "tauv"      "phiv"      "Piv"       "Q"
[13] "lk1"       "lk2"       "lk3"       "lk4"       "call"      "Y"
[19] "H"         "model"     "W"         "X"         "biv"       "flag"
[25] "main"      "free_cov"  "free_biv"  "free_flag" "se_out"    "seN"
[31] "selambda"

   LogLik        np       AIC       BIC
 14278.46      5.00 -28546.93 -28517.34

Population size:
      est.     s.e.
N 13435.07 805.6171

Parameters affecting the conditional capture probabilities given the latent class:
           est.       s.e.    t-test p-value
main1 -4.955289 0.11983134 -41.35219       0
main2 -3.122125 0.07590029 -41.13456       0
main3 -2.350668 0.07246115 -32.44038       0
main4 -2.750334 0.07340456 -37.46816       0
main5 -3.663166 0.08267378 -44.30868       0
```

The output contains the maximum likelihood at convergence, number of parameters, and AIC and BIC values. Moreover, it reports the estimate of the population size, with standard error, and each probability of capture on the logit scale together with other statistics.

To estimate, on the same data, a simple model with two classes, it is possible to use the command

```
est2 = estLCCR(Ya,H=2,se_out=TRUE)
```

The output of the function can be obtained by the method `summary()` as usual. This output will be illustrated in more detail in the following example about meningitis data. Here, it is important to note that the EM algorithm requires a large number of iterations for both the independence model and the LC model with 2 classes, equal to 134 and 3,926, respectively, with the default tolerance level. However, the computing time is not excessive, being less than 1 second for the first model and 1 minute for the second. Finally, it is also possible to estimate these models using as input the matrix `Y` having dimension 18 and provided by function `freq_data()` as shown above. In this case, however, the computing time is slightly higher.

6 Example based on meningitis data

In order to illustrate the use of the package, we also describe an application about meningitis data collected in an Italian region from 2001 to 2005; we refer to Rossi et al. (2009) for a detailed description and some analyses of the data. Further analyses are reported in Bartolucci and Forcina (2018) and Bartolucci and Forcina (2024) by latent class models for capture-recapture data.

The data are collected in the file 'meningits_data.rda' that we make available through the journal site. The capture occasions are four corresponding to: *HSS* (hospital surveillance of bacterial meningitis); *NDS* (mandatory infectious diseases notifications); *LIS* (the laboratory information system); and *HIS* (hospital information system). There are also some individual covariates: *Age* (binary variable equal to 1 for up to 1 year old and 0 otherwise); *Aez* (binary variable for the recorded type of bacteria, which is equal to 1 for Pneumococcus, Meningococcus, or Tuberculosis and 0 otherwise); *Year* (year of first appearance in a list).

The overall sample size is $n = 944$ with the overall number of captures by list, and also by covariate, shown in Table 1.

	HSS	NDS	LIS	HIS	sample size
overall	355	644	178	826	944
Age=0	261	527	130	689	785
Age=1	94	117	48	137	159
Aez=0	118	290	15	437	516
Aez=1	237	354	163	389	428
Year=2001	46	113	50	149	175
Year=2002	68	112	37	154	175
Year=2003	58	130	34	152	179
Year=2004	66	120	24	182	191
Year=2005	117	169	33	189	224

Table 1: Frequency of captures by list and covariate.

6.1 Analysis without covariates

In the following we show the code to organize the data and estimate the model without covariates from 1 to 3 latent classes, summarizing the main results in an output matrix:

```
# load package
require(LCCR)

# load data
load("meningits_data.rda")
Y = freq_data(D[,1:4])

# estimate with model with 1 to 3 latent classes
est1 = estLCCR(Y,H=1,se_out=TRUE)
est2 = estLCCR(Y,H=2,se_out=TRUE)
est3 = estLCCR(Y,H=3,se_out=TRUE)

# table of results
Tab = rbind(c(k=1,Loglik=est1$lk,np=est1$np,BIC=est1$BIC,N=est1$N),
            c(2,est2$lk,est2$np,est2$BIC,est2$N),
            c(3,est3$lk,est3$np,est3$BIC,est3$N))
```

The output matrix, reporting for each model the value of the maximum log-likelihood, the number of free parameters, the value of BIC, and the estimate of N is as follows:

```
> Tab
     k    Loglik np      BIC         N
[1,] 1 -2948.504  4 5924.409  968.0993
[2,] 2 -2759.193  9 5580.037 1089.0081
[3,] 3 -2754.930 14 5605.762 1234.9012
```

According to the BIC, we select the model with $k = 2$ latent classes. On the other hand, the model with a greater number of classes might not be identifiable. In order to check for the presence of different local maxima, we can also repeat the estimation starting from different parameter values that are randomly generated. For example, with 2 latent classes, we can use the following code:

```
# check starting values
est2r = est2
for(it in 1:5){
  tmp = estLCCR(Y,H=2,init_rand=TRUE)
  if(est2r$lk>est2$lk) est2r = tmp
}
```

According to this analysis, it emerges that the found values of the maximum log-likelihood starting with the deterministic initialization rule are not surpassed by those found starting with the random initialization rule.

In order to obtain a profile confidence interval for the population size, we can simply use command `CI2 = confint(est2)` and then we can show the parameter estimates and the confidence interval by command `summary()` obtaining the output below. The confidence interval can also be represented by command `plot(CI2)`, obtaining the plot in Figure 1. The estimation output is as follows:

```
> summary(est2)

Estimation of latent class models for capture-recapture data

Call:
estLCCR(Y = Y, H = 2, se_out = TRUE)

Available objects:
 [1] "beta"      "lambda"    "lk"        "N"         "np"        "AIC"
 [7] "BIC"       "M"         "tauv"      "phiv"      "Piv"       "Q"
[13] "lk1"       "lk2"       "lk3"       "lk4"       "call"      "Y"
[19] "H"         "model"     "W"         "X"         "biv"       "flag"
[25] "main"      "free_cov"  "free_biv"  "free_flag" "se_out"    "seN"
[31] "sebeta"    "selambda"

   LogLik        np       AIC       BIC
-2759.193     9.000  5536.386  5580.037

Population size:
      est.     s.e.
N 1089.005 32.57444

Parameters affecting the class weights:
                 est.      s.e.    t-test      p-value
class2.int -0.3655014 0.1029695 -3.549609 0.0003858038

Parameters affecting the conditional capture probabilities given the latent class:
                   est.      s.e.     t-test      p-value
class1.main1 -3.7318944 0.4999153  -7.465054 8.326673e-14
class1.main2 -0.7065195 0.1887566  -3.743019 1.818222e-04
class1.main3 -3.1788471 0.2634316 -12.067068 0.000000e+00
class1.main4  0.5757279 0.1988735   2.894946 3.792242e-03
class2.main1  1.1642476 0.2489039   4.677498 2.903962e-06
class2.main2  3.3943767 0.4367170   7.772486 7.771561e-15
class2.main3 -0.6569705 0.1209995  -5.429531 5.650224e-08
class2.main4  2.5736757 0.2194706  11.726744 0.000000e+00

> summary(CI2)

Confidence interval for the population size based on latent class models
```

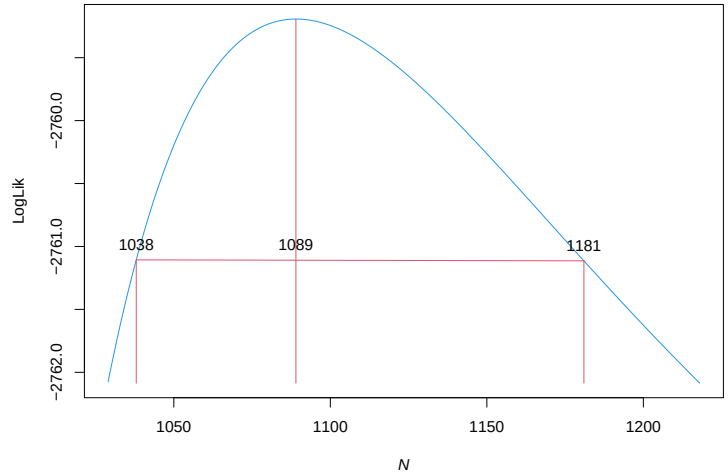


Figure 1: Confidence interval for the latent class model with $k = 2$ classes without covariates.

```
for capture-recapture data

Call:
confint.estLCCR(object = est2, parm = 0.5)

Available objects:
[1] "conf"  "Nv"    "lkv"   "level" "Nh"    "lkh"   "lk1"   "lk2"   "call"

Level:
[1] 0.95

Interval limits:
[1] 1038.005 1181.005
```

Comparing the parameter estimates on the logit scale for the first class (named as `class1.main1` to `class1.main4`) with those for the second class (named as `class2.main1` to `class2.main4`), we note that the latter corresponds to a lower tendency to be captured for all lists. Moreover, considering the intercept of the logit model for the class weights, which is positive, the second class has a higher probability than the first that may be obtained as `exp(est2$be)/(1+exp(est2$be))` and is equal to 0.590. Regarding the estimation of the population size, we have a point estimate of 1,089 with a 95% profile confidence interval equal to (1,037, 1,181).

6.2 Analysis with covariates

In order to illustrate the package for the data with covariates, we consider the model selected for the same data in Bartolucci and Forcina (2018) and also considered in Bartolucci and Forcina (2024). The model, based again on $k = 2$ latent classes, follows a log-linear formulation for the conditional distribution of the capture history given the latent class, with (*i*) main effects depending on the covariate *Year* (centered on 2003) in a way that varies with the latent class and the specific list and (*ii*) bivariate interactions between lists (1,2) and (2,4) that do not depend on the latent class. Moreover, the class weights are affected by the covariates *Age* and *Aez*.

The following commands are used to prepare the data, which are suitably aggregated so as to speed up the routines, and to estimate the model and build the profile confidence interval for the population size.

```
# build covariate matrices
Age = D[,5]
Aez = D[,6]
Year = D[,7]
W = cbind(Age,Aez)
X = Year-2003
agg = aggr_data(Y,W=W,X=X)
Wa = agg$Wa; Xa = cbind(agg$Xa); Ya = agg$Ya
colnames(Xa) = "Year"

# estimate model with covariates
biv = rbind(c(1,2),c(2,4))
est2cov = estLCCR(Ya,model ="loglin",H=2,W=Wa,X=Xa,biv=biv,free_cov="both",
                  free_biv="int",se_out=TRUE)

# estimate model without covariates and aggregated data
est2aggr = estLCCR(Ya,H=2)

# build confidence interval
CI2cov = confint(est2cov,parm=list(step=0.5))
```

In this example, data aggregation is effective because there are $I = 20$ distinct strata despite an overall sample size close to one thousand. Also note that to estimate the model of interest, apart from using arguments `W` and `X`, we use argument `biv` that contains the list of the bivariate interactions, `free_cov="both"` to require that the effect of the covariate *Year* on the main log-linear parameters varies with list and the latent class, and `free_biv="int"` to require that the two interactions are distinct. Finally, the model without covariates is estimated on the aggregated data in order to make a fair comparison in terms of log-likelihood and BIC.

To better understand the model structure, for the model with 2 classes it is also possible to obtain the design matrices used to parametrize the conditional distribution of the capture configurations given the covariates and the latent class as follows:

```
M = design_matrix_loglin(J = 4, H = 2, X = Xa, biv = biv, free_cov= "both",
                         free_biv = "int")$M
```

obtaining an array of dimension $16 \times 18 \times 2$, where 16 is the number of capture configurations, 18 is the number of parameters, 2 is the number of classes, and 20 is the number of strata. Note that these design matrices are directly provided among the output arguments of the estimation function.

By the usual command `summary()`, applied to the previous output objects, it is possible to obtain the main estimation results; by command `plot(est2cov)` it is also possible to represent the profile log-likelihood function with respect to N, which is reported in Figure 2.

```
> summary(est2cov)

Estimation of latent class models for capture-recapture data

Call:
estLCCR(Y = Ya, H = 2, model = "loglin", W = Wa, X = Xa, biv = biv,
    free_cov = "both", free_biv = "int", se_out = TRUE)

Available objects:
 [1] "beta"      "lambda"    "lk"        "N"         "np"        "AIC"
 [7] "BIC"       "M"         "tauv"      "phiv"      "Piv"       "Q"
[13] "lk1"       "lk2"       "lk3"       "lk4"       "call"      "Y"
```

```
[19] "H"         "model"     "W"         "X"         "biv"       "flag"
[25] "main"      "free_cov"  "free_biv"  "free_flag" "se_out"    "seN"
[31] "sebeta"    "selambda"

   LogLik        np       AIC       BIC
 1298.696    21.000 -2555.392 -2453.540

Population size:
      est.     s.e.
N 1358.617 164.8137

Parameters affecting the class weights:
               est.      s.e.    t-test     p-value
class2.int -3.396401 0.3559948 -9.540592 0.000000000
class2.Age  1.196301 0.4632977  2.582143 0.009818901
class2.Aez  3.881855 0.3764615 10.311426 0.000000000

Parameters affecting the conditional capture probabilities given the latent class:
                        est.       s.e.      t-test      p-value
class1.main1      -3.97671435 0.33631698 -11.8243045 0.000000e+00
class1.main2      -1.88968786 0.37771864  -5.0028981 5.647479e-07
class1.main3      -7.63620950 4.54854883  -1.6788232 9.318650e-02
class1.main4      -0.50990995 0.38807709  -1.3139398 1.888665e-01
class2.main1      -1.70265877 0.32081373  -5.3073127 1.112532e-07
class2.main2      -0.64344325 0.44667633  -1.4405134 1.497222e-01
class2.main3      -0.10490127 0.16023619  -0.6546665 5.126825e-01
class2.main4       1.27829624 0.37543619   3.4048296 6.620536e-04
class1.resp1.Year  0.47673522 0.12227804   3.8987805 9.667837e-05
class1.resp2.Year  0.09851558 0.07392841   1.3325809 1.826694e-01
class1.resp3.Year -1.38492913 2.08737446  -0.6634790 5.070238e-01
class1.resp4.Year  0.17408169 0.09860628   1.7654219 7.749288e-02
class2.resp1.Year  0.31957031 0.10915597   2.9276486 3.415358e-03
class2.resp2.Year  0.02815591 0.15996829   0.1760093 8.602866e-01
class2.resp3.Year -0.16880741 0.08695542  -1.9413098 5.222072e-02
class2.resp4.Year  0.09128870 0.17884609   0.5104316 6.097491e-01
biv1-2             2.82970526 0.29756256   9.5096146 0.000000e+00
biv2-4             1.51790409 0.36899015   4.1136710 3.894163e-05

> summary(CI2cov)

Confidence interval for the population size based on latent class models
for capture-recapture data

Call:
confint.estLCCR(object = est2cov, parm = 0.5)

Available objects:
[1] "conf"  "Nv"    "lkv"   "level" "Nh"    "lkh"   "lk1"   "lk2"   "call"

Level:
[1] 0.95

Interval limits:
[1] 1127.617 1827.117
```

From this output, first of all we note that the BIC of the model with covariates is -2,553.5,

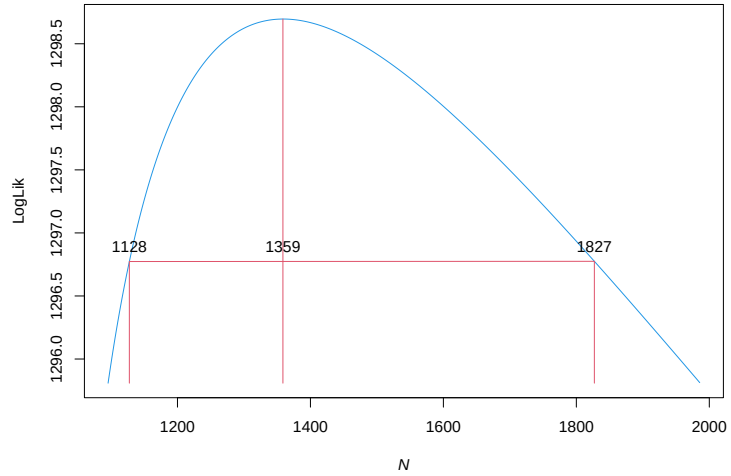


Figure 2: Confidence interval for the latent class model with $k = 2$ classes without covariates.

a value much lower than that of the model without covariates, which is equal to -2,336.7. Moreover, the estimated population size is equal to 1,359 with a 95% confidence interval equal to (1,128, 1,827).

Appendix

Score vector and information matrix

The score vector and the information matrix for $\tilde{\ell}_1(\boldsymbol{\beta})$ are

$$\begin{aligned} \tilde{\boldsymbol{s}}_1(\boldsymbol{\beta}) &= \sum_{i\in\mathcal{I}} \boldsymbol{W}_i'(\tilde{\boldsymbol{n}}_i - \hat{N}_i\boldsymbol{\pi}_i), \\ \tilde{\boldsymbol{F}}_1(\boldsymbol{\beta}) &= \sum_{i\in\mathcal{I}} \hat{N}_i\boldsymbol{W}_i'\boldsymbol{\Omega}(\boldsymbol{\pi}_i)\boldsymbol{W}_i, \end{aligned}$$

where, in general, $\boldsymbol{\Omega}(\boldsymbol{v}) = \text{diag}(\boldsymbol{v}) - \boldsymbol{v}\boldsymbol{v}'$ for any probability vector $\boldsymbol{v}$.

Under the log-linear parametrization, the score vector and information matrix for $\tilde{\ell}_2(\boldsymbol{\beta})$ are

$$\begin{aligned} \tilde{\boldsymbol{s}}_2(\boldsymbol{\lambda}) &= \sum_{i\in\mathcal{I}} \boldsymbol{M}_{h,i}'(\tilde{\boldsymbol{y}}_{h,i} - \tilde{n}_{h,i}\boldsymbol{q}_{h,i}), \\ \tilde{\boldsymbol{F}}_2(\boldsymbol{\lambda}) &= \sum_{i\in\mathcal{I}} \tilde{n}_{h,i}\boldsymbol{M}_{h,i}'\boldsymbol{\Omega}(\boldsymbol{q}_{h,i})\boldsymbol{M}_{h,i}. \end{aligned}$$

Under the recursive logit parametrization, first of all note that on the basis of expression (4) we can reformulate $\tilde{\ell}_2(\boldsymbol{\beta})$ as

$$\tilde{\ell}_2(\boldsymbol{\beta}) = \sum_{h\in\mathcal{H}}\sum_{i\in\mathcal{I}} \tilde{\boldsymbol{y}}_{h,i}'\{\boldsymbol{A}\boldsymbol{M}_{h,i}\boldsymbol{\lambda} - \boldsymbol{B}\log[\boldsymbol{1} + \exp(\boldsymbol{M}_{h,i}\boldsymbol{\lambda})]\}.$$

Consequently, we have the following expression for the score vector and observed information matrix

$$\begin{aligned} \tilde{\boldsymbol{s}}_2(\boldsymbol{\beta}) &= \sum_{h\in\mathcal{H}}\sum_{i\in\mathcal{I}} \boldsymbol{M}_{h,i}'[\boldsymbol{A}' - \text{diag}(\boldsymbol{\rho}_{h,i})\boldsymbol{B}']\tilde{\boldsymbol{y}}_{h,i}, \\ \tilde{\boldsymbol{F}}_2(\boldsymbol{\beta}) &= \sum_{h\in\mathcal{H}}\sum_{i\in\mathcal{I}} \boldsymbol{M}_{h,i}'\text{diag}(\boldsymbol{\rho}_{h,i})\text{diag}(\boldsymbol{1} - \boldsymbol{\rho}_{h,i})\text{diag}(\boldsymbol{B}'\tilde{\boldsymbol{y}}_{h,i})\boldsymbol{M}_{h,i}, \end{aligned}$$

where

$$\boldsymbol{\rho}_{h,i} = \text{diag}[\mathbf{1} + \exp(\boldsymbol{M}_{h,i}\boldsymbol{\lambda})]^{-1} \exp(\boldsymbol{M}_{h,i}\boldsymbol{\lambda}).$$

Initialization of the EM algorithm

As already mentioned, the EM algorithm can be initialized by a deterministic rule, based on the observed data, or a random rule. These rules consist in choosing the starting values as follows:

- N is initialized as $1.25n$ with the deterministic rule and as un, where $u \sim \text{Unif}(0,1)$, with the random rule;
- with the deterministic rule, $\boldsymbol{\lambda}$ is initialized as a vector of zeros of suitable dimension when $H = 1$ and by a suitable transformation of the estimates obtained under this model when $H > 1$; with the random rule, the elements of the initial $\boldsymbol{\lambda}$ are drawn from a standard Normal distribution;
- $\boldsymbol{\beta}$ is initialized from a vector of zeros of suitable dimension with the deterministic rule and as a vector of random numbers drawn from a standard Normal distribution with the random rule.

References

H. Akaike. Information theory as an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki, editors, *Second International Symposium on Information Theory*, pages 267–281, Budapest, 1973. Akademiai Kiado. [p56]

S. Aleshin-Guendel, M. Sadinle, and J. Wakefield. The central role of the identifying assumption in population size estimation. *Biometrics*, 80(1):ujad028, 2024. doi: 10.1093/biomtc/ujad028. [p49]

D. Alunni Fegatelli and L. Tardella. Flexible behavioural capture-recapture modelling. *Biometrics*, 72:125–135, 2016. doi: 10.1111/biom.12417. [p48, 53]

S. Baillargeon and L.-P. Rivest. Rcapture: loglinear models for capture-recapture in R. *Journal of Statistical Software*, 19:1–31, 2007. doi: 10.18637/jss.v019.i05. [p48]

F. Bartolucci and A. Forcina. A class of latent marginal models for capture–recapture data with continuous covariates. *Journal of the American Statistical Association*, 101:786–794, 2006. doi: 10.1198/073500105000000243. [p47]

F. Bartolucci and A. Forcina. Latent class: Rasch models and marginal extensions. In D. Böhning, P. G. M. van der Heijden, and J. Bunge, editors, *Capture-Recapture Methods for the Social and Medical Sciences*, pages 291–304. Chapman and Hall/CRC, 2018. doi: 10.4324/9781315151939-20. [p58, 61]

F. Bartolucci and A. Forcina. Estimating the size of a closed population by modeling latent and observed heterogeneity. *Biometrics*, 80(2):ujae017, 2024. doi: 10.1093/biomtc/ujae017. [p47, 48, 50, 51, 56, 58, 61]

F. Bartolucci and A. Forcina. *LCCR: Latent Class Capture-Recapture Models*, 2025. URL https://cran.r-project.org/web/packages/LCCR/index.html. R package version 2.0.1. [p47]

F. Bartolucci and F. Pennoni. A class of latent Markov models for capture–recapture data allowing for time, heterogeneity, and behavior effects. *Biometrics*, 63:568–578, 2007. doi: 10.1111/j.1541-0420.2006.00702.x. [p47]

D. Böhning, J. Bunge, and P. G. M. van der Heijden. *Capture-Recapture Methods for the Social and Medical Sciences*. Chapman and Hall, 2018. doi: 10.4324/9781315151939. [p47]

R. M. Cormack. Interval estimation for mark-recapture studies of closed populations. *Biometrics*, 48:567–576, 1992. doi: 10.2307/2532310. [p48]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society: Series B*, 39: 1–38, 1977. doi: 10.1111/j.2517-6161.1977.tb01600.x. [p51]

Y. Liu, P. Li, and J. Qin. Maximum empirical likelihood estimation for abundance in a closed population from capture-recapture data. *Biometrika*, 104:527–543, 2017. doi: 10.1093/biomet/asx038. [p47, 50, 56]

D. Manrique-Vallier. Bayesian population size estimation using Dirichlet process mixtures. *Biometrics*, 72:1246–1254, 2016. doi: 10.1111/biom.12502. [p48]

D. Manrique-Vallier. *LCMCR: Bayesian Non-Parametric Latent-Class Capture-Recapture*, 2023. URL https://cran.r-project.org/web/packages/LCMCR/index.html. R package version 0.4.14. [p48]

D. L. Otis, K. P. Burnham, G. C. White, and D. R. Anderson. Statistical inference from capture data on closed animal populations. *Wildlife Monographs*, 64:3–135, 1978. doi: 10.2307/3830650. [p47, 48]

A. B. Owen. *Empirical Likelihood*. Chapman and Hall/CRC, 2001. doi: 10.1201/9781420036152. [p47]

L.-P. Rivest and S. Baillargeon. Applications and extensions of Chao's moment estimator for the size of a closed population. *Biometrics*, 63:999–1006, 2007. doi: 10.1111/j.1541-0420.2007.00779.x. [p48]

P. G. Rossi, J. Mantovani, E. Ferroni, A. Forcina, E. Stanghellini, F. Curtale, and P. Borgia. Incidence of bacterial meningitis (2001–2005) in Lazio, Italy: the results of an integrated surveillance system. *BMC Infectious Diseases*, 9:1–10, 2009. doi: 10.1186/1471-2334-9-13. [p58]

L. Sanathanan. Estimating the size of a multinomial population. *Annals of Mathematical Statistics*, 43:142–152, 1972. doi: 10.1214/aoms/1177692709. [p47]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978. doi: 10.1214/aos/1176344136. [p56]

B. W. Silverman. Modern slavery: an application of multiple systems estimation. Technical report, Home Office, London, 2014. URL https://www.gov.uk/government/publications/modern-slavery-an-application-of-multiple-systems-estimation. [p57]

B. W. Silverman. Multiple-systems analysis for the quantification of modern slavery: classical and Bayesian approaches. *Journal of the Royal Statistical Society, Series A*, 183:691–736, 2020. doi: 10.1111/rssa.12505. [p57]

Francesco Bartolucci
Department of Economics, University of Perugia
Perugia, 06124
Italy
(0000-0001-7057-1421)
francesco.bartolucci@unipg.it

Antonio Forcina
Formerly at University of Perugia
Perugia, 06124
Italy
(0000-0001-5239-5495)
forcinarosara@gmail.com

spheresmooth: An R Package for Penalized Piecewise Geodesic Curve Fitting on a Sphere

by Jae-Hwan Jhong, Seyoung Lee, Ja-Yong Koo, and Kwan-Young Bak

Abstract This paper introduces an R package **spheresmooth**, which implements a penalized piecewise geodesic curve fitting method on a sphere. Spherical data observed over a continuum arise frequently in various fields including cardiology, computer vision, physiology, and geophysics. We propose an adaptive smoothing method by extending the linear spline approach to spherical data. Penalization based on differences of velocity vectors endows sparsity among control points of the spherical curve, which enables data-adaptive curve fitting. The proposed method is implemented with a Riemannian block coordinate descent algorithm. Illustrations on Triassic and Jurassic polar wander data and tropical cyclone data demonstrate practicality of the proposed method and the associated **spheresmooth** package.

1 Introduction

This paper aims to introduce an R package **spheresmooth**, which implements a penalization method for fitting piecewise geodesic curves to spherical data. Spherical data observed over a continuum arise frequently in various fields including cardiology, computer vision, physiology, and geophysics; see, for example, Gould (1969), Jupp and Kent (1987), Kume et al. (2007), Su et al. (2012), Thompson and Clark (1982), and the references therein. We propose an adaptive curve fitting method based on sparsity-inducing penalization, and develop an R package to enable researchers from various fields to easily utilize it for their applications.

Smoothing methods for data observed on a manifold can be broadly categorized into two categories. The methods falling into the first category utilize conventional techniques applied to Euclidean data after mapping the data to the tangent space. Examples include the smoothing spline method based on the rolling and wrapping procedures introduced by Jupp and Kent (1987) and related works employing parallel transport as in Kume et al. (2007) and Kim et al. (2021). The second category involves performing intrinsic optimization on the manifold without explicitly mapping the data to a plane. Approaches based on variational calculus with regularizing constraints fall into this category; for details, refer to Camarinha et al. (1995), Noakes et al. (1989), Samir et al. (2012), Su et al. (2012), Machado et al. (2006). More recently, Bak et al. (2023) introduced an intrinsic curve fitting method based on the generalized Bézier curve and a local penalization scheme on the unit sphere.

The penalized intrinsic Bézier curve fitting method proposed by Bak et al. (2023) has the advantage of adaptively detecting local trends in the data. Furthermore, it takes a form that is easily understandable for researchers in applied fields, akin to an extension of polynomial regression models to the sphere. Spherical spline smoothing methods provide a natural extension of Bézier curve-based approaches. By utilizing spherical splines, one can flexibly model local changes in the data without compromising the overall smoothness. Developing intrinsic spline methods for smoothing spherical data is expected to facilitate the easy understanding and analysis of intricate and dynamically changing patterns of spherical data.

Spherical coordinates and the analysis of spherical data are also highly relevant in astronomy. Many methods and tools for spherical data analysis have been developed and motivated by astronomical observations, such as the Cosmic Microwave Background (CMB) maps produced by the Wilkinson Microwave Anisotropy Probe (WMAP) and Planck satellites. The popular HEALPix (Hierarchical Equal Area isoLatitude Pixelization) scheme

was specifically developed by astronomers for the analysis of CMB data from WMAP (Górski et al., 2005). Such tools have been foundational in handling all-sky observations and have inspired further developments in the field of spherical data analysis. The methods introduced in this paper can be viewed as contributing to this broader context of spherical data applications in fields including astronomy.

In this paper, we consider an intrinsic curve fitting method based on the piecewise geodesic curve and a local penalization scheme on a sphere. The proposed method can be understood as an extension of the linear spline method for spherical data. The performance of the spline-type method is determined by the number and location of the control points. We develop a penalization scheme based on the velocity of the curve to ensure sparsity among the corresponding control points. This allows for the data-adaptive selection of the control points, which guarantees that the curve fits the given spherical data well and identifies the change points on the sphere. The proposed method is implemented with a Riemannian block coordinate descent algorithm.

To the best of our knowledge, no available R packages offer methods for fitting a smooth path to a given set of noisy spherical data observed at known times. We introduce the **spheresmooth** package that implements the proposed piecewise geodesic curve fitting method. This package includes a main function for smoothing spherical data along with several auxiliary functions necessary for data processing. The usages of these functions are explained with examples in detail. Piecewise geodesic curves can be understood as piecewise linear methods on manifolds, making them intuitive and easily applicable for researchers in various fields. Moreover, all processes including the selection of complexity parameters are automatically handled so that users can readily apply the method to data even in exploratory stages. To illustrate the practical utility of the **spheresmooth** package, we provide examples of applying the proposed methodology to Triassic and Jurassic polar wander data for North America (Kent and Irving, 2010) and tropical cyclone data provided by the Regional Specialized Meteorological Center (RSMC) Tokyo Typhoon Center.

Visualization of the generated curve requires loading **sphereplot** (Robotham, 2022), which includes the packages **rgl** (Murdoch and Adler, 2024). These packages are used for generating and manipulating 3D graphics for spherical data. To visualize the fitted curve on a map, we load the packages **rworldmap** (South, 2011) and **ggplot2** (Wickham, 2016). The **sf** package provides a standardized and efficient way to handle spatial data using simple features, making it easy to perform geographic operations and analyses within the R environment.

```
library(spheresmooth)
library(sphereplot)
library(rworldmap)
library(ggplot2)
library(sf)
```

The remainder of this paper is organized as follows. Section 2 collects mathematical preliminaries related to the spherical geometry. Section 3 defines the penalized piecewise curve fitting method and summarizes the implementation scheme based on the Riemannian block coordinate descent algorithm. Section 4 outlines the structure and usage of the **spheresmooth** package, followed by practical application examples in Section 5. Section 6 provides a summary of the results presented in this paper.

2 Preliminaries

This section reviews some basic concepts of Riemannian geometry used in this paper. One may refer to, for example, do Carmo (1976) and do Carmo (1992) for a detailed overview of the relevant mathematical backgrounds.

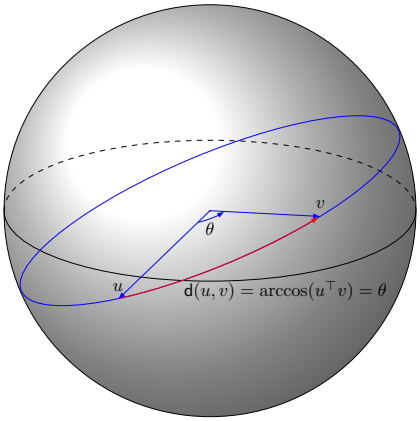


Figure 1: Illustration of geodesic segment between u and v and spherical distance.

Spherical distance and geodesic segment

In this work, the term 'geodesic segment' is equivalent to a 'great circle segment' on a sphere. This distinction is made due to differences in terminology preferences across various academic disciplines.

Let u and v be points on the unit sphere and $\theta, 0 \leq \theta \leq \pi$, be the angle formed by u and v relative to the origin of the sphere. Define the spherical distance between two points u and v as

$$\mathrm{d}(u, v) = \arccos(u^\top v).$$

Observe $\mathrm{d}(u, v) = \theta$ on the unit sphere.

Let $\mathbb{S}$ be the two-dimensional unit sphere embedded in $\mathbb{R}^3$. A geodesic segment $\alpha : I = [0, 1] \to \mathbb{S}$ between two points u and v on $\mathbb{S}$ is defined as

$$\alpha(t; u, v) = \frac{\sin((1-t)\theta)}{\sin(\theta)} u + \frac{\sin(t\theta)}{\sin(\theta)} v, \quad t \in I$$

where $\theta = \mathrm{d}(u, v)$. Figure 1 illustrates a geodesic segment between u and v on $\mathbb{S}$ in the red line. Here, the length of the red line represents the spherical distance between u and v.

Piecewise geodesic curve

The methodology proposed in this paper aims to fit a linear spherical spline or piecewise geodesic curve to spherical data observed over time in a data-adaptive way. To define a piecewise geodesic curve, the time interval needs to be divided into subintervals, and corresponding control points on the sphere should be determined. Given a time interval I, we consider a sequence of knots $\tau_0 < \tau_1 < \tau_2 < \cdots < \tau_J$ that defines subintervals

$$I_0 = [\tau_0, \tau_1), \quad I_1 = [\tau_1, \tau_2), \quad \cdots \quad , \quad I_{J-1} = [\tau_{J-1}, \tau_J).$$

Let $\xi = (\xi_0, \ldots, \xi_J)$ be a set of control points in $\mathbb{S}$ corresponding to the knot sequence $\tau_1, \ldots, \tau_J$. The associated piecewise geodesic (linear spherical spline) curve is defined as

$$\gamma(t; \xi_1, \ldots, \xi_J) = \sum_{j=0}^{J-1} \alpha\left(\frac{t - \tau_j}{\tau_{j+1} - \tau_j}; \xi_j, \xi_{j+1}\right) \mathsf{ind}(t \in I_j),$$

where $\mathsf{ind}(\cdot)$ denotes the indicator function. The sequence of points on the sphere need not necessarily correspond to different time instances. Rather, they represent a collection of spatial locations that may or may not be observed sequentially in time. This is further discussed in Section 4.

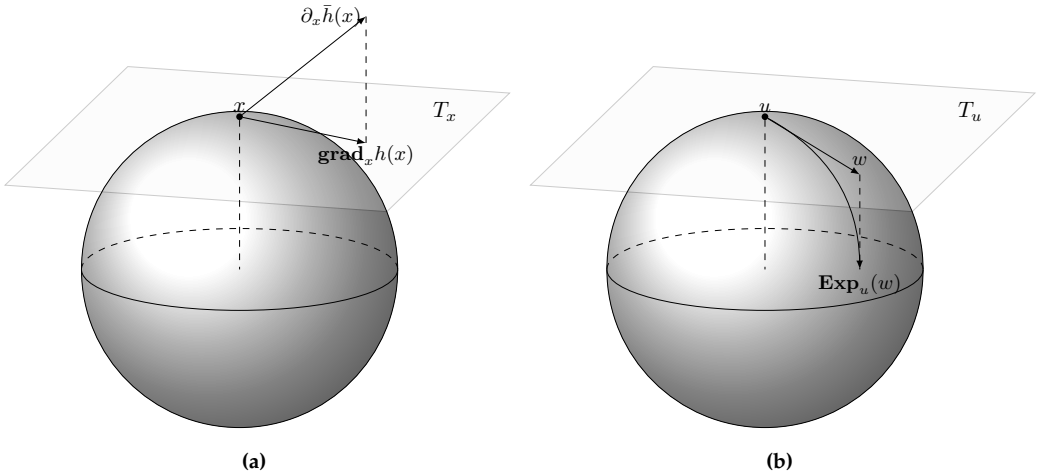


Figure 2: Illustration of (left) Riemannian gradient and (right) exponential map.

Riemannian gradient and exponential map

A blockwise gradient descent algorithm is employed as the algorithm for implementing the spherical smoothing method. The key elements in deriving the algorithm are the Riemannian gradient and the exponential map. The Riemannian gradient is a projection of the Euclidean gradient onto the tangent space. Define the Euclidean gradient as

$$\partial_x \bar{h}(x) = \left[\frac{\partial \bar{h}(x)}{x_j}(x) \right] \quad \text{for} \quad x \in \mathbb{S}, \quad j \leq J$$

where $h : \mathbb{S} \to \mathbb{R}$ and $\bar{h}$ is a differentiable extension of h to $\mathbb{R}^3$. Denote $P_x = I - xx^\top$ as a projection operation onto the tangent space at x. The Riemannian gradient of h with respect to x is defined as

$$\mathsf{grad}_x h(x) = P_x \partial_x \bar{h}(x).$$

The exponential map is a map from a subset of a tangent space $T_u(\mathbb{S})$, $u \in \mathbb{S}$ to $\mathbb{S}$ itself. If $w \in T_u(\mathbb{S})$ is a nonzero vector, a point on $\mathbb{S}$ corresponding to w is denoted as $\mathsf{Exp}_u(w)$. When $w = 0$, then $\mathsf{Exp}_u(w) = u$. The exponential map is a distance-preserving map in the sense that $|u - w| = d(u, \mathsf{Exp}_u(w))$ for $u, w \in \mathbb{S}$, where $|\cdot|$ is the Euclidean ℓ_2 norm. Figure 2 illustrates the Riemannian gradient and exponential map.

3 Penalized piecewise geodesic curve fitting

Penalized piecewise geodesic curve

Let $\{(t_n, y_n)\}_{n=1}^N$ be a set of data, where $t_n \in I$ are the given time points and y_n are the associated data points on $\mathbb{S}$. Our goal is to find a piecewise geodesic curve that fits the data well. To this end, we adopt the squared distance loss function

$$\ell(\xi) = \sum_{n=1}^{N} d^2(y_n, \gamma(t_n; \xi)) \quad \text{for} \quad \xi \in \mathbb{S}^{J+1}$$

where $\mathbb{S}^{J+1}$ is the $J+1$ product of $\mathbb{S}$.

The most important aspect of fitting a piecewise curve is selecting the appropriate number of pieces. The number of pieces is determined by the number of knots and the corresponding control points. To avoid the issues of oversmoothing and undersmoothing, we propose a penalty function that allows us to select the number of control points in a data-adaptive way. This enables the fitted spherical curve to capture local trends in the data

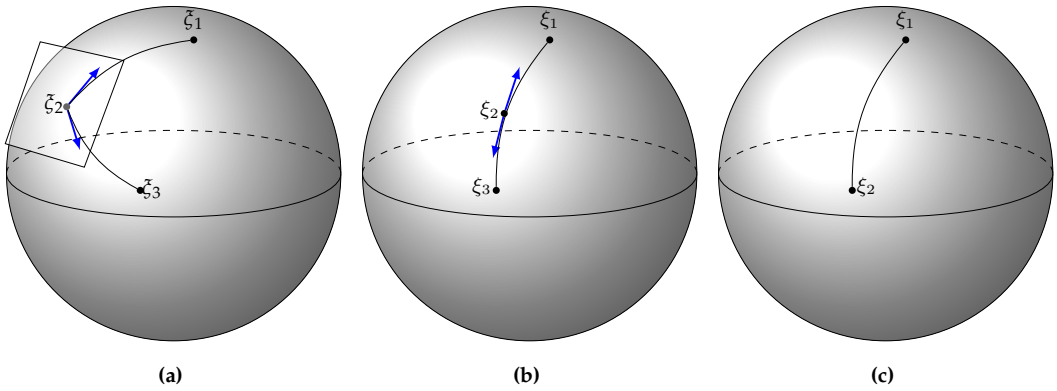


Figure 3: Example of elimination process of the piecewise geodesic curve with three control points. Blue arrow vector indicates tangent vector of ξ_2 in consecutive geodesic segments. The left plot shows the spline curve with small value of complexity parameter. As the complexity parameter λ increases, ξ_2 moves into the geodesic between ξ_1 and ξ_3 and the difference between tangent vectors becomes zero. The middle plot shows that step. When the difference between tangent vectors is zero, control point is removed. The right plot shows the resulting curve after control point is removed in the middle plot.

without compromising the overall smoothness. We introduce a smoothness penalty function

$$\mathsf{p}(\xi) = \sum_{j=1}^{J-1} |D_j(\xi)| \quad \text{for} \quad D_j(\xi) = \frac{1}{\tau_{j+1} - \tau_j} \dot{\alpha}(0; \xi_j, \xi_{j+1}) - \frac{1}{\tau_j - \tau_{j-1}} \dot{\alpha}(1; \xi_{j-1}, \xi_j),$$

which regularizes the number of geodesic segments needed to fit the data. Here, $\cdot$ above a curve denotes its derivative with respect to t and $|\cdot|$ denotes the Euclidean ℓ_2-norm of a vector.

The velocity-based penalty function induces sparsity on the control points. As λ increases, the difference $|D_j(\xi)|$ in the penalty term tends to zero. Suppose that $|D_j(\xi)| = 0$ for some $1 \leq j \leq J-1$ and λ. This implies that the geodesic segments $\alpha(\cdot; \xi_{j-1}, \xi_j)$ and $\alpha(\cdot; \xi_j, \xi_{j+1})$ are on the same great circle. Then, we eliminate the inactive control point ξ_j and replace the two geodesic segments with $\alpha(\cdot; \xi_{j-1}, \xi_{j+1})$. The elimination process reduces the number of the control points by one. Figure 3 illustrates the elimination process that starts with two geodesic segments.

The objective function to minimize is given by

$$\ell^\lambda(\xi) = \ell(\xi) + \lambda \mathsf{p}(\xi) \quad \text{for} \quad \xi \in \mathbb{S}^{J+1},$$

where $\lambda > 0$ is the complexity parameter. For a fixed λ, the penalized piecewise geodesic (linear spherical spline) curve is defined as

$$\gamma_J(\cdot; \hat{\xi}_\lambda),$$

where

$$\hat{\xi}_\lambda = \underset{\xi \in \mathbb{S}^J}{\operatorname{argmin}} \ell^\lambda(\xi).$$

Block coordinate descent algorithm

To obtain $\hat{\xi}_\lambda$, we adopt a block coordinate descent method, where each block consists of the three-dimensional coordinates of each control point. First, we set the initial values of ξ by $J+1$ number of data points y_n associated with (j/J)th quantiles of the time points t_n for $j = 0, 1, \ldots, J$. Denote the initial values by $\xi^{(0)} = (\xi_0^{(0)}, \ldots, \xi_J^{(0)})$. Then, we obtain the Riemannian gradient of ℓ^λ with respect to ξ_j and apply the gradient descent algorithm to

update the control points. Since the updated points belong to a tangent space of $\mathbb{S}$, we apply the exponential function to map the points into $\mathbb{S}$.

Let $\xi_j^{(l)}$ denote the current values of the control point ξ_j in the l-th iteration for $j = 0, \ldots, J$. For $l = 1, 2, \cdots$, the update formula has the following form:

$$\begin{aligned}
\xi_0^{(l)} &\leftarrow \mathsf{Exp}_{\xi_0^{(l-1)}}\left[-\rho_{l,0}\mathsf{grad}_{\xi_0}\ell^{\lambda}\left(\xi_0^{(l-1)}, \xi_1^{(l-1)}, \xi_2^{(l-1)}, \ldots, \xi_J^{(l-1)}\right)\right] \\
\xi_1^{(l)} &\leftarrow \mathsf{Exp}_{\xi_1^{(l-1)}}\left[-\rho_{l,1}\mathsf{grad}_{\xi_1}\ell^{\lambda}\left(\xi_0^{(l)}, \xi_1^{(l-1)}, \xi_2^{(l-1)}, \ldots, \xi_J^{(l-1)}\right)\right] \\
\xi_2^{(l)} &\leftarrow \mathsf{Exp}_{\xi_2^{(l-1)}}\left[-\rho_{l,2}\mathsf{grad}_{\xi_2}\ell^{\lambda}\left(\xi_0^{(l)}, \xi_1^{(l)}, \xi_2^{(l-1)}, \ldots, \xi_J^{(l-1)}\right)\right] \\
&\ldots \\
\xi_J^{(l)} &\leftarrow \mathsf{Exp}_{\xi_J^{(l-1)}}\left[-\rho_{l,J}\mathsf{grad}_{\xi_J}\ell^{\lambda}\left(\xi_0^{(l)}, \xi_1^{(l)}, \xi_2^{(l)}, \ldots, \xi_J^{(l-1)}\right)\right],
\end{aligned}$$

where $\rho_{l,j}$ denotes a specified step size for the update of ξ_j in the lth cycle. The specific form of the update formula is derived below.

As a line search strategy, we adopt the step-halving method to determine the step size $\rho_{l,j}$ to guarantee the stable convergence of the proposed algorithm. When the objective function does not decrease with the current step size, the step size is halved. The step-halving procedure is repeated until the objective function decreases with the reduced step size. The iteration stops when the difference between the current and the updated values of the objective function ℓ^{λ} is less than $\varepsilon = 10^{-5}$.

Derivation of the coordinate-wise update formula

The derivation of the update formula is based on the result in Bak et al. (2023). For notational convenience, denote

$$C(z) = \cos(z), \quad S(z) = \sin(z) \quad \text{and} \quad R_s(z) = \frac{S(sz)}{S(z)}$$

for $s \in I$ and $z \in \mathbb{R}$. With this notation, the parametrized geodesic segment between control points u and v is expressed as

$$\alpha(t; u, v) = R_{1-t}(\theta)u + R_t(\theta)v, \quad t \in I,$$

with $\theta = \mathrm{d}(u, v)$. We first obtain the Euclidean derivative of the geodesic segment with respect to each endpoint.

Theorem 3.1 in Bak et al. (2023) implies that the derivatives of α with respect to the control points are given by

$$\left(\frac{d}{du}\gamma(t; u, v)\right)^{\top} = R_{1-t}(\theta)I + Q_{1-t}(\theta)vu^{\top} + Q_t(\theta)vv^{\top},$$

and

$$\left(\frac{d}{dv}\gamma(t; u, v)\right)^{\top} = R_t(\theta)I + Q_{1-t}(\theta)uu^{\top} + Q_t(\theta)uv^{\top},$$

where

$$Q_s(z) \triangleq \frac{S_s(z)C(z) - sC_s(z)S(z)}{S^3(z)} \quad \text{for} \quad s \in I \quad \text{and} \quad z \in \mathbb{R}.$$

Consider now a piecewise geodesic curve $\gamma(\cdot, \xi)$ defined by a set of control points

$\xi = (\xi_1, \ldots, \xi_J)$. Since the spline curve consists of connected geodesic segments, we have

$$\frac{d}{d\xi_j}\gamma(t;\xi) = \begin{cases} \frac{d}{d\xi_0}\alpha\left(\frac{t-\tau_0}{\tau_1-\tau_0}; \xi_0, \xi_1\right) & \text{for} \quad j = 0 \\ \frac{d}{d\xi_J}\alpha\left(\frac{t-\tau_{J-1}}{\tau_J-\tau_{J-1}}; \xi_{J-1}, \xi_J\right) & \text{for} \quad j = J \\ \frac{d}{d\xi_j}\alpha\left(\frac{t-\tau_{j-1}}{\tau_j-\tau_{j-1}}; \xi_{j-1}, \xi_j\right) + \frac{d}{d\xi_j}\alpha\left(\frac{t-\tau_{j-1}}{\tau_j-\tau_{j-1}}; \xi_j, \xi_{j+1}\right) & \text{o.w.} \end{cases}$$

Combining the results, we are able to obtain the derivatives of $\gamma(\cdot, \xi)$ with respect to each control point. This enables us to compute the Riemannian gradient of the squared distance loss:

$$\mathsf{grad}_{\xi_k}\ell(\xi) = -\sum_{n=1}^{N}\frac{\phi_n}{S(\phi_n)}P_{\xi_k}\left(\frac{d\gamma(t_n;\xi)}{d\xi_k}\right)^\top y_n,$$

where $\phi_n \triangleq \mathsf{d}(y_n, \gamma(t_n;\xi))$.

In addition to the result above, the Riemannian gradient of the velocity difference penalty is needed to derive a coordinate-wise update formula. Following Theorem 3.4 in Bak et al. (2023), for a fixed $1 \le j \le J-1$, we have

$$\mathsf{grad}_{\xi_{j-1}}|D_j(\xi)| = P_{\xi_{j-1}}\frac{1}{\tau_j-\tau_{j-1}}\left[\frac{C(\theta_{j-1})\theta_{j-1}-S(\theta_{j-1})}{S^3(\theta_{j-1})}\xi_j\xi_{j-1}^\top + \frac{\theta_{j-1}}{S(\theta_{j-1})}I\right]\frac{D_j(\xi)}{|D_j(\xi)|},$$

$$\mathsf{grad}_{\xi_{j+1}}|D_j(\xi)| = \frac{1}{\tau_{j+1}-\tau_j}P_{\xi_{j+1}}\left[\frac{C(\theta_j)\theta_j-S(\theta_j)}{S^3(\theta_j)}\xi_j\xi_{j+1}^\top + \frac{\theta_j}{S(\theta_j)}I\right]\frac{D_j(\xi)}{|D_j(\xi)|},$$

and

$$\begin{aligned}\mathsf{grad}_{\xi_j}|D_j(\xi)| = {} & \frac{1}{\tau_{j+1}-\tau_j}P_{\xi_j}\left[\frac{C(\theta_j)\theta_j-S(\theta_j)}{S^3(\theta_j)}\xi_{j+1}\xi_{j+1}^\top - \frac{\theta_j C(\theta_j)}{S(\theta_j)}I\right]\frac{D_j(\xi)}{|D_j(\xi)|} \\ & + \frac{1}{\tau_j-\tau_{j-1}}P_{\xi_j}\left[\frac{C(\theta_{j-1})\theta_{j-1}-S(\theta_{j-1})}{S^3(\theta_{j-1})}\xi_{j-1}\xi_{j-1}^\top - \frac{\theta_{j-1}C(\theta_{j-1})}{S(\theta_{j-1})}I\right]\frac{D_j(\xi)}{|D_j(\xi)|}.\end{aligned}$$

Combining the derivatives of the loss function and the penalty function yields the required Riemannian gradient of the objective function with respect to the control points.

Selection of optimal tuning parameter

The procedure for finding the optimal complexity parameter is as follows. The piecewise geodesic curves that minimize the objective function are computed for an increasing sequence of the complexity parameters $\lambda_1 < \cdots < \lambda_K$, where λ_K is sufficiently large. As the complexity parameter increases, we eliminate the control points that are no longer active.

In general, the choice of the control points plays a crucial role in the spline-type curve fitting problem. We begin by setting λ_K to a sufficiently large value and λ_1 to a value close to zero, ensuring the flexibility of the model. As the complexity parameter increases, the elimination process leads to data-adaptive control point selection. Then, we determine the optimal complexity parameter and the corresponding optimal curve whose control points are adaptively determined by the given set of data.

To choose an optimal complexity parameter, we use a modified version of the Bayes information criterion (BIC); see Schwarz (1978). The BIC for a sequence of the complexity parameters is defined as

$$\mathrm{BIC}_k = N\log\ell(\hat{\xi}_{\lambda_k}) + 3J_k\log N \quad \text{for} \quad k = 1, \ldots, K$$

where J_k denotes the number of control points for λ_k. The optimal value for λ is chosen as $\lambda_{\hat{k}}$, where

$$\hat{k} = \underset{1\le k\le K}{\mathrm{argmin}}\mathrm{BIC}_k.$$

4 The spheresmooth package: structure and functionality

Overall structure and available functions

The **spheresmooth** package is a versatile toolkit designed for performing mathematical and geometric operations on the sphere, with a primary focus on smoothing spherical data observed at known time points. It finds applications across various fields such as geography, geometry, physics, and computer graphics. Table 1 contains a compact summary of the available functions. When the name of a function is in uppercase, it means it computes the output for multiple points simultaneously except for `exp_map()` function. The package includes a corresponding internal function for a single point calculation. For example, the `geodesic()` function computes the coordinates of a geodesic curve at multiple time points, which is the result of multiple internal executions of the `geodesic_lower()` function for a single time point. These functions take arguments in the form of numeric matrices.

Spherical data may be given in the form of spherical coordinates or Cartesian coordinates depending on the case. We may need to switch between the two coordinate systems as necessary. The `cartesian_to_spherical()` function converts Cartesian coordinates to spherical coordinates, which is crucial for representing points on a spherical surface accurately. The `spherical_to_cartesian()` function performs the opposite operation. The input data for the main function must be in the form of Cartesian coordinates. Therefore, if the data is given in spherical coordinates, one can use the `spherical_to_cartesian()` function to perform coordinate transformation before applying the proposed method.

The proposed method is based on the calculation of piecewise geodesic paths. To this end, we need functions that compute the geodesic segments and partition the given time interval. The `geodesic()` function computes the value of the geodesic curve connecting two points p and q on the sphere at specified time points. On the other hand, the `knots_quantile()` function generates a sequence of knots for a given set of time points based on the quantile values, thereby offering support for interpolating and approximating time-series data using spherical spline curves. The `piecewise_geodesic()` function is one of the core functions in the package. It computes the coordinates of the piecewise geodesic curve at the input time point for the given knots in the time interval and the corresponding control points on the sphere.

To evaluate the goodness-of-fit of the fitted curve, a distance-based loss function is needed. The `spherical_dist()` function calculates the geodesic distance between two input points. Based on this function, the `calculate_loss()` calculates the loss function based on the squared spherical distances between observed values and predicted values on the curve. Finally, the main function `penalized_linear_spherical_spline()` fits the penalized piecewise geodesic curve to the given data. In summary, the **spheresmooth** package provides core functions for smoothing spherical data, along with several useful functions for handling spherical data.

Basic functions for handling spherical data

Before delving into the main functions of the package, we introduce several useful functions for handling spherical data within the package. These functions are essential for performing core functionalities of the package but can also be applied to other spherical data analysis tasks. Only a subset of the functions listed in Table 1 is illustrated based on their frequency of use.

The `spherical_dist()` function is used to compute the geodesic distance between two points on the unit sphere. For example, the distance along the geodesic segment between $(1, 0, 0)$ and $(0, 1, 0)$ is one-fourth of the circumference of a great circle containing two points, which is 2π, and it is calculated as follows in code:

```
x <- c(1, 0, 0)
y <- c(0, 1, 0)
```

Table 1: Summary of the functions in the spheresmooth package.

Function	Description
`calculate_loss`	Calculates the loss function based on the squared spherical distances between observed values and predicted values on the curve.
`cartesian_to_spherical`	Converts Cartesian coordinates to spherical coordinates.
`cross`	Computes the cross product of two input vectors
`dot`	Computes the dot product of two input vectors u and v.
`edp`	Computes the equal-distance projection of a point p onto the xy plane.
`exp_map`	Computes the exponential map on the unit sphere given a base point x and a vector v.
`geodesic`	Computes the value of the geodesic curve connecting two points p and q on the unit sphere at specified time points.
`knots_quantile`	Generates a sequence of knots for a given set of time points based on the quantiles.
`norm2`	Computes the L2 norm (Euclidean norm) of the input vector.
`normalize`	Normalizes the rows of the input matrix x by dividing each row by its L2 norm (Euclidean norm).
`penalized_linear_spherical_spline`	Fits a penalized linear spherical spline (piecewise geodesic) curve to the given data.
`piecewise_geodesic`	Computes a piecewise geodesic path between control points.
`spherical_dist`	Calculates the spherical distance between two vectors.
`spherical_to_cartesian`	converts spherical coordinates (θ, ϕ) to Cartesian coordinates.

```
spherical_dist(x, y)

#> [1] 1.570796
```

Consider another example involving the point $(1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$. The antipodal point to this given point is $(-1/\sqrt{3}, -1/\sqrt{3}, -1/\sqrt{3})$, which traverses half the circumference of the associated great circle, resulting in a length of π. This can be checked with the following code.

```
x <- c(1/sqrt(3), 1/sqrt(3), 1/sqrt(3))
y <- c(-1/sqrt(3), -1/sqrt(3), -1/sqrt(3))
spherical_dist(x, y)

#> [1] 3.141593
```

Another useful function is the `cross` function with `normalize = TRUE` argument used to obtain a unit vector orthogonal to a pair of given vectors. For example, we can use this function to obtain a unit vector $(0, 0, 1)$ orthogonal to the plane spanned by $(1, 0, 0)$ and $(0, 1, 0)$.

```
x <- c(1, 0, 0)
y <- c(0, 1, 0)
cross(x, y, normalize = TRUE)

#> [1] 0 0 1
```

Besides standard utility functions, there are two crucial auxiliary functions in spherical data analysis: `cartesian_to_spherical()` and `spherical_to_cartesian()`. These transformations have already been discussed earlier in this section. For basic spherical operations, it is utilized as described previously. When data is given in Cartesian coordinates, the `cartesian_to_spherical()` function computes spherical coordinates, enabling visualization and other analyses. On the other hand, when the data is given in spherical coordinates, the main function of the **spheresmooth** package, `penalized_linear_spherical_spline()`, cannot be directly applied. In such cases, the `spherical_to_cartesian()` function allows the transformation of data into the required format. The following code demonstrates examples implementing these transformations.

```
# example: cartesian_to_spherical
cartesian_points <- matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1), ncol = 3, byrow = TRUE)
cartesian_to_spherical(cartesian_points)

#>         theta      phi
#> [1,] 1.570796 0.000000
#> [2,] 1.570796 1.570796
#> [3,] 0.000000 0.000000

# example: spherical_to_cartesian
theta_phi <- matrix(c(pi/4, pi/3, pi/6, pi/4), ncol = 2, byrow = TRUE)
spherical_to_cartesian(theta_phi)

#>           [,1]      [,2]      [,3]
#> [1,] 0.3535534 0.6123724 0.7071068
#> [2,] 0.3535534 0.3535534 0.8660254
```

The following code demonstrates that these two functions perform inverse operations.

```
theta_phi <- matrix(c(pi/4, pi/3, pi/6, pi/4), ncol = 2, byrow = TRUE)
theta_phi

#>           [,1]      [,2]
#> [1,] 0.7853982 1.0471976
#> [2,] 0.5235988 0.7853982

cartesian_to_spherical(spherical_to_cartesian(theta_phi))

#>          theta       phi
#> [1,] 0.7853982 1.0471976
#> [2,] 0.5235988 0.7853982
```

Piecewise geodesic curve

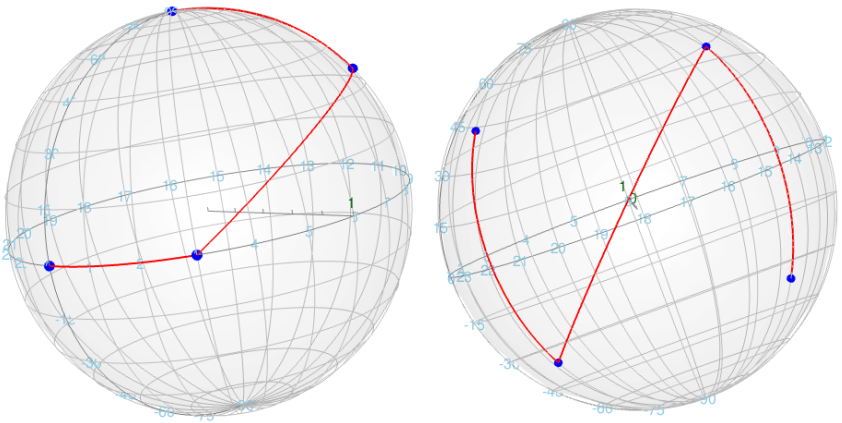
As illustrated in Section 2, we need to provide a sequence of knots and associated control points to define a piecewise geodesic curve. Although we introduce our methodology targeting data observed over time, it is important to note that the argument of a parametrized curve, denoted as t, does not necessarily imply physical time. This methodology can be applied to the smoothing problem of any directional data evolving over a continuous domain. The only requirements for computing a piecewise geodesic curve are the knots and their corresponding control points. The function that returns the coordinates of the curve at a given t point is the `piecewise_geodesic()` function. Table 2 summarizes the arguments of the function.

Table 2: Summary of the arguments of the `piecewise_geodesic()` function.

Argument	Description
t	A numeric vector representing the time or location.
control_points	A matrix of control points where each row represents a control point.
knots	A numeric vector of knot values.

Here, the `control_points` is a matrix where each row represents a point in Cartesian coordinates that determines the geodesic segments. Since a pair of points defines a geodesic segment, this function can be used to compute the geodesic curve between two points if the `knots` is given as two endpoints. The `knots` is a vector of points separating the curve in the time domain. The `piecewise_geodesic()` function performs the following steps. First, it initializes an empty matrix to store the generated points on the curve. Second, it iterates over the separated geodesic segments to find the segment to which the `t` values belong. Then, it internally calls the `geodesic()` function for the corresponding segment to compute the coordinates of those points on the curve and appends the generated points to the initialized matrix. Finally, it returns the matrix containing all the points corresponding to the given `t` values.

In the Figure 4, the left plot depicts a piecewise geodesic curve determined by the control points (1, 0, 0), $(1/\sqrt{2}, 1/\sqrt{2}, 0)$, $(-1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$, and (0, 0, 1). The function `piecewise_geodesic()` is used to compute the coordinates of the curve at given time points. The `piecewise_geodesic` function presented here is roughly equivalent to the `rgl.sphline()` function from the **sphereplot** package, which also generates great circle line segments. However, `piecewise_geodesic` provides additional flexibility for handling complex geometries and segmentations, making it a suitable choice for a broader range of applications. In the following code, we create a variable called `control_points`, which is a matrix containing four control points. The control points are used to determine the curve on the sphere while the `knots` indicate the points where transitions occur in the time domain. To obtain a smooth curve for visualization, we generate `t_example` by slicing the interval



(a) (b)

Figure 4: Plot of (left) piecewise geodesic curve with the control points at (1, 0, 0), ($1/\sqrt{2}$, $1/\sqrt{2}$, 0), (-$1/\sqrt{3}$, $1/\sqrt{3}$, $1/\sqrt{3}$), and (0, 0, 1) and (right) piecewise geodesic curve with the control points at ($1/\sqrt{3}$, $1/\sqrt{3}$, $1/\sqrt{3}$), ($1/\sqrt{3}$, $1/\sqrt{3}$, -$1/\sqrt{3}$), (-$1/\sqrt{3}$, $1/\sqrt{3}$, $1/\sqrt{3}$), and (-$1/\sqrt{3}$, $1/\sqrt{3}$, -$1/\sqrt{3}$).

from 0 to 4 into increments of 0.01, and evaluate the coordinates of the curve at those points using the `piecewise_geodesic()` function. The following code yields the left plot of Figure 4.

```
control_points <- matrix(c(1, 0, 0,                              # Control point 1
                         1/sqrt(2), 1/sqrt(2), 0,                 # Control point 2
                        -1/sqrt(3), 1/sqrt(3), 1/sqrt(3),         # Control point 3
                         0, 0, 1),                                # Control point 4
                         nrow = 4, byrow = TRUE)
knots <- c(1, 2, 3, 3.5)  # Knots indicating transitions
# Example of generating piecewise geodesic curve
t_example <- seq(0, 4, by = 0.01)
gamma_example <- piecewise_geodesic(t_example, control_points, knots)
# Plotting the piecewise geodesic curve
rgl.sphgrid(deggap = 15, col.long = "skyblue", col.lat = "skyblue")
spheres3d(x = 0, y = 0, z = 0, radius = 1, col = "grey", alpha = 0.05)
pch3d(control_points, col = "blue", cex = 0.2, pch = 19)
lines3d(gamma_example, col = "red", lty = 1, lwd = 2)
```

The right plot of Figure 4 is an example of the curve with control points located at ($1/\sqrt{3}$, $1/\sqrt{3}$, $1/\sqrt{3}$), ($1/\sqrt{3}$, $1/\sqrt{3}$, -$1/\sqrt{3}$), (-$1/\sqrt{3}$, $1/\sqrt{3}$, $1/\sqrt{3}$), and (-$1/\sqrt{3}$, $1/\sqrt{3}$, -$1/\sqrt{3}$). Below is the code that computes the curve and draws the corresponding plot.

```
control_points <- matrix(c(1/sqrt(3), 1/sqrt(3), 1/sqrt(3),      # Control point 1
                         1/sqrt(3), 1/sqrt(3), -1/sqrt(3),       # Control point 2
                         -1/sqrt(3), 1/sqrt(3), 1/sqrt(3),       # Control point 3
                         -1/sqrt(3), 1/sqrt(3), -1/sqrt(3)),     # Control point 4
                           nrow = 4, byrow = TRUE)
knots <- c(1, 2, 3, 3.5)  # Knots indicating transitions
# Example of generating piecewise geodesic curve
t_example <- seq(0, 4, by = 0.01)
gamma_example <- piecewise_geodesic(t_example, control_points, knots)
# Plotting the piecewise geodesic curve
rgl.sphgrid(deggap = 15, col.long = "skyblue", col.lat = "skyblue")
spheres3d(x = 0, y = 0, z = 0, radius = 1, col = "grey", alpha = 0.05)
pch3d(control_points, col = "blue", cex = 0.2, pch = 19)
lines3d(gamma_example, col = "red", lty = 1, lwd = 2)
```

Penalized piecewise geodesic curve fitting

The main function `penalized_linear_spherical_spline()` fits a penalized piecewise geodesic curve to the given spherical data as illustrated in Section 3. As implied by the name of the function, this approach can also be understood as using a penalized linear spline method on the sphere. The crucial aspect here is the selection of knots that capture the points of change within the given time domain. The strategy proposed in this paper involves generating a sufficient number of initial knots using the `knots_quantile()` function and then adaptively selecting knots and control points based on penalization. The initial control points corresponding to the generated knots are determined by the corresponding data points y_i on the given spherical surface.

After ensuring the flexibility of the model by setting the number of initial knots and control points sufficiently large, piecewise geodesic curves are computed for an increasing sequence of the complexity parameters $\lambda_1 < \cdots < \lambda_{\max}$. As the complexity parameter increases, we prune or delete the knots that are no longer active. If λ is small, then so is the amount of penalization, resulting in a wiggly fit. On the other hand, the fitted curve gets smoother as λ gets larger. If one wishes to consider as many possibilities as possible to find the optimal curve, it is possible to generate complexity parameters densely starting from very small values, and ensuring that $\lambda_{\max}$ is sufficiently large to yield a least squares geodesic curve. Among the fitted curves, we can select the optimal curve based on the BIC, thus obtaining a data-adaptive piecewise geodesic curve.

The `penalized_linear_spherical_spline()` function is designed to automatically perform all these steps when initial knots and initial control points are not specified, aiming to enhance convenience for researchers. If there are valid numbers of knots determined through exploratory data analysis and preliminary research, along with coordinates for initial control points, they can be used as arguments. Table 3 summarizes the arguments of the function.

Table 3: Summary of the arguments of the `penalized_linear_spherical_spline()` function.

Argument	Description
t	A numeric vector representing the time or location.
y	A matrix where each row represents a data point.
initial_control_points	An optional matrix specifying initial control points. Default is NULL.
dimension	An integer specifying the dimension of the spline.
initial_knots	An optional numeric vector specifying initial knots. Default is NULL.
lambdas	A numeric vector specifying the penalization parameters.
step_size	A numeric value specifying the step size for optimization. Default is 0.01.
maxiter	An integer specifying the maximum number of iterations. Default is 1000.
epsilon_iter	A numeric value specifying the convergence criterion for iterations. Default is 1e-05.
jump_eps	A numeric value specifying the threshold for pruning control points based on jump size. Default is 1e-02.
verbose	A logical value indicating whether to print progress information. Default is FALSE.

The function returns a list containing the fitted result for each complexity parameter, the dimension and BIC values associated with the complexity parameters required for model selection. The BIC values are stored in the last element of the list. It is possible to directly examine the BIC values and their corresponding fitted curves. Unless there is a specific reason, it is recommended to consider the λ that attains the minimum BIC value as the

optimal complexity parameter and select the corresponding fitted curve as the final fitted model. The usage of the function and the model selection process are illustrated in Section 5.

5 Applications

APW Data

The data considered in this section is the polar wander dataset presented in Kent and Irving (2010). They argued that previous estimates of Triassic and Jurassic paleolatitudes for North America tend to be biased because of inclination error in sedimentary rocks. They constructed a new composite APW path for Triassic through Paleogene based on igneous rocks. The 17 Triassic/Jurassic cratonic poles from other major cratons are rotated into North American coordinates and combined with the 14 observations from North America. We apply the proposed method to these 31 observations ranging in age from 243 to 144 Ma (millions of years ago), which covers the late Triassic and Jurassic periods.

The APW dataset is included in the **spheresmooth**. The data consists of the time, longitude, and latitude values for 31 observations. It can be loaded as follows.

```
dim(apw_spherical)
```

```
#> [1] 31  3
```

The APW data is represented in spherical coordinates. To apply the main function, it is necessary to convert the given data into Cartesian coordinates. We apply the `spherical_to_cartesian()` function to the second and third columns of the datasets, since the first column represents the time points.

```
apw_cartesian = spherical_to_cartesian(apw_spherical[, 2:3])
```

To ensure sufficient flexibility of the model, we set the initial dimension to 15, where the dimension refers to the number of knots or the corresponding number of control points. The knot sequence corresponding to the specified dimension is created using the knots_quantile() function. The grid for tuning the complexity parameter is determined as a sequence of 40 values ranging from 10^{-7} to 1 on a logarithmic scale. We compute the fitted curves using the `penalized_linear_spherical_spline()` function.

```
t = apw_spherical[, 1]
dimension = 15
initial_knots = knots_quantile(t, dimension = dimension)
lambda_seq = exp(seq(log(1e-07), log(1), length = 40))

fit = penalized_linear_spherical_spline(t = t, y = apw_cartesian,
                                        dimension = dimension,
                                        initial_knots = initial_knots,
                                        lambdas = lambda_seq)
```

As explained in the previous section, the returned result is a list containing the information about 40 fitted curves and the dimension and BIC values for model selection.

```
class(fit)
```

```
#> [1] "list"
```

```
length(fit)
```

```
#> [1] 42
```

```
fit$dimension_list

#>  [1] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
#> [26] 15 13 13 10 10  9  9  8  8  8  8  8  8  8  8

fit$bic_list

#>  [1] 95.27152 95.22637 95.19061 95.12282 95.08984 95.05878 95.02849 95.00298
#>  [9] 94.97865 94.96560 94.95472 80.58414 80.58430 80.58470 80.58565 80.58750
#> [17] 80.59240 80.60174 80.62544 80.68245 80.77479 80.96411 81.44382 82.18175
#> [25] 83.45483 85.39508 66.62308 67.99973 38.50011 39.57477 29.76234 30.82627
#> [33] 21.74930 32.81671 37.56793 46.46920 47.62672 51.38727 54.17926 54.60989
```

The best model according to the BIC can be selected as follows.

```
# choose a curve that minimizes the BIC
best_index = which.min(fit$bic_list)
best_index

#> [1] 33

# obtained control points for the piecewise geodesic curve
fit[[best_index]]$control_points

#>             [,1]       [,2]      [,3]
#> [1,] -0.48039447 -0.2266981 0.8472480
#> [2,] -0.11927705  0.1300343 0.9843089
#> [3,] -0.06869155  0.1967739 0.9780396
#> [4,]  0.01640654  0.2026538 0.9791130
#> [5,]  0.03231127  0.3512434 0.9357265
#> [6,] -0.03372888  0.5085848 0.8603510
#> [7,] -0.08310953  0.5246722 0.8472378
#> [8,] -0.45295783  0.3987197 0.7974032
```

The optimal complexity parameter is determined as

```
lambda_seq[best_index]

#> [1] 0.0554102
```

and the corresponding dimension is 8. Since we started with the initial dimension 15, the result illustrates the sparsity-inducing property of the proposed method.

The code begins by retrieving a world map using the `getMap()` function, which is then converted to an `sf` object (`worldMap_sf`) using `st_as_sf()` for easier handling in spatial analyses. Control points are transformed from Cartesian coordinates to spherical coordinates using the `cartesian_to_spherical` function, with the resulting coordinates converted to degrees (latitude and longitude) to create `cp_long_lat_df`. The latitude values are adjusted to range from 0 to 90 degrees instead of being measured from the center. The `apw_spherical` data is also converted into a data frame (`apw_spherical_df`), where the latitude and longitude values are transformed from radians to degrees, making them suitable for mapping. Then, a geodesic curve is fitted using the `piecewise_geodesic` function, generating a fitted curve (`fitted_geodesic_curve`). This fitted curve is converted from Cartesian to spherical coordinates (`fitted_cs`), and then transformed into a data frame (`fitted_cs_long_lat_df`) with latitude and longitude values in degrees. Finally, three data frames (`apw_spherical_df`, `cp_long_lat_df`, `fitted_cs_long_lat_df`) are converted to `sf` objects for further geospatial analysis, specifying the appropriate coordinate reference system (EPSG:4326) to facilitate geographic visualizations and operations.

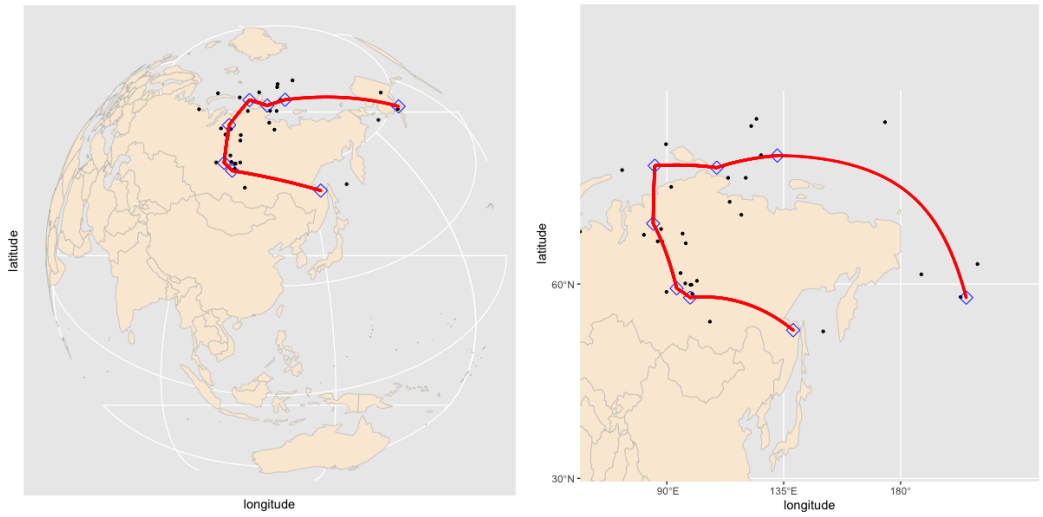
```
worldMap = getMap()
worldMap_sf = st_as_sf(worldMap)

cp_best = cartesian_to_spherical(fit[[best_index]]$control_points)
cp_long_lat = cp_best * 180 / pi
cp_long_lat_df = data.frame(latitude = 90-cp_long_lat[, 1],
                            longitude = cp_long_lat[,2])

apw_spherical_df = data.frame(apw_spherical)
apw_spherical_df$latitude = 90 - apw_spherical_df$latitude * 180 / pi
apw_spherical_df$longitude = apw_spherical_df$longitude * 180 / pi

fitted_geodesic_curve = piecewise_geodesic(seq(0, 1, length = 2000),
                                           fit[[best_index]]$control_points,
                                           fit[[best_index]]$knots)
fitted_cs = cartesian_to_spherical(fitted_geodesic_curve)
fitted_cs_long_lat = fitted_cs * 180 / pi
fitted_cs_long_lat_df = data.frame(latitude = 90 - fitted_cs_long_lat[, 1],
                            longitude = fitted_cs_long_lat[, 2])

apw_spherical_df_sf = st_as_sf(apw_spherical_df,
                               coords = c("longitude", "latitude"), crs = 4326)
cp_long_lat_df_sf = st_as_sf(cp_long_lat_df,
                             coords = c("longitude", "latitude"), crs = 4326)
fitted_cs_long_lat_df_sf = st_as_sf(fitted_cs_long_lat_df,
                              coords = c("longitude", "latitude"), crs = 4326)
```

The optimal fitted curve is visualized in the left plot of Figure 5 using the following code. It presents the obtained APW path (red line) and the associated control points (blue rhombus-shaped points) with the observations in the geographic coordinates. The visualization was generated using the `coord_sf()` function with the `+proj=ortho` option, providing an orthographic projection of the spherical data. The fitted curve shows that the obtained APW path has a clockwise rotational trend.

```
worldmap = ggplot() +
  geom_sf(data = worldMap_sf, color = "grey", fill = "antiquewhite") +
  geom_sf(data = apw_spherical_df_sf, size = 0.8) +
  geom_sf(data = cp_long_lat_df_sf, color = "blue", shape = 23, size = 4) +
  geom_sf(data = fitted_cs_long_lat_df_sf, color = "red", size = 0.5) +
  xlab("longitude") +
  ylab("latitude") +
  scale_y_continuous(breaks = (-2:2) * 30) +
  scale_x_continuous(breaks = (-4:4) * 45) +
 coord_sf(crs = "+proj=ortho +lat_0=38 +lon_0=120 +y_0=0 +ellps=WGS84 +no_defs")
worldmap
```

A zoomed version of the plot obtained from the following code is presented in the right panel of Figure 5.

```
mar = 20
zoommap = ggplot() +
  geom_sf(data = worldMap_sf, color = "grey", fill = "antiquewhite") +
  geom_sf(data = apw_spherical_df_sf, size = 0.8) +
  geom_sf(data = cp_long_lat_df_sf, color = "blue", shape = 23, size = 4) +
  geom_sf(data = fitted_cs_long_lat_df_sf, color = "red", size = 0.5) +
  xlab("longitude") +
```



(a) (b)

Figure 5: Plots of the APW path (red line) and the associated control points (blue points) obtained from the proposed method. The path goes from left to right in the plots. The left plot displays the path on the globe, and the right plot is a zoomed in version of the left plot on the projection map.

```
  ylab("latitude") +
  scale_y_continuous(breaks = (-2:2) * 30) +
  scale_x_continuous(breaks = (-4:4) * 45) +
  coord_sf(xlim = c(min(cp_long_lat_df$longitude) - mar,
                    max(cp_long_lat_df$longitude) + mar),
           ylim = c(min(cp_long_lat_df$latitude) - mar,
                    max(cp_long_lat_df$latitude) + mar))
zoommap
```

Goni Data

This section considers the tropical cyclone (TC) data provided by the Regional Specialized Meteorological Center (RSMC) Tokyo Typhoon Center. This data consists of the variables TC name, time, latitude, longitude, TC central pressure, and maximum sustained wind speed. To illustrate the proposed method and the **spheresmooth** package, we select the data concerning a cyclone named Goni observed from August 13th, 2015, to August 29th, 2015. The number of data points for Goni is 69, corresponding to each time point.

The Goni dataset is also included in the **spheresmooth** and can be loaded as follows.

```
goni_cartesian = spherical_to_cartesian(goni_spherical[, 2:3])
```

The code used for analysis is very similar to that used for computing the APW path. It involves transforming the data given in spherical coordinates to Cartesian coordinates, setting the initial dimension to 15, and fitting it with a sequence of complexity parameters. Subsequently, the optimal curve is selected using the BIC, and the coordinates of the corresponding control points are determined.

```
t = goni_spherical[, 1]
dimension = 15
initial_knots = knots_quantile(t, dimension = dimension)
lambda_seq = exp(seq(log(1e-07), log(1), length = 40))

fit = penalized_linear_spherical_spline(t = t, y = goni_cartesian,
```

```
                                        dimension = dimension,
                                        initial_knots = initial_knots,
                                        lambdas = lambda_seq)
# choose a curve that minimizes the BIC
best_index = which.min(fit$bic_list)
best_index

#> [1] 23

fit$dimension_list[best_index]

#> [1] 12

fit[[best_index]]$control_points

#>             [,1]      [,2]      [,3]
#>  [1,] -0.8572250 0.4708035 0.2085888
#>  [2,] -0.8205531 0.5235703 0.2292744
#>  [3,] -0.7429817 0.6032236 0.2899991
#>  [4,] -0.6430559 0.6947774 0.3221234
#>  [5,] -0.5554791 0.7659919 0.3235729
#>  [6,] -0.5088629 0.7970417 0.3252431
#>  [7,] -0.5041307 0.7884991 0.3523087
#>  [8,] -0.5073276 0.7575879 0.4107058
#>  [9,] -0.5258286 0.7266336 0.4421627
#> [10,] -0.5470298 0.6626324 0.5115434
#> [11,] -0.5418227 0.5904786 0.5981331
#> [12,] -0.4863508 0.5146343 0.7061263
```

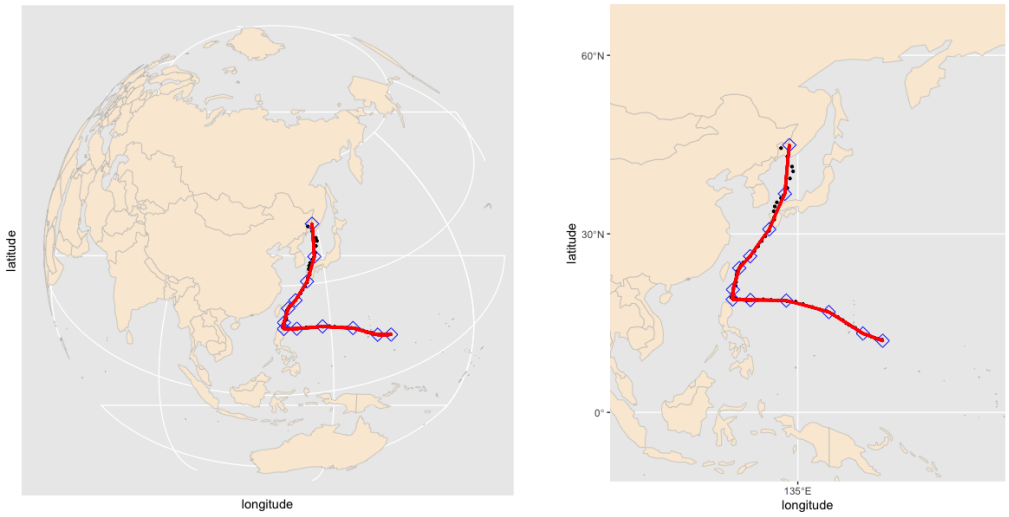
The index of the curve that minimizes BIC is 23 with the corresponding dimension 12. The optimal complexity parameter is determined as

```
lambda_seq[best_index]

#> [1] 0.0008886238
```

We can visualize the fitted curve and obtain Figure 6 by executing code similar to that used in the previous section. The obtained path for Goni (red line) and the corresponding control points (blue rhombus-shaped points) are visualized along with the observations in Figure 6. Although the remaining control points may appear somewhat dense, in reality, these points are not redundant. The proposed penalization method utilizes the difference in velocity vectors to induce sparsity. Therefore, even control points that do not appear visually active on a map can provide valuable information for further analysis when viewed alongside time points, as they represent points where the velocity changes.

6 Summary

This paper introduced the **spheresmooth** package. We proposed a piecewise geodesic curve fitting method based on a velocity-based penalization scheme. The **spheresmooth** package implements the proposed method with the Riemannian block coordinate descent algorithm. It provides an automatic procedure for fitting a smooth path to a given set of noisy spherical data at known times. The **spheresmooth** package demonstrated its usefulness by applying the functions to the polar wander path data and tropical cyclone data. The methods presented in this paper not only advance spherical data analysis in general but also hold significant potential for application in astronomy, where spherical coordinate systems



(a) (b)

Figure 6: Plots of the Goni path (red line) and the associated control points (blue points) obtained from the proposed method. The path goes from left to right in the plots. The left plot displays the path on the globe, and the right plot is a zoomed in version of the left plot on the projection map.

and data are foundational. For example, these methods could be effectively utilized in analyzing celestial data, similar to the work conducted with CMB maps by the WMAP and Planck missions, demonstrating the versatility and applicability of our approach in handling complex, all-sky astronomical observations. We expect that the **spheresmooth** package will be helpful for applications in various fields, including statistics, machine learning, astronomy, cardiology, computer vision, physiology, and geophysics.

7 Acknowledgments

The work of Jae-Hwan Jhong was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(RS-2024-00342014 and RS-2024-00440787). The work of Ja-Yong Koo was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(RS-2023-00253020 and RS-2023-00219212). The work of Kwan-Young Bak was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(RS-2024-00342014 and RS-2022-00165581).

References

K.-Y. Bak, J.-K. Shin, and J.-Y. Koo. Intrinsic spherical smoothing method based on generalized Bézier curves and sparsity inducing penalization. *Journal of Applied Statistics*, 50(9): 1942–1961, 2023. [p67, 72, 73]

M. Camarinha, F. Silva Leite, and P. Crouch. Splines of class C^k on non-Euclidean spaces. *IMA Journal of Mathematical Control and Information*, 12(4):399–410, 1995. [p67]

M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976. [p68]

M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, 1992. [p68]

K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622(2):759, 2005. [p68]

A. L. Gould. A regression technique for angular variates. *Biometrics*, pages 683–700, 1969. [p67]

P. E. Jupp and J. T. Kent. Fitting smooth paths to spherical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 36(1):34–46, 1987. [p67]

D. V. Kent and E. Irving. Influence of inclination error in sedimentary rocks on the Triassic and Jurassic apparent pole wander path for North America and implications for Cordilleran tectonics. *Journal of Geophysical Research: Solid Earth*, 115(B10), 2010. [p68, 80]

K.-R. Kim, I. L. Dryden, H. Le, and K. E. Severn. Smoothing splines on Riemannian manifolds, with applications to 3D shape space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 83(1):108–132, 2021. [p67]

A. Kume, I. L. Dryden, and H. Le. Shape-space smoothing splines for planar landmark data. *Biometrika*, 94(3):513–528, 2007. [p67]

L. Machado, F. S. Leite, and K. Hüper. Riemannian means as solutions of variational problems. *LMS Journal of Computation and Mathematics*, 9:86–103, 2006. [p67]

D. Murdoch and D. Adler. *rgl: 3D Visualization Using OpenGL (R package version 1.3.1)*, 2024. URL https://CRAN.R-project.org/package=rgl. [p68]

L. Noakes, G. Heinzinger, and B. Paden. Cubic splines on curved spaces. *IMA Journal of Mathematical Control and Information*, 6(4):465–473, 1989. [p67]

A. Robotham. *sphereplot: Spherical Plotting (R package version 1.5.1)*, 2022. URL https://CRAN.R-project.org/package=sphereplot. [p68]

C. Samir, P. A. Absil, A. Srivastava, and E. Klassen. A gradient-descent method for curve fitting on Riemannian manifolds. *Foundations of Computational Mathematics*, 12(1):49–73, 2012. [p67]

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978. [p73]

A. South. rworldmap: A new R package for mapping global data. *The R Journal*, 3(1):35–43, 2011. doi: 10.32614/RJ-2011-006. URL https://journal.r-project.org/articles/RJ-2011-006/. [p68]

J. Su, I. L. Dryden, E. Klassen, H. Le, and A. Srivastava. Fitting smoothing splines to time-indexed, noisy points on nonlinear manifolds. *Image and Vision Computing*, 30(6-7):428–442, 2012. [p67]

R. Thompson and R. M. Clark. A robust least-squares Gondwanan apparent polar wander path and the question of palaeomagnetic assessment of Gondwanan reconstruction. *Earth and Planetary Science Letters*, 57(1):152–158, 1982. [p67]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p68]

Jae-Hwan Jhong
ChungBuk National University
Department of Information Statistics
1, Chungdae-ro, Seowon-gu, Cheongju-si, Chungcheongbuk-do, Republic of Korea
ORCiD: 0000-0003-2266-4986

Seyoung Lee
Sungshin Women's University

School of Mathematics, Statistics and Data Science
2, Bomun-ro 34da-gil, Seongbuk-gu, Seoul 02844, Republic of Korea

Ja-Yong Koo
Korea University
Department of Statistics
145, Anam-ro, Seongbuk-gu, Seoul 02841, Republic of Korea

Kwan-Young Bak
Sungshin Women's University
School of Mathematics, Statistics and Data Science
Data Science Center
2, Bomun-ro 34da-gil, Seongbuk-gu, Seoul 02844, Republic of Korea
ORCiD: 0000-0002-4541-160X
kybak@sungshin.ac.kr

Space-Time Smoothing of Survey Outcomes using the R Package SUMMER

by Zehang Richard Li, Bryan D Martin, Tracy Qi Dong, Geir-Arne Fuglstad, Jessica Godwin, John Paige, Andrea Riebler, Samuel J Clark, and Jon Wakefield

Abstract The increasing availability of complex survey data, and the continued need for estimates of demographic and health indicators at a fine spatial and temporal scale, has led to the need for spatio-temporal smoothing methods that acknowledge the manner in which the data were collected. The open source R package SUMMER implements a variety of methods for spatial or spatio-temporal smoothing of survey outcomes. In this paper, we focus primarily on demographic and health indicators. Our methods are particularly useful for data from Demographic Health Surveys (DHS) and Multiple Indicator Cluster Surveys (MICS). We build upon functions within the survey package, and use INLA for fast Bayesian computation. This paper includes a brief overview of these methods and illustrates the workflow of processing surveys, fitting space-time smoothing models for both binary and composite indicators, and visualizing results with both simulated data and DHS surveys.

1 Introduction

A wealth of health and demographic indicators are now collected across the world, and interest often focuses on patterns in space and time. Spatial patterns indicate potential disparities, while temporal trends are important for determining the impact of interventions and to assess whether targets, such as the sustainable development goals (SDGs), are being met (MacFeely, 2020). In low- and middle-income countries (LMIC), the most reliable data with sufficient spatial resolution are often collected under complex sampling designs. Common sources of data include the Demographic Health Surveys (DHS) and Multiple Indicator Cluster Surveys (MICS), both of which use multi-stage cluster sampling. A two-stage cluster design is most common for these surveys. A sampling frame of *clusters* (for example, enumeration areas) is constructed, often from a census, and then strata are formed. The strata consist of some administrative geographical partition crossed with urban/rural (with countries having their own definitions of this dichotomy). Then a pre-specified number of clusters are sampled from these strata under some probabilistic scheme, for example, with probability proportional to size (PPS). Different surveys are powered to different geographical levels. Then, within the selected clusters, households are randomly sampled and individuals are sampled within these households, and asked questions on a range of health and demographic variables. This data collection process must be acknowledged in the analysis to reduce bias and obtain proper uncertainty measures in the prevalence estimates.

Various packages are available within R for small area estimation (SAE) of prevalence, including the **sae** package (Molina and Marhuenda, 2015) that supports the popular book of Rao and Molina (2015) and includes the famous Fay and Herriot (1979) model and spatial smoothing options. Other packages include **rsae** (Schoch, 2014), **hbsae** (Boonstra, 2012), **BayesSAE** (Shi, 2018) and **msae** (Permatasari and Ubaidillah, 2021). A more comprehensive list of related packages is described at *OfficialStatistics*. Most of the existing packages focus on classical SAE models and provide very limited options for fitting spatial and space-time smoothing models.

In this paper we introduce the R package, **SUMMER**[1]. This package and its details are available on CRAN. **SUMMER** provides a computational framework and a collection of tools for smoothing and mapping the prevalence of indicators with complex survey data over space and time, with a special focus on estimating mortality rates. Smoothing is important to avoid unstable estimates and combine information from multiple surveys

[1]The name originally arises from 'Spatio-temporal Under-five Mortality Methods for Estimation in R'. As the package becomes a more general toolkit, it now stands for 'Sae Unit/area Models and Methods for Estimation in R'

over time. Originally developed for small area estimation of the under-5 child mortality rate (U5MR), the **SUMMER** package has been extended to broader mortality rate estimation and more general tasks in SAE. The implemented methods have already been successfully applied to a range of data, e.g., subnational estimates of mortality rates (Mercer et al., 2015; Li et al., 2019; Schlüter and Masquelier, 2021; Fuglstad et al., 2021), HIV prevalence (Wakefield et al., 2020) and vaccination coverage (Dong and Wakefield, 2021). Recently, the **SUMMER** package was used to obtain the official United Nations Inter-Agency Group for Mortality Estimation (UN IGME) yearly estimates (1990–2021) of U5MR at administrative level 2 below the national level (admin-2 estimates) for 31 countries in Africa and Asia (United Nations Inter-agency Group for Child Mortality Estimation, 2023). Previously, the UN IGME only produced national estimates using the *B3 model* (Alkema and New, 2014). The results of these endeavors are available online at `https://childmortality.org`.

The main focus of this paper is to provide an overview of the different prevalence models using survey data and how they can be implemented in **SUMMER**. The rest of the paper is organized as follows. We first briefly describe different methods to estimate prevalence using survey data. We start with a generic binary indicator and proceed with estimating mortality rates. We then provide an overview of the **SUMMER** package and the workflows of using **SUMMER** for prevalence mapping. We then discuss three examples for spatial and space-time smoothing of binary and composite indicators with increasing complexity. The first two examples use simulated data that are included in the **SUMMER** package. The last example uses the most recent DHS survey from Malawi. Then we illustrate various visualization and model checking tools in the **SUMMER** package. Finally, we conclude with future work.

2 Space-time smoothing using complex survey data

In this section we review different methods to estimate the prevalence of a health outcome from complex survey data. We begin by discussing design-based, *direct* estimates (Rao and Molina, 2015) which are based on response data from that area only. Next, we describe space-time smoothing of the direct estimates using a Fay-Herriot model (Fay and Herriot, 1979). We discuss both estimating the prevalence of a single binary indicator and the composite indicators such as U5MR. We then describe a cluster-level model to estimate prevalence at finer spatial and temporal resolutions.

2.1 Estimating the prevalence of a generic binary indicator

Consider a study region that is partitioned into n areas, with interest focusing on estimating the prevalence of a binary indicator in each area, possibly over time. The data are collected via some complex survey design. For each individual j, let y_j denote the individual's outcome, and w_j denote the design weight associated with this individual. Further, let s_{it} represent the indexes of individuals sampled in area i and in time period t. The design-based estimator (Horvitz and Thompson, 1952; Hájek, 1971) is

$$\hat{p}_{it}^{\mathsf{HT}} = \frac{\sum_{j \in s_{it}} w_j y_j}{\sum_{j \in s_{it}} w_j}. \tag{1}$$

This is an example of a direct estimate. The variance of $\hat{p}_{it}^{\mathsf{HT}}$ can be calculated using standard methods (Wolter, 2007) and can be easily computed using the **survey** package. Let V_{it}^{HT} denote the design-based variance of $\text{logit}(p_{it}^{\mathsf{HT}})$, obtained from the design-based variance of p_{it}^{HT} via linearization (the delta method). We take the logit transformed direct estimates as input data and estimate the true prevalence with the random effects model,

$$\text{logit}(\hat{p}_{it}^{\mathsf{HT}}) | \lambda_{it} \sim \text{Normal}(\lambda_{it}, V_{it}^{\mathsf{HT}}), \tag{2}$$

$$\lambda_{it} = x_{it}^{\mathsf{T}} \beta + \alpha_t + \epsilon_t + S_i + e_i + \delta_{it}. \tag{3}$$

In this model, which is a space-time smoothing extension of the Fay and Herriot (1979) model, expit(λ_{it}) is the true prevalence we aim to estimate, and x_{it} are area-level covariates. The rest of the terms are normally distributed random effects including structured time trends α_t, unstructured, independent and identically distributed (iid), temporal terms ϵ_t, structured spatial trends S_i, unstructured spatial terms e_i, and space-time interaction terms δ_{it}. The terms $e_i + S_i$ are implemented via the BYM2 parameterization (Riebler et al., 2016), a reparameterization of the classical BYM model (Besag et al., 1991) that combines iid error terms with intrinsic conditional autoregressive (ICAR) random effects. Several different temporal models are implemented in **SUMMER** for the structured temporal trends and space-time interaction effects, including random walks of order 1 and 2, and autoregressive models (Rue and Held, 2005) with additional linear trends. The interaction term δ_{it} can be one of the type I to IV interactions of the chosen temporal model and the ICAR model in space, as described in Knorr-Held (2000). In order for the model to be identifiable, we impose sum-to-zero constraints on each group of random effects. More details on the prior choices are provided in the supplementary materials.

2.2 Estimating mortality rates using area-level models

For composite indicators such as mortality rates, the direct estimates require additional modeling. Here we focus on the estimation of the U5MR, one of the most critical and widely available population health indicators. The methodology and the functions in **SUMMER** are readily applicable to mortality rates of other age groups as well, but we note that modeling mortality beyond age 5 is usually more challenging in practice because death becomes rarer, and survey data alone are not sufficient for reliable inference.

The **SUMMER** package implements the discrete hazards model described in Mercer et al. (2015). We use discrete time survival analysis to estimate age-specific monthly probabilities of dying in user-defined age groups. We assume constant hazards within the age bands. The default choice uses the monthly age bands

$$[0,1), [1,12), [12,24), [24,36), [36,48), [48,60)$$

for U5MR and they can be easily specified by the user. The U5MR for area i and time t can be calculated as,

$$\hat{p}_{it}^{\mathsf{HT}} = {}_{60}\hat{q}_0^{it} = 1 - \prod_{a=1}^{6} \left(1 - {}_{n_a}\hat{q}_{x_a}^{it}\right), \tag{4}$$

where x_a and n_a are the start and end of the a-th age group, and ${}_{n_a}q_{x_a}^{it}$ is the probability of death in age group $[x_a, x_a + n_a)$ in area i and time t, with ${}_{n_a}\hat{q}_{x_a}^{it}$ the estimate of this quantity. This calculation follows the synthetic cohort life table approach in which mortality probabilities for each age segment based on real cohort mortality experience are combined. This allows the full use of the most recent data, which is especially useful when survey data are sparse and is the default approach that The DHS Program (2020) uses.

The constant one-month hazards in each age band can be estimated by a weighted logistic regression model (Binder, 1983):

$$\text{logit}\left({}_1q_m^{it}\right) = \beta_{a[m]}^{it}, \tag{5}$$

where $a[m]$ is the age band indicator for the m-th month, i.e.

$$a[m] = \begin{cases} 1 & \text{if } m = 0, \\ 2 & \text{if } m = 1, \ldots, 11, \\ 3 & \text{if } m = 12, \ldots, 23, \\ 4 & \text{if } m = 24, \ldots, 35, \\ 5 & \text{if } m = 36, \ldots, 47, \\ 6 & \text{if } m = 48, \ldots, 59. \end{cases} \tag{6}$$

The design-based variance of $\text{logit}(\hat{p}_{it}^{\text{HT}})$ may then be estimated using the delta method or resampling methods such as the jackknife (Pedersen and Liu, 2012). The smoothing of the direct estimates can then proceed using the model described in equations (2) – (3). When multiple surveys exist, one may choose to either model the survey-specific effects as fixed or random (Mercer et al., 2015) or first aggregate the direct estimates from multiple surveys to obtain a "meta-analysis" estimate in each area and time period (Li et al., 2019), i.e., at each time t, we combine the K_t available direct estimates from multiple surveys to form the estimate

$$\hat{p}_{it}^{\text{meta}} = \text{expit}\Big(\sum_{k=1}^{K_t} \frac{(\hat{V}_{it,k}^{\text{HT}})^{-1}}{\sum_{k'=1}^{K_t}(\hat{V}_{it,k'}^{\text{HT}})^{-1}} \text{logit}(\hat{p}_{it}^{\text{HT}})\Big),$$

and the associated design-based variance on the logit scale is $\left(\sum_{k'=1}^{K_t}(\hat{V}_{it,k'}^{\text{HT}})^{-1}\right)^{-1}$. To mitigate the sparsity of available data in each year, Li et al. (2019) also considers a temporal model defined at the yearly level while the direct estimates are calculated at multi-year periods. All these variations can be fit using the **SUMMER** package.

2.3 Estimating mortality rates with cluster-level models

The space-time Fay-Herriot estimates are useful when there are enough observations at the spatial and temporal unit of the analysis. When the target of inference is at finer resolution, e.g., on a yearly time scale with admin-2 areas and surveys stratified at admin-1 levels, the direct estimates may contain many 0s or 1s and the design-based variance cannot be calculated reliably. In this case, we can consider unit-level models where the individual survey responses are modeled. In the rest of this section, we describe a model for the cluster-level risk, where we account for the additional within-cluster variation by allowing overdispersion in the likelihood. More detailed comparisons of this modeling choice were examined in (Dong and Wakefield, 2021). In a two-stage cluster design, the clusters are referred to as primary sampling units (PSUs) and the households are referred to as secondary sampling units (SSUs). Thus we refer to such models as `cluster-level model` to avoid confusion. We describe the model for the mortality estimation problem below, while the same formulation applies to the case of any generic binary indicators as well.

In the most general setting, we consider multiple surveys over time, indexed by k. The sampling frame that was used for survey k, will be denoted by $r[k]$. We assume a discrete hazards model as before. We consider a beta-binomial model for the probability (hazard) of death from month m to $m+1$ in survey k and at cluster c in year t. This model allows for overdispersion relative to the binomial model. Assuming constant hazards within age bands, we assume the number of deaths occurring within age band $a[m]$, in cluster c, time t, and survey k follow the beta-binomial distribution,

$$Y_{a[m],k,c,t} \mid p_{a[m],k,c,t} \sim \text{BetaBinomial}\left(n_{a[m],k,c,t}, p_{m,k,c,t}, d\right), \tag{7}$$

where $p_{m,k,c,t}$ is the monthly hazard at m-th month of age, in cluster c, time t, and survey k and d is the overdispersion parameter. The latent logistic model we use is,

$$\begin{aligned} p_{m,k,c,t} =& \text{expit}(\alpha_{m,c,k,t} + \epsilon_t + b_k), & (8)\\ \alpha_{m,k,c,t} =& \beta_{a[m],r[k],t} I(s_c \in \text{ rural }) + \gamma_{a[m],r[k],t} I(s_c \in \text{ urban }) \\ &+ S_{i[s_c]} + e_{i[s_c]} + \delta_{i[s_c],t} + \text{BIAS}_{k,t}. & (9)\end{aligned}$$

This form consists of a collection of terms that are used for prediction and a number that are not, as we now describe. We include a survey fixed effect b_k with the constraint $\sum_k b_k \mathbf{1}_{r[k]=r} = 0$ for each sampling frame r, so that the main temporal trends are identifiable for each sampling frame. The b_k terms are not included in the prediction, i.e., they are set to zero. The ϵ_t are unstructured temporal effects that allow for perturbations over time. It is a contextual choice whether they are used in predictions. We include terms in (9) that are

analogous to those in equations (2)–(3), in particular the spatial main effects S_i and e_i and the space-time interactions δ_{it}.

For the temporal main effects $\beta_{a[m],r[k],t}$ and $\gamma_{a[m],r[k],t}$, we have stratum-specific distinct trends for each age group $a[m]$ in surveys from each sampling frame. We include separate urban and rural temporal terms to acknowledge the sampling design; often urban clusters are oversampled and have different risk from rural clusters, and so it is important to acknowledge this aspect in the model (Paige et al., 2020). The urban-rural stratification effects may also be parameterized as time-invariant fixed effects, i.e., restricting $\beta_{a[m],r[k],t} = \gamma_{a[m],r[k],t} + \Delta_{a[m],r[k]}$. For a detailed discussion of the parameterization of stratification effects, we refer readers to Wu et al. (2021). In addition, it is usually reasonable to assume shared temporal trends up to a constant shift across some age groups. For example, we may let

$$\beta_{a[m],r[k],t} = \beta_{a[m],r[k],0} + \beta^{\star}_{a^{\star}[m],r[k],t}$$

where $\beta^{\star}_{a^{\star}[m],r[k],t}$ is a collection of temporal random effects with sum-to-zero constraint $\sum_t \beta^{\star}_{a^{\star}[m],r[k],t} = 0$, and $a^{\star}[m]$ is a reduced set of age bands. The default choice for U5MR in the package is

$$a^{\star}[m] = \begin{cases} 1 & \text{if } m = 0, \\ 2 & \text{if } m = 1, \ldots, 11, \\ 3 & \text{if } m = 12, \ldots, 59. \end{cases} \tag{10}$$

That is, we assume the temporal trends for logit hazards in the last four age groups are parallel and only differ by the intercept term $\beta_{a[m],r[k],0}$.

In situations where biases are known for particular surveys and/or years, we can adjust for bias following Wakefield et al. (2019) by including the bias ratio term, $\text{BIAS}_{k,t} = \text{U5MR}^{\star}_t / \widehat{\text{U5MR}}_{k,t}$, where $\text{U5MR}^{\star}_t$ is the expected U5MR in year t and $\widehat{\text{U5MR}}_{k,t}$ is the biased version. This approach has been used to adjust for mothers who have died from AIDS (Walker et al., 2012); such mothers cannot be surveyed, and their children are more likely to have died, so the missingness is informative.

The predicted U5MRs in urban and rural regions of area i and at time t according to sampling frame r are,

$$\text{U5MR}_{i,t,U,r} = 1 - \prod_{a=1}^{6} \left[\frac{1}{1 + \exp(\beta_{a,r,t} + S_i + e_i + \delta_{i,t})} \right]^{z[a]} \tag{11}$$

$$\text{U5MR}_{i,t,R,r} = 1 - \prod_{a=1}^{6} \left[\frac{1}{1 + \exp(\gamma_{a,r,t} + S_i + e_i + \delta_{i,t})} \right]^{z[a]}, \tag{12}$$

where $z[a] = 1, 11, 12, 12, 12, 12$, for the default choice of age bands. The aggregate risk in area i and in year t according to sampling frame r is

$$p_{itr} = q_{itr} \times \text{U5MR}_{i,t,U,r} + (1 - q_{itr}) \times \text{U5MR}_{i,t,R,r}, \tag{13}$$

where q_{itr} and $1 - q_{itr}$ are the proportions of the under-5 population in area i that are urban and rural in year t according to the classification of sampling frame r. The final aggregation over different sampling frames can be done using meta-analysis combination, so that,

$$\widehat{\text{U5MR}}_{it} = \text{expit}\left(\sum_r w_{itr} \times \text{logit}(p_{itr}) \right),$$

where $w_{itr} = U^{-1}_{itr} / \sum_{r'} U^{-1}_{itr'}$ is the scaled inverse of U_{itr}, which is the posterior variance of $\text{logit}(\widehat{\text{U5MR}}^{(r)}_{it})$. Beyond point estimates, we obtain the full posterior of U5MR_{it}, and various summaries can be reported or mapped. The estimate constructed for U5MR is not relevant to any child, because that child would have to experience the hazards for each age group simultaneously in time period t, rather than moving through age groups over multiple time periods. Nevertheless, the resultant U5MR is a useful summary and the

conventional measure that is used to inform on child mortality.

3 Overview of SUMMER

The **SUMMER** package provides a collection of functions for SAE with complex survey data. The package can be installed in R directly by

```
install.packages("SUMMER")
```

The **SUMMER** package requires the **INLA** package (Rue et al., 2009) to be installed. All analyses in this package are conducted with **SUMMER** package version 1.4.0 and **INLA** version 24.03.09. **INLA** can be installed with

```
install.packages("INLA", repos=c(getOption("repos"),
                INLA="https://inla.r-inla-download.org/R/stable"), dep=TRUE)
```

The **SUMMER** package implements a variety of space-time smoothing models using survey data. There are three main functions to implement these models, discussed below and in the three examples in the following sections.

- `smoothSurvey()` produces direct and Fay-Herriot estimates for a generic binary indicator from raw survey data.
- `smoothDirect()` takes direct estimates as input and produces the Fay-Herriot estimates for mortality estimation discussed in Li et al. (2019).
- `smoothCluster()` performs cluster-level smoothing using the Beta-Binomial model discussed in the previous section, and with more details discussed in Wu et al. (2021) and Fuglstad et al. (2021).

We note that `smoothDirect()` and `smoothCluster()` include many more features in modeling composite indicators such as child mortality. In comparison, `smoothSurvey()` can only model generic binary indicators, but it has a simpler interface and workflow, which is appealing for broader communities of practitioners.

The main sources of data required for these methods are the survey data and the corresponding spatial adjacency matrix, which can be derived from spatial polygons data describing region boundaries. For cluster-level modeling, we also need to know which region each cluster belongs to. In the context of modeling DHS data, the survey data and cluster locations are usually recorded in separate files. Figure 1 shows schematically the workflow of data processing and smoothing for generic binary indicators and mortality estimates using the **SUMMER** package. In this paper, our workflow starts with the birth records and GPS files in .dta files as an example. Such files can be directly downloaded from the DHS data portal. It is also straightforward to load data in other formats and supply the R objects into the functions. The entire pipeline of analysis can be carried out using functions in **SUMMER**. The analysis of the main paper can be reproduced without registering for data access. We include a more extensive analysis of DHS data in the supplementary materials, which requires registration with DHS program for data access.

Before demonstrating the utilities of these functions in the following examples, we first load the packages for the analysis, data processing and visualization. For the analysis presented in this paper, we use the **ggplot2** package (Wickham, 2016) and **patchwork** package (Pedersen, 2019) to make further customizations to the visualization produced by **SUMMER**.

```
library(SUMMER)
library(ggplot2)
library(patchwork)
```

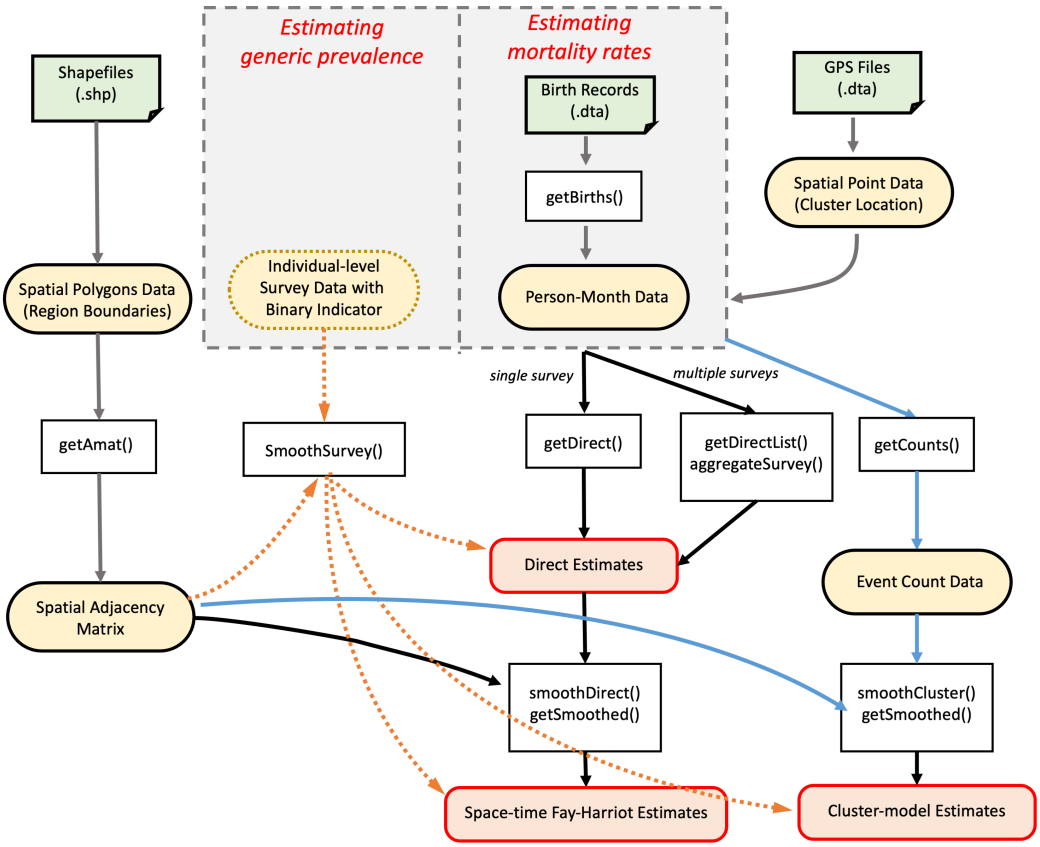


Figure 1: Workflow of the SUMMER package. Rounded blocks represent data types and rectangular blocks represent functions in the SUMMER package. Output estimates are highlighted in the boxes with red borders. The dotted yellow arrows represent the workflow using `smoothSurvey()` to estimate the prevalence of a generic binary indicator. The black solid arrows represent the workflow using `smoothDirect()` to perform area-level smoothing of mortality rates. The blue solid arrows represent the workflow using `smoothCluster()` to perform cluster-level smoothing of mortality rates.

4 Example 1: prevalence estimation for a binary indicator

We start by considering the simplest scenario of estimating the prevalence of a binary indicator using a dataset from the Behavioral Risk Factor Surveillance System (BRFSS) survey. BRFSS is an annual telephone health survey conducted by the Centers for Disease Control and Prevention (CDC) that tracks health conditions and risk behaviors in the United States and its territories since 1984. The BRFSS sampling scheme is complex, with high variability in the sampling weights. In this example, we estimate the prevalence of Type II diabetes in health reporting areas (HRAs) in the King County of Washington using BRFSS data. We will compare the direct estimates and the Fay-Herriot estimates.

The `BRFSS` dataset in **SUMMER** contains the full BRFSS dataset with 16,283 observations. The `diab2` variable is the binary indicator of Type II diabetes, `strata` is the strata indicator, and `rwt_llcp` is the final design weight. For the purpose of this analysis, we first remove records with missing HRA codes or diabetes status from this dataset.

```
data(BRFSS)
data <- subset(BRFSS, !is.na(diab2) & !is.na(hracode))
```

The `KingCounty` dataset in **SUMMER** contains the map of the HRAs in the King County. We first extract the spatial adjacency matrix for the HRAs using the `getAmat()` function.

```
data(KingCounty)
KingGraph <- getAmat(KingCounty, KingCounty$HRA2010v2_)
```

We then use the `smoothSurvey()` function to obtain both the direct and Fay-Herriot estimates by HRA. The function requires specifications of the variables that determine the survey design, including sampling weights (`weightVar`), strata indicator (`strataVar`), and cluster identifiers (`clusterVar`). In this dataset, there are no clusters so we use the formula `~1` in this situation (Lumley, 2004). We also need to specify region indicators (`regionVar`) in the data frame that match the column and row names of the spatial adjacency matrix.

```
fit.BRFSS <- smoothSurvey(data = data, Amat = KingGraph,
                          response.type = "binary", responseVar = "diab2",
                          strataVar="strata", weightVar="rwt_llcp",
                          regionVar="hracode", clusterVar = "~1")
head(fit.BRFSS$direct, n = 3)
```

```
#>      region direct.est direct.var direct.logit.est direct.logit.var direct.logit.prec
#> 1 Auburn-North      0.10   0.00046          -2.2         0.053           18.8
#> 2 Auburn-South      0.23   0.00240          -1.2         0.075           13.3
#> 3     Ballard      0.07   0.00050          -2.6         0.115            8.7
```

```
head(fit.BRFSS$smooth, n = 3)
```

```
#>      region  mean    var median lower upper logit.mean logit.var logit.median
#> 1 Auburn-North 0.102 0.00026  0.101 0.074 0.137      -2.2    0.031       -2.2
#> 2 Auburn-South 0.160 0.00092  0.157 0.108 0.226      -1.7    0.051       -1.7
#> 3     Ballard 0.059 0.00018  0.057 0.037 0.089      -2.8    0.057       -2.8
#>   logit.lower logit.upper
#> 1        -2.5        -2.5
#> 2        -2.1        -2.1
#> 3        -3.3        -3.3
```

The fitted object is of class `SUMMERmodel.svy`, and the direct (`$direct`) and Fay-Herriot estimates (`$smooth`) are saved as data frames in the fitted objects. Notice that since the analysis is performed on the logit of the prevalence, estimates on both the logit and the probability scales are returned in the output. We use the `mapPlot()` function in **SUMMER** to show the estimates on a map. In essence, the `mapPlot()` function takes a data frame, a `SpatialPolygonsDataFrame` object, the column names indexing regions in both the data frame and the polygons, and returns a `ggplot` object. Additional function arguments are available to more easily customize the visualizations. Figure 2 compares the point estimates on the map and the effect of spatial smoothing can be easily seen.

```
g1 <- mapPlot(fit.BRFSS$direct, geo = KingCounty,
              by.data = "region", by.geo = "HRA2010v2_",
              variables = "direct.est", label = "Direct Estimates",
              legend.label = "Prevalence", ylim = c(0, 0.24))
g2 <- mapPlot(fit.BRFSS$smooth, geo = KingCounty,
              by.data = "region", by.geo = "HRA2010v2_",
              variables = "median", label = "Fay Herriot Estimates",
              legend.label = "Prevalence", ylim = c(0, 0.24))
g1 + g2
```

Similar analysis can be implemented for Gaussian observations, and can include temporal smoothing or covariates in the smoothing model. The hyperpriors can also be further customized. For more details and examples, we refer the readers to the package documentation under the `smoothSurvey()` section.

5 Example 2: area-level model of NMR and U5MR using simulated data

In the second example, we consider estimating mortality rates using multiple surveys. We use the NMR and U5MR as two examples, but the implementation in the **SUMMER** package

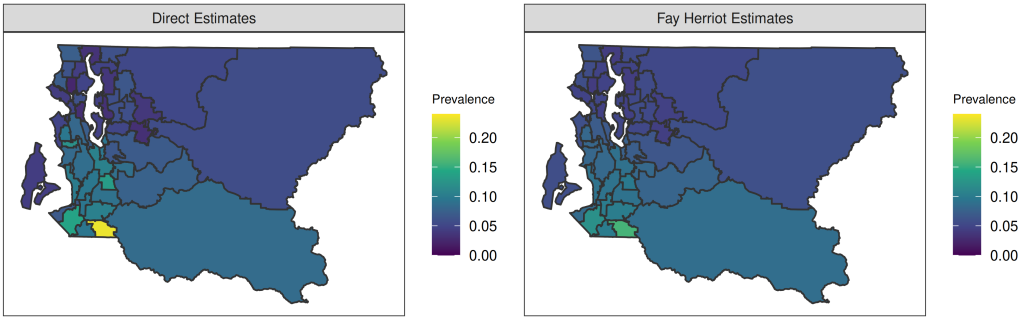


Figure 2: Direct and Fay-Herriot estimates of the prevalence of Type II diabetes in King county HRAs.

allows straightforward extensions to other age groups. We use a simulated survey dataset in this example. A more detailed case study using cluster-level models is provided in the supplementary materials.

We load the `DemoData` dataset from the **SUMMER** package. The `DemoData` is a list that contains full birth history data from simulated surveys with stratified cluster sampling design, similar to most of the DHS surveys. It has been pre-processed into the person-month format, where for each list entry, each row represents one person-month record. Each record contains columns for the cluster ID (`clustid`), household ID (`id`), strata membership (`strata`) and survey weights (`weights`). The region and time period associated with each person-month record has also been pre-computed. The age variable in this data frame is in the form of a_1-a_2, i.e., 1-11 corresponds to age group with 1 to 11 completed months, whereas age groups with only one month are stored using a single number representation, e.g., age group 0. This is also the data structure in the output of the `getBirths` function in the **SUMMER** package. In all the analyses of this paper, we use the default age bands as described before. If a different set of age bands is desired, they can be specified by the `month.cut` argument in the `getBirths` function.

```
data(DemoData)
head(DemoData[[1]])
```

```
#>   clustid id  region  time  age weights        strata died
#> 1       1  1 eastern 00-04    0     1.1 eastern.rural    0
#> 2       1  1 eastern 00-04 1-11     1.1 eastern.rural    0
#> 3       1  1 eastern 00-04 1-11     1.1 eastern.rural    0
#> 4       1  1 eastern 00-04 1-11     1.1 eastern.rural    0
#> 5       1  1 eastern 00-04 1-11     1.1 eastern.rural    0
#> 6       1  1 eastern 00-04 1-11     1.1 eastern.rural    0
```

In order to compute NMR, we create a new list of surveys with only deaths within age group 0.

```
DemoDataNMR <- DemoData
for(i in 1:length(DemoData)){
    DemoDataNMR[[i]] <- subset(DemoData[[i]], age == "0")
}
```

We now turn to the estimation of NMR and U5MR using four simulated surveys in `DemoData`. For multiple surveys, we combine the person-month records into a list and use the `getDirectList()` function to obtain the survey-specific direct estimates. When there are no deaths in a given area and time period, or when more than half of the age groups do not exist in the person-month data, the direct estimates cannot be reliably computed and are set to NA. When only a small fraction of the age groups are not observed, they will be combined with the previous age groups when fitting the discrete hazard model.

```
periods <- c("85-89", "90-94", "95-99", "00-04", "05-09", "10-14")
directNMR <- getDirectList(births = DemoDataNMR, years = periods,
                           regionVar = "region", timeVar = "time",
                           clusterVar = "~clustid + id", ageVar = "age",
                           weightsVar = "weights")
directU5 <- getDirectList(births = DemoData, years = periods,
                          regionVar = "region", timeVar = "time",
                          clusterVar = "~clustid + id", ageVar = "age",
                          weightsVar = "weights")
```

The direct estimates from multiple surveys can be combined to produce a "meta-analysis" estimator using the `aggregateSurvey()` function.

```
directNMR.comb <- aggregateSurvey(directNMR)
directU5.comb <- aggregateSurvey(directU5)
```

Once the direct estimates are calculated, we fit the space-time Fay-Herriot model in the same fashion as in the previous example. The argument `year.label` specifies the order of the years column in the direct estimates, so that it does not have to be integer valued, and can easily allow extensions to future and past time periods not in the data. We can also fit the temporal model at the yearly level even though the direct estimates are in five year periods (Li et al., 2019). In this case, we need to specify the proper range of the time periods (`year_range`) encoded by the time periods in `year.label`, and the number of years in each period (`m`). Unequal periods are not supported at this time. The `smoothDirect()` function returns a fitted object of class `SUMMERmodel`.

```
fhNMR <- smoothDirect(data = directNMR.comb, Amat = DemoMap$Amat,
                  year.label = c(periods, "15-19"), year.range = c(1985, 2019),
                     time.model = "rw2", type.st = 4, is.yearly = TRUE, m = 5)
fhU5 <- smoothDirect(data = directU5.comb, Amat = DemoMap$Amat,
                  year.label = c(periods, "15-19"), year.range = c(1985, 2019),
                     time.model = "rw2", type.st = 4, is.yearly = TRUE, m = 5)
```

The Fay-Herriot estimates can be summarized by the `getSmoothed()` function. The desired posterior credible intervals are specified by the `CI` argument. It organizes the estimates into a data frame of class `SUMMERproj`, which can be directly viewed or plotted using the `plot` method. Additional customization can be added using the syntax of **ggplot2**, as shown in Figure 3.

```
est.NMR <- getSmoothed(fhNMR, CI = 0.95)
est.U5 <- getSmoothed(fhU5, CI = 0.95)
g3 <- plot(est.NMR, per1000 = TRUE) + ggtitle("NMR")
g4 <- plot(est.NMR, per1000 = TRUE, plot.CI=TRUE) + facet_wrap(~region)
g5 <- plot(est.U5, per1000 = TRUE)  + ggtitle("U5MR")
g6 <- plot(est.U5, per1000 = TRUE, plot.CI=TRUE) + facet_wrap(~region)
(g3 + g4) / (g5 + g6)
```

6 Example 3: cluster-level model of U5MR using Malawi DHS data

We now consider a more realistic example of estimating U5MR at the admin-2 level using the 2015–2016 Malawi DHS survey. The full dataset is available on the DHS website at https://dhsprogram.com/data/available-datasets.cfm?ctryid=24. Access to the full micro-level data requires registration with the DHS. Once access is approved, the **rdhs** (Watson and Eaton, 2019) package can be used to load data directly from the DHS API in

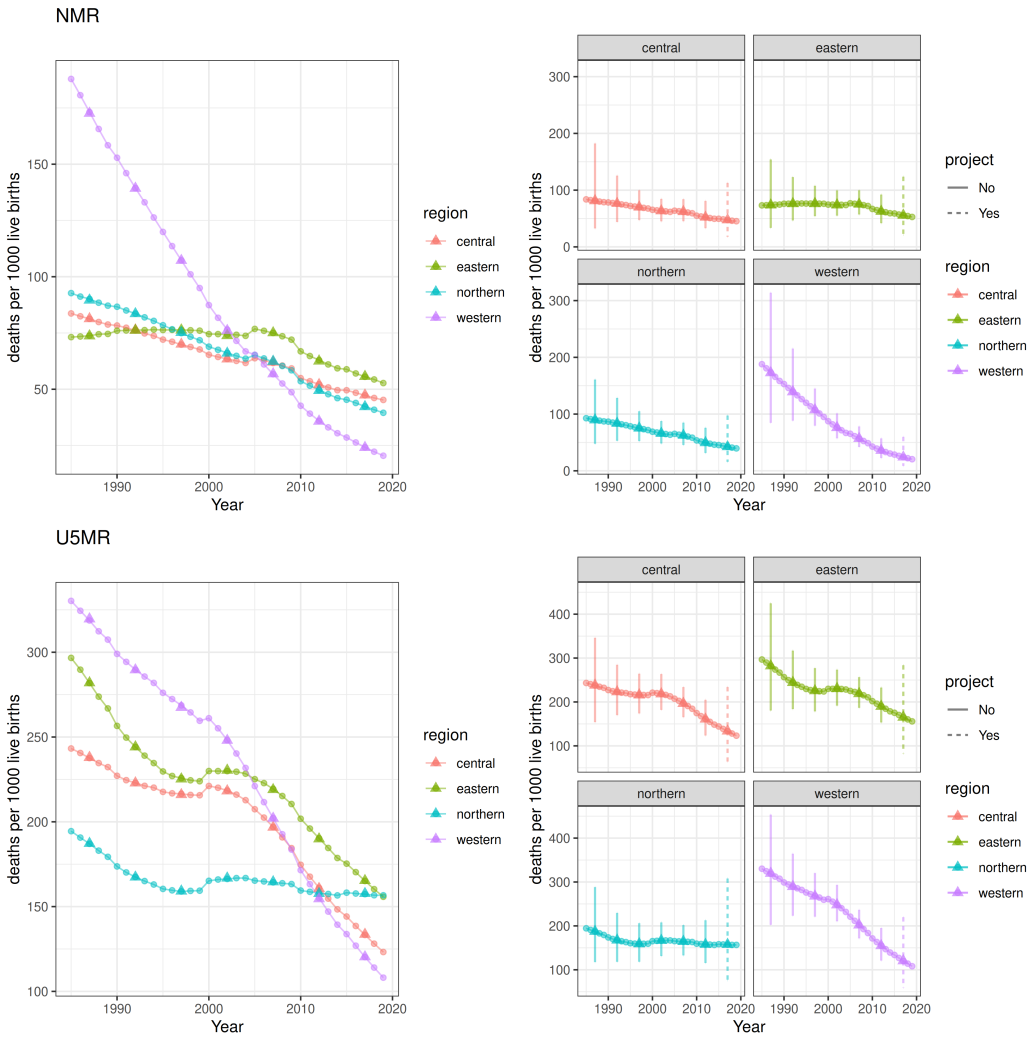


Figure 3: Smoothed direct estimates of NMR (top row) and U5MR (bottom row) on the yearly scale (dots) and 5-year period scale (triangles) in the simulated dataset. The vertical error bars correspond to 95% credible interval of the 5-year estimates. The plots on the left are from the default plot function. The plots on the right show simple customization of the default plots.

R. We document the process to read the raw DHS files and process the birth records in the supplementary materials.

For reproducibility of the examples in this paper, we start with the aggregated count data from the 2015 Malawi DHS. The pre-processed count data is available in the supplementary materials. This aggregated dataset consists of the counts of deaths occurring within each age band and the total number of person-months by cluster and year. This aggregated dataset and the full data acquisition and cleaning steps to obtain this dataset are described in the supplementary materials. The processing steps involve primarily the `getBirths()` and `getCounts()` functions and some data cleaning in region names. The supplementary materials also include workflows and results on fitting several other smoothing models on the Malawi DHS data.

Subnational spatial polygon files can usually be found on the DHS spatial data repository (The DHS Program, 2020) or the GADM database of global administrative areas (Global Administrative Areas, 2012). The admin-2 region polygon of Malawi is included in the **SUMMER** package already and can be directly loaded.

```
data(MalawiMap)
MalawiGraph = getAmat(MalawiMap, names=MalawiMap$ADM2_EN)
```

We then load the pre-processed count data and fit the cluster-level model using the `smoothCluster()` function. We consider the observations from 2007 to 2015 and project the mortality rates to 2019. To simplify results, we fit the unstratified model in this example by removing the `strata` variable from the data frame or setting it to `NA`. The supplementary materials contain additional details to fit stratified cluster-level models.

```
load("Data/DHS_counts.rda")
agg.counts$strata <- NA
head(agg.counts)
```

```
#>     v001  v025 admin2 time age  v005 died total  survey cluster strata region years Y
#> 55427  696 urban Likoma 2000  0 12778   0    3 DHS2015    696    NA Likoma  2000 0
#> 55428  696 urban Likoma 2001  0 12778   0    4 DHS2015    696    NA Likoma  2001 0
#> 55429  696 urban Likoma 2002  0 12778   0    2 DHS2015    696    NA Likoma  2002 0
#> 55430  696 urban Likoma 2003  0 12778   0    4 DHS2015    696    NA Likoma  2003 0
#> 55431  696 urban Likoma 2004  0 12778   0    3 DHS2015    696    NA Likoma  2004 0
#> 55432  696 urban Likoma 2005  0 12778   0    2 DHS2015    696    NA Likoma  2005 0
```

Sometimes, additional information is available to adjust the estimates from the surveys. For example, in countries with high prevalence of HIV, estimates of U5MR can be biased, particularly before ART treatment became widely available. Pre-treatment, HIV positive women had a high risk of dying, and such women who had given birth were therefore less likely to appear in surveys. The children of HIV positive women are also more likely to have a higher probability of dying compared to those born to HIV negative women. Hence, we expect that the U5MR is underestimated if we do not adjust for the missing women. For the two surveys in Malawi, the calculated HIV adjustment ratios as described in Walker et al. (2012) are stored in the **SUMMER** as `MalawiData$HIV.yearly`. The unstratified cluster-level model can be fitted using the `smoothCluster()` function.

```
fit.bb <- smoothCluster(data = agg.counts, Amat = MalawiGraph,
                        family = "betabinomial", year.label = 2000:2019,
                        time.model = "rw2", st.time.model = "ar1",
                  age.group = c("0", "1-11", "12-23", "24-35", "36-47", "48-59"),
                        age.n = c(1, 11, 12, 12, 12, 12),
                        age.time.group = c(1, 2, 3, 3, 3, 3),
                        pc.st.slope.u = 2, pc.st.slope.alpha = 0.1,
                        bias.adj = MalawiData$HIV.yearly,
                        bias.adj.by = c("years", "survey"),
                        survey.effect = FALSE)
```

When not specified explicitly, the space-time interaction term inherits the same temporal dependency structure defined by `time.model`. We can use different models for the interaction term by specifying the `st.time.model` argument. For example, in the model above, we can model the main temporal trends using random walks of order 2, and model the space-time interaction using the interaction of a temporal AR(1) process and an ICAR process in space. To allow each region to have more flexible temporal trends, we add region-specific random slopes to the interaction terms by specifying the priors `pc.st.slope.u` and `pc.st.slope.alpha`. These arguments specify that the probability of the absolute temporal change from the shared temporal trend (on the logit scale) over the entire time period exceeding `pc.st.slope.u` is `pc.st.slope.alpha`. The `age.group`, `age.n` and `age.time.group` specify the age groups, their corresponding length (in months), and how they are grouped when modeling the temporal trends, i.e., the $\alpha^{\star}[m]$ term defined before.

After we fit the model, we use the `getSmoothed()` function to obtain the posterior summaries of the prevalence by taking `nsim` draws from the posterior distribution. Since for the cluster-level models, the estimates may not be a linear combination of the random effect terms in the case of a composite indicator, the posterior summaries are obtained via

posterior samples. For the cluster-level model, the `getSmoothed()` function returns an object of class `SUMMERprojlist`, which includes potentially multiple projections for each stratum (`$stratified`), sampling frame (`$overall`), and aggregated over different sampling frames (`$final`) if applicable. In this example, since we fit an unstratified model with only one sampling frame, the estimates stored in `est.bb$stratified` and `est.bb$overall` are the same. In addition, by specifying `save.draws = TRUE`, the full posterior draws are stored, which can be re-used to speed up other functions that require access to posterior samples of internal model parameters.

```
est.bb <-  getSmoothed(fit.bb, nsim = 1000, save.draws = TRUE)
```

7 Visualization and model checking

In addition to the plots shown in the previous sections, the **SUMMER** package provides a collection of visualization tools to assess model fit and uncertainty in the estimates. Assessment of uncertainty is a key step in analysis as maps of point estimates can be intoxicating, but often hide huge uncertainty, which should temper initial enthusiasm. Most of the visualization options return a **ggplot2** object, which can be further customized. In this section, we use the fitted models for Malawi 2015 – 2016 DHS as an example.

The first set of visualizations are the line plots that we have shown before. Figure 4 shows subnational posterior median U5MR estimates over time for the five northern regions. We also scale the estimates to be deaths per 1,000 live births using the `per1000` argument.

```
select <- c("Chitipa", "Karonga", "Rumphi", "Mzimba")
plot(subset(est.bb$overall, region %in% select), per1000 = TRUE, year.proj = 2016)
```

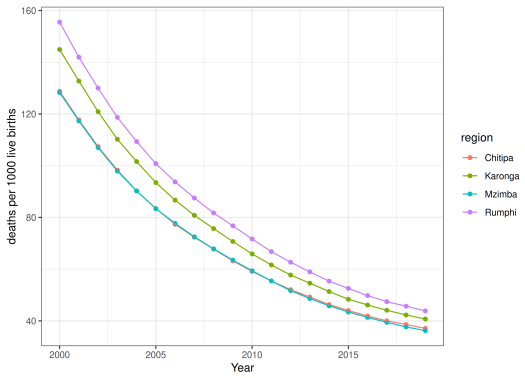


Figure 4: Subnational temporal trends of U5MR using the 2015–2016 DHS in Malawi in four regions.

By default, subnational estimates do not show the intervals to avoid many overlapping vertical bars, but they can be added back with the `plot.CI` option as illustrated in previous examples. We also compare the smoothed estimates with the pre-computed direct estimates in Figure 5, where the shrinkage in point estimates and the reduction in uncertainty intervals can be easily seen. The direct estimate computation is detailed in the supplementary materials.

```
load("Data/DHS_direct_hiv_adj.rda")
plot(subset(est.bb$overall, region %in% select), per1000 = TRUE,
     year.proj = 2016, plot.CI = TRUE,
     data.add = direct.2015.hiv, label.add = "Direct Estimates",
     option.add = list(point = "mean", lower = "lower", upper = "upper")) +
facet_wrap(~region, ncol = 4)
```

The `mapPlot()` function visualizes the estimates on a map. The estimates, `est.bb$overall`, are in the long format where estimates of each year and period are stacked. This is specified

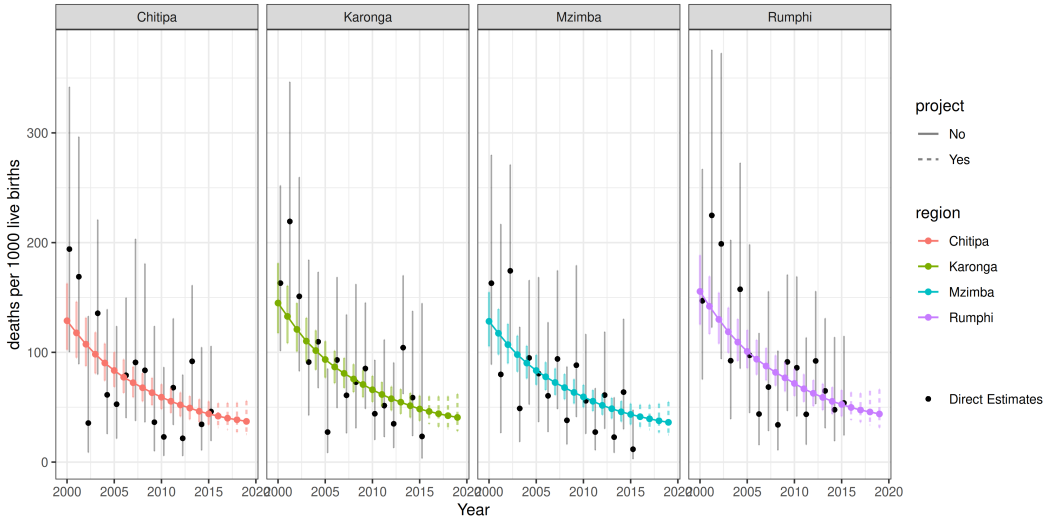


Figure 5: Comparing subnational temporal trends of U5MR under cluster-level model and direct estimates, using the 2015–2016 DHS in Malawi in four regions.

with the `is.long` argument. Figure 6 maps the changes over time. The drops in U5MR in all regions are apparent, though there is great spatial heterogeneity.

```
year.plot <- c("2007", "2010", "2013", "2016", "2019")
mapPlot(subset(est.bb$overall, years %in% year.plot),
        geo = MalawiMap, by.data = "region", by.geo = "ADM2_EN",
        is.long = TRUE, variables = "years", values = "median",
        ncol = 5, direction = -1, per1000 = TRUE, legend.label = "U5MR")
```

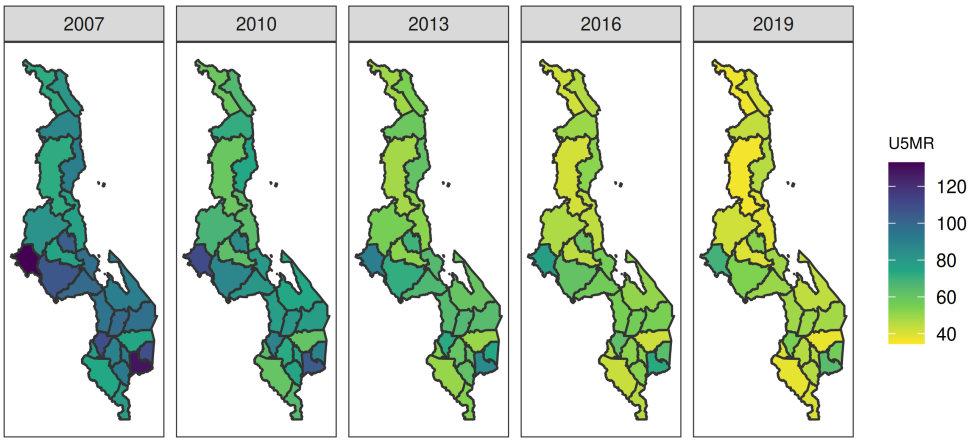


Figure 6: Spatial distribution of U5MR using the 2015–2016 DHS in Malawi over selected years.

The `hatchPlot()` function plots additional hatching lines on the map indicating the width of the uncertainty intervals. Denser hatching lines represent higher uncertainty. Usually, estimates of the early years have higher uncertainty, as shown in Figure 7. It also clearly shows the increase in uncertainty in the projections. We also note that both `mapPlot()` and `hatchPlot()` functions can be used in broader cases as they provide a general tool to visualize rectangular data on a map.

```
hatchPlot(subset(est.bb$overall, years %in% year.plot),
          geo = MalawiMap, by.data = "region", by.geo = "ADM2_EN",
          is.long = TRUE, variables = "years", values = "median",
          lower = "lower", upper = "upper", hatch = "red",
```

```
    ncol = 5, direction = -1, per1000 = TRUE, legend.label = "U5MR")
```

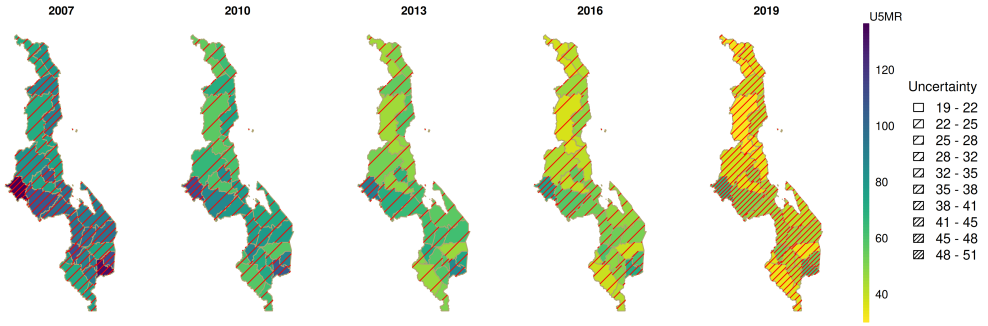


Figure 7: Subnational estimates of U5MR using the 2015–2016 DHS in Malawi over selected years, with hatching lines indicating the width of the 95% credible intervals of the estimates. Denser hatching correspond to higher uncertainty. Estimates for 2019 in the last column are from the model projection and thus have higher uncertainty.

The `ridgePlot()` function provides another visual comparison of the estimates and their associated uncertainty. Figure 8 shows one such example where the marginal posterior densities of the estimates in the selected years are plotted with regions sorted by their posterior medians in the last plotted period. The posterior densities can also be grouped with all estimates in each region plotted in the same panel using the `by.year = FALSE` argument in the `ridgePlot()` function. These plots are particularly useful to quickly identify regions with high and low estimates, while also showing the uncertainties associated with the rankings as well. The ranking of areas is often an important endeavor, since it can inform interventions in areas that are performing poorly or, more optimistically, allow areas with better outcomes to be examined to see if covariates (for example) are explaining their more positive performance.

```
ridgePlot(draws = est.bb, Amat = MalawiGraph, year.plot = year.plot,
          ncol = 5, per1000 = TRUE, order = -1, direction = -1) + xlim(c(0, 200))
```

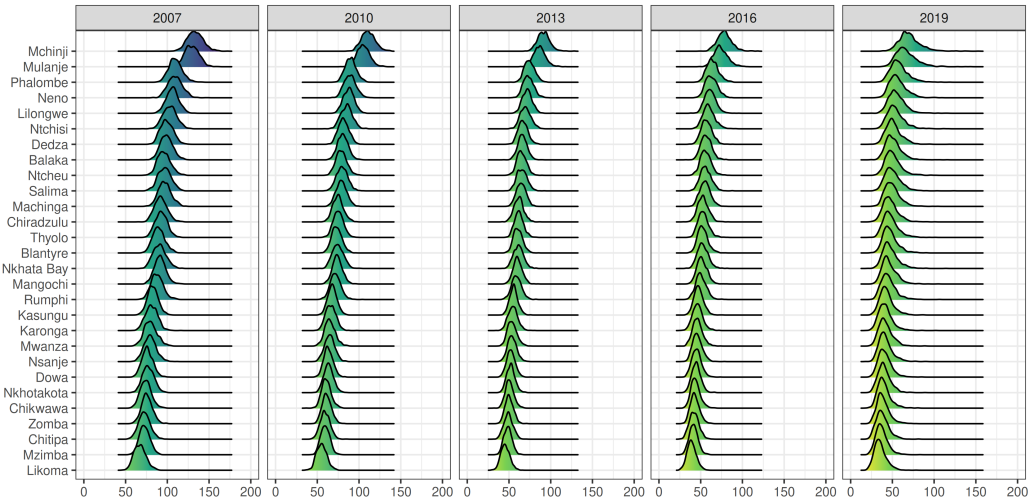


Figure 8: Posterior densities of the subnational estimates of U5MR using the 2015–2016 DHS in Malawi over selected years. Admin-2 regions are ordered by their median estimates in 2019. Estimates for 2019 in the last column are from the model projection and thus have higher uncertainty.

Finally, as models get more complicated, it becomes increasingly important to examine the estimated random effects for idiosyncratic behavior that may be evidence of model

misspecification. The SUMMER package provides tools to easily extract and plot posterior marginal distributions for each of the random effect components. We can use the `getDiag()` function to extract the posterior marginal distributions of the spatial, temporal, and space-time interaction terms from the fitted models, which can then be plotted in a similar fashion as the estimates. The space-time interaction term in the fitted model contains a sum of a region-specific linear trend and an AR(1) random effect. Thus, we need posterior samples to compute their marginal distributions. We can feed the saved posterior draws from the `est.bb` here to speed up the computation.

```
r.time <- getDiag(fit.bb, field = "time")
r.space <- getDiag(fit.bb, field = "space")
r.interact <- getDiag(fit.bb, field = "spacetime", draws = est.bb$draws)
```

The extracted posterior summaries of the random effects can then be examined and visualized. Figure 9 shows the posterior summaries of the temporal, spatial, and interaction terms in the model.

```
g.time <- ggplot(r.time, aes(x = years, y = median, ymin=lower, ymax=upper)) +
                geom_line() +
                geom_ribbon(color=NA, aes(fill = label), alpha = 0.3) +
                facet_wrap(~group, ncol = 3) +
                theme_bw() +
                ggtitle("Age-specific Temporal effects")
g.space <- mapPlot(subset(r.space, label = "Total"),
                  geo=MalawiMap, by.data="region", by.geo = "ADM2_EN",
                  direction = -1, variables="median",
                  removetab=TRUE, legend.label = "Effect") +
           ggtitle("Spatial effects")
g.interact <- ggplot(r.interact, aes(x = years, y = median, group=region)) +
                    geom_line() +  ggtitle("Interaction effects")
g.time + g.space + g.interact + plot_layout(widths = c(3, 2, 3))
```

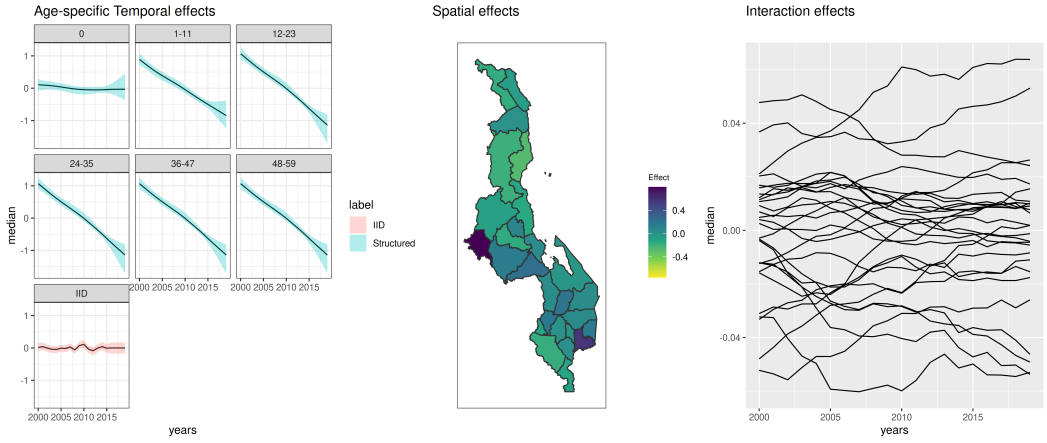


Figure 9: Posterior medians for the random effect terms in the cluster-level model using Malawi 2015–2016 DHS. Left: posterior medians and 95% credible intervals of the age-specific temporal effects and the IID temporal shocks. Middle: posterior medians of the spatial effects. Right: posterior medians of the space-time interaction effects.

8 Discussion

The present paper aims to provide a general overview of the R package SUMMER for space-time smoothing of demographic and health indicators. The particular focus of this

paper is on mortality estimation and the demonstration of the workflow for practitioners to fit flexible Bayesian smoothing models with DHS data. The implementation using **INLA** allows fast computation of these smoothing models. The Fay-Herriot estimates can usually be fit within seconds to minutes depending on the number of regions and time period. The cluster-level model may require longer computation time, especially with surveys containing many samples. We leave the fitting of more time-consuming models in the supplementary materials.

The **SUMMER** package is in constant development. This paper introduces the core functionalities of the package. There are many more functionalities to tackle application-specific issues, such as different age-time interactions, aggregation over urban/rural strata, benchmarking to external national estimates, etc. We are also expanding the package to include more options for the traditional SAE models with fast implementations built on our current computational framework. Recent extensions built on **SUMMER** functionalities include the recent addition of SAE methods in the **survey** package (Lumley, 2024) and the **surveyPrev** package (Dong et al., 2024) which provides a general pipeline to download, process, and map a broad variety of DHS indicators.

In the future, we have several plans to improve the functionality of **SUMMER**. In the cluster-level model, we would like to allow different overdispersion parameters for different age groups. We plan to incorporate methods for child mortality estimation using summary birth history data (Hill et al., 2015; Wilson and Wakefield, 2021) in which women provide only information on the number of children born, and number died, without the dates of these events. We also expect to extend the core functionalities to model other demographic and health indicators such as fertility. In our examples in this paper, we did not include covariates. In both the area-level and the cluster-level models, covariates can be included; see Wakefield et al. (2020) for an example in the context of HIV prevalence mapping in Malawi. Finally, in the long term, we would like to incorporate continuous spatial models as well.

References

L. Alkema and J. R. New. Global estimation of child mortality using a Bayesian B-spline bias-reduction model. *The Annals of Applied Statistics*, 8:2122–2149, 2014. [p89]

J. Besag, J. York, and A. Mollié. Bayesian image restoration, with two applications in spatial statistics (with discussion). *Annals of the Institute of Statistical Mathematics*, 43:1–59, 1991. [p90]

D. A. Binder. On the variances of asymptotically normal estimators from complex surveys. *International Statistical Review*, 51:279–292, 1983. [p90]

H. J. Boonstra. *hbsae: Hierarchical Bayesian small area estimation*, 2012. R package version 1.0. [p88]

Q. Dong, Z. R. Li, Y. Wu, A. Boskovic, and J. Wakefield. *surveyPrev: Mapping the prevalence of binary indicators using survey data in small areas*, 2024. URL `https://CRAN.R-project.org/package=surveyPrev`. R package version 1.0.0. [p104]

T. Q. Dong and J. Wakefield. Modeling and presentation of vaccination coverage estimates using data from household surveys. *Vaccine*, 39(18):2584–2594, 2021. [p89, 91]

R. E. Fay and R. A. Herriot. Estimates of income for small places: An application of James–Stein procedure to census data. *Journal of the American Statistical Association*, 74:269–277, 1979. [p88, 89, 90]

G.-A. Fuglstad, Z. R. Li, and J. Wakefield. The two cultures for prevalence mapping: Small area estimation and spatial statistics. *arXiv preprint arXiv:2110.09576*, 2021. [p89, 93]

Global Administrative Areas. Gadm database of global administrative areas, 2012. [p98]

J. Hájek. Discussion of, "An essay on the logical foundations of survey sampling, part I", by D. Basu. In V. Godambe and D. Sprott, editors, *Foundations of Statistical Inference*. Holt, Rinehart and Winston, Toronto, 1971. [p89]

K. Hill, E. Brady, L. Zimmerman, L. Montana, R. Silva, and A. Amouzou. Monitoring change in child mortality through household surveys. *PloS One*, 10:e0137713, 2015. [p104]

D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47:663–685, 1952. [p89]

L. Knorr-Held. Bayesian modelling of inseparable space-time variation in disease risk. *Statistics in Medicine*, 19:2555–2567, 2000. [p90]

Z. R. Li, Y. Hsiao, J. Godwin, B. D. Martin, J. Wakefield, and S. J. Clark. Changes in the spatial distribution of the under five mortality rate: Small-area analysis of 122 DHS surveys in 262 subregions of 35 countries in Africa. *PLoS One*, 2019. Published January 22, 2019. [p89, 91, 93, 97]

T. Lumley. Analysis of complex survey samples. *Journal of Statistical Software*, 9(1):1–19, 2004. [p95]

T. Lumley. *survey: Analysis of complex survey samples*, 2024. R package version 4.4. [p104]

S. MacFeely. Measuring the Sustainable Development Goal indicators: An unprecedented statistical challenge. *Journal of Official Statistics*, 36(2):361–378, 2020. [p88]

L. D. Mercer, J. Wakefield, A. Pantazis, A. M. Lutambi, H. Masanja, and S. Clark. Small area estimation of childhood mortality in the absence of vital registration. *Annals of Applied Statistics*, 9:1889–1905, 2015. [p89, 90, 91]

I. Molina and Y. Marhuenda. sae: An R package for small area estimation. *The R Journal*, 7: 81–98, 2015. [p88]

J. Paige, G.-A. Fuglstad, A. Riebler, and J. Wakefield. Model-based approaches to analysing spatial data from complex surveys. *Journal of Survey Statistics and Methodology*, 2020. To appear. [p92]

J. Pedersen and J. Liu. Child mortality estimation: Appropriate time periods for child mortality estimates from full birth histories. *PLoS Med*, 9(8):e1001289, 2012. [p91]

T. L. Pedersen. *patchwork: The composer of plots*, 2019. URL `https://CRAN.R-project.org/package=patchwork`. R package version 1.0.0. [p93]

N. Permatasari and A. Ubaidillah. msae: An R package of multivariate Fay Herriot models for small area estimation. *The R Journal*, 2021. [p88]

J. N. Rao and I. Molina. *Small Area Estimation, Second Edition*. John Wiley, New York, 2015. [p88, 89]

A. Riebler, S. H. Sørbye, D. Simpson, and H. Rue. An intuitive Bayesian spatial model for disease mapping that accounts for scaling. *Statistical Methods in Medical Research*, 25: 1145–1165, 2016. [p90]

H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Application*. Chapman and Hall/CRC Press, Boca Raton, 2005. [p90]

H. Rue, S. Martino, and N. Chopin. Approximate Bayesian inference for latent Gaussian models using integrated nested Laplace approximations (with discussion). *Journal of the Royal Statistical Society, Series B*, 71:319–392, 2009. [p93]

B.-S. Schlüter and B. Masquelier. Space-time smoothing of mortality estimates in children aged 5-14 in Sub-Saharan Africa. *PloS One*, 16(1):e0245596, 2021. [p89]

T. Schoch. *rsae: Robust small area estimation*, 2014. R package version 0.1-5. [p88]

C. Shi. *BayesSAE: Bayesian analysis of small area estimation*, 2018. R package version 1.0-2. [p88]

The DHS Program. Spatial data repository, 2020. URL https://spatialdata.dhsprogram.com/home/. [p90, 98]

United Nations Inter-agency Group for Child Mortality Estimation. Subnational under-five and neonatal mortality estimates, 2000–2021 estimates developed by the United Nations Inter-agency Group for Child Mortality Estimation, 2023. United Nations Children's Fund, New York. [p89]

J. Wakefield, G.-A. Fuglstad, A. Riebler, J. Godwin, K. Wilson, and S. Clark. Estimating under five mortality in space and time in a developing world context. *Statistical Methods in Medical Research*, 28:2614–2634, 2019. [p92]

J. Wakefield, T. Okonek, and J. Pedersen. Small area estimation of health outcomes. *International Statistical Review*, 2020. To appear. [p89, 104]

N. Walker, K. Hill, and F. Zhao. Child mortality estimation: Methods used to adjust for bias due to AIDS in estimating trends in under-five mortality. *PLoS Med*, 9:e1001298, 2012. [p92, 99]

O. Watson and J. Eaton. *rdhs: API client and dataset management for the Demographic and Health Survey (DHS) data*, 2019. URL https://CRAN.R-project.org/package=rdhs. R package version 0.6.3. [p97]

H. Wickham. *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p93]

K. Wilson and J. Wakefield. Child mortality estimation incorporating summary birth history data. *Biometrics*, 77(4):1456–1466, 2021. [p104]

K. Wolter. *Introduction to Variance Estimation*. Springer Science & Business Media, 2007. [p89]

Y. Wu, Z. R. Li, B. K. Mayala, H. Wang, P. Gao, J. Paige, G.-A. Fuglstad, C. Moe, J. Godwin, R. E. Donohue, T. N. Croft, and J. Wakefield. Spatial modeling for subnational administrative level 2 small-area estimation. Technical report, ICF, Rockville, Maryland, USA, 2021. [p92, 93]

Zehang Richard Li
University of California, Santa Cruz
Santa Cruz CA, USA
ORCiD: 0000-0002-9079-593X
lizehang@ucsc.edu

Bryan D Martin
University of Washington
Seattle, WA, USA
bmartin6@uw.edu

Tracy Qi Dong
Fred Hutchinson Cancer Research Center
Seattle, WA, USA
ORCiD: 0000-0001-6769-608X
qdong@fredhutch.org

Geir-Arne Fuglstad
Norwegian University of Science and Technology

Trondheim, Norway
ORCiD: 0000-0003-4995-2152
geir-arne.fuglstad@ntnu.no

Jessica Godwin
University of Washington
Seattle, WA, USA
jlg0003@uw.edu

John Paige
Norwegian Computing Center
Oslo, Norway
ORCiD: 0000-0003-2683-2596
paigejo@gmail.com

Andrea Riebler
Norwegian University of Science and Technology
Trondheim, Norway
andrea.riebler@ntnu.no

Samuel J Clark
The Ohio State University
Columbus, OH, USA
ORCiD: 0000-0002-4929-6231
work@samclark.net

Jon Wakefield
University of Washington
Seattle, WA, USA
jonno@uw.edu

latrend: A Framework for Clustering Longitudinal Data

by , Steffen Pauws, and Edwin van den Heuvel

Abstract Clustering of longitudinal data is used to explore common trends among subjects over time. In this paper, we focus on cases where the sole repeated measurement of interest is numeric. Various R packages have been introduced throughout the years for identifying clusters of longitudinal patterns, summarizing the variability in trajectories between subjects in terms of one or more trends. We introduce the R package latrend as a framework for the unified application of methods for longitudinal clustering, enabling comparisons between methods with minimal coding. The package also serves as an interface to commonly used packages for clustering longitudinal data, including dtwclust, flexmix, kml, lcmm, mclust, mixAK, and mixtools. This enables researchers to easily compare different approaches, implementations, and method specifications. Furthermore, researchers can build upon the standard tools provided by the framework to quickly implement new cluster methods, enabling rapid prototyping.

1 Introduction

In this work, we consider the case where subjects are repeatedly measured on the same variable over a period of time. This type of data is referred to as longitudinal data. In this paper, we focus on the repeated measurement of a single numeric variable, although in other applications, longitudinal measurements may be ordinal or even categorical (e.g., sequence analysis), and comprise multiple variables. No two subjects are identical, and therefore observations made across subjects may develop differently over time. Usually, longitudinal datasets are represented by a single general trend, i.e., an average representative trajectory indicating the expected change and variability over time. However, there may be structural deviations from the trend caused by observed and unobserved factors, or the distribution of random deviations is difficult to model parametrically.

Clustering longitudinal data is a practical approach for exploring or representing the variability between subjects in more detail (Hamaker, 2012). Here, the variability is summarized in terms of a manageable number of common trends, which are identified in an unsupervised manner from the data using a cluster algorithm. The approach is especially useful for exploring datasets involving a large number of trajectories, where a visual inspection of the trajectories would be impractical. In essence, the data are assumed to comprise several groups, each with a different longitudinal data generating mechanism. It differs from cross-sectional clustering due to the need to account for the dependency between observations within subjects, and the possible temporal correlation in the repeated measurements.

The exploration of subgroups in longitudinal studies is of interest in many domains. Examples include recidivism behavior in criminology, the development of adolescent antisocial behavior or substance use in psychology, and medication adherence in medicine. An example application that we will demonstrate further in this paper is the exploration of the different ways in which patients with sleep apnea adhere to positive airway pressure (PAP) therapy over time. Here, therapy adherence is measured in terms of the number of hours of sleep during which the therapy is used, recorded daily. Patients exhibit different levels of adherence to the therapy, depending on many factors such as their sleep schedule, motivation, self-efficacy, and the perceived importance of therapy (Cayanan et al., 2019). Moreover, patients may exhibit a different level of change over time, depending on their initial usage and their ability to adjust to the therapy. To account for the many possibly unobserved factors involved, researchers have used longitudinal clustering to summarize the between-subject variability in terms of longitudinal patterns of therapy adherence (Babbin et al., 2015; Den Teuling et al., 2021a; Yi et al., 2022).

A number of packages have been created in R (R Core Team, 2024) that can be used for clustering longitudinal data. However, for researchers analyzing a novel case study, choosing the best method or implementation is not straightforward due to the inherent exploratory nature of such an analysis. Considering that each of these packages has been created to fulfill a gap in the capabilities of other existing implementations or approaches, there is value in comparing the results for new case studies at hand. In any case, the evaluation of different approaches across packages is an activity of considerable effort, as the methods, inputs, estimation procedure, and cluster representations differ greatly between packages.

The aim of the **latrend** package is to facilitate the exploration of heterogeneity in a longitudinal dataset on a numeric variable of interest, through a variety of cluster methods from various fields of research in a standardized manner. The package provides a unifying framework, enabling users to specify, estimate, select, compare, and evaluate any supported longitudinal cluster method in an easy and consistent way, with minimal coding. Most importantly, users can easily compare results between different approaches, or run a simulation study. A second aim of **latrend** is to enable users to extend the framework with new methods or add support for other existing methods. The **latrend** package is available from the Comprehensive R Archive Network (CRAN) at (`https://CRAN.R-project.org/package=latrend`) and on GitHub at (`https://github.com/niekdt/latrend`).

Currently, a total of 18 methods for longitudinal clustering are supported. To provide support for such a variety of approaches, the **latrend** package interfaces with an extensive set of packages that provide methods that are applicable for clustering longitudinal data, including **akmedoids** (Adepeju et al., 2020), **crimCV** (Nielsen, 2023), **dtwclust** (Sardá-Espinosa, 2019), **flexmix** (Grün and Leisch, 2008), **funFEM** (Bouveyron, 2021), **kml** (Genolini et al., 2015), **lcmm** (Proust-Lima et al., 2017), **mclust** (Scrucca et al., 2016), **mixAK** (Komárek, 2009), and **mixtools** (Benaglia et al., 2009). In this way, we build upon the cluster packages created by the R community. Support has also been added for MixTVEM; a mixture model proposed and implemented as an R script by Dziak et al. (2015).

To the best of our knowledge, such a comprehensive package does not yet exist in the context of clustering longitudinal data. The **latrend** package has similar aspirations as the **flexmix** package (Grün and Leisch, 2008), which also provides an extensible framework for (multilevel) clustering. However, the scope of our package is purposefully broader, to facilitate users to apply approaches from various fields of research. Our framework is agnostic to the specification, estimation, and representation used by the methods.

The paper is organized as follows. A short overview of different approaches to clustering longitudinal data is given in Section 2. In Section 3, the design principles and high-level structure of the framework are described. The usage of the package is demonstrated in Section 4. Section 5 describes three ways in which users can implement their own cluster methods. Lastly, a summary and future steps are presented in Section 6.

2 Methods

We will briefly describe common general approaches to clustering longitudinal data, and the main strengths of these approaches. For brevity, we do not go into the specifics of any of the packages. We refer to the accompanying articles of these packages for further details. Starting with the aspects that all approaches have in common, let the repeated observations of the trajectory from subject i be denoted by

$$\mathbf{y}_i = (y_{i1}, y_{i2}, ..., y_{iJ_i}),$$

where y_{ij} is a numerical value of some variable of interest, t_{ij} is the measurement time, and J_i is the number of observations of trajectory $\mathbf{y}_i$ for subject i.

Regardless of the approach, any method for clustering longitudinal data approximates the dataset heterogeneity in terms of a set of K clusters, with each cluster representing

a proportion π_k of the population, with $\pi_k > 0$ and $\sum_{k=1}^{K} \pi_k = 1$. The clusters may be discovered by identifying groupings of similar subjects, based on their trajectory. Typically, a cluster method is estimated for a given number of clusters, specified by the user. By applying a cluster method for a different number of clusters, the most appropriate number of clusters can then be determined for the respective data.

Subjects are generally assumed to belong to a single cluster. Therefore, many cluster methods partition the subjects into k mutually exclusive sets $I_1, I_2, ..., I_K$, where I_k denotes the set of subjects to belong to cluster k, with $\bigcup_{k=1}^{K} I_k = I$. Depending on the application, it may be desirable to identify a representation for each cluster, also referred to as the cluster center, which provides a summary of the cluster. This representation may be obtained from the averaged representation of all the subjects assigned to the respective cluster, by designating a representative subject, or through the cluster representation defined by the method, if applicable.

Other cluster methods allow for overlapping clusters, commonly referred to as soft or fuzzy clustering. Here, subjects may belong to multiple clusters, with a certain degree or weight to which subjects belong to each cluster. In the case of model-based clustering (McNicholas and Murphy, 2010), the clusters are represented by a mixture of statistical models, for which cluster membership is expressed as a probability. In applications where each subject is assumed to belong to one cluster, subjects are typically assigned to the cluster with the highest subject-specific posterior probability, referred to as modal assignment.

2.1 Cross-sectional clustering

In a cross-sectional cluster approach, also referred to as a raw-data-based approach (Liao, 2005), the different observation moments are treated as separate features for a standard cluster algorithm, i.e., as if we are conducting a cross-sectional cluster analysis. In standard cluster algorithms such as k-means, the features are assumed to be independent, although this is generally not a strict requirement. The temporal independence assumption made in this approach yields a non-parametric representation of the trajectories. This makes it a useful approach for an exploratory analysis without any prior assumptions on the shape of the trajectories. The main limitation of this approach is that observations must be aligned between trajectories, i.e., measured at the same respective moments in time. Consequently, missing observations should be imputed.

An example of a cross-sectional approach is longitudinal k-means (KmL). KmL applies the k-means cluster algorithm directly to the observations. The cluster trajectories are determined by the averaged observations of trajectories assigned to the respective cluster. The method is implemented in the **kml** package by Genolini et al. (2015).

A model-based cross-sectional approach is seen in longitudinal latent profile analysis (LLPA), otherwise known as longitudinal latent class analysis (Muthén, 2004). Here, Gaussian mixture modeling is used to describe each moment in time as a normally distributed random variable. A dataset with trajectories each comprising J observations is thus described by J random variables, each modeling the response distribution at a different moment in time. Gaussian mixture models can be estimated using, for example, the **mclust** package by Scrucca et al. (2016). In the simplest case, the J observations are modeled as being independent and the variance is shared between clusters, but by relaxing constraints on the covariance matrix, temporal correlations and different cluster shapes can be accounted for.

2.2 Distance-based clustering

Distance-based cluster algorithms operate on the pairwise distance between trajectories. These methods take a distance matrix of pairwise trajectory distances as input, where the choice of the distance metric, i.e., the dissimilarity measure, is left to the user. Examples of cluster algorithms that use this approach include k-medoids and agglomerative hierarchical clustering.

Given the trajectories of subject a and b, the distance metric is denoted by $d(\mathbf{y}_a, \mathbf{y}_b)$. As an example, the Euclidean distance

$$d(\mathbf{y}_a, \mathbf{y}_b) = \sqrt{\sum_j (y_{bj} - y_{aj})^2}.$$

may be used as the distance metric. Cross-sectional clustering is a special case of distance-based clustering where a raw-data distance metric is used.

The approach is commonly used for time series clustering[1], and the list of available distance metrics that have been proposed over the past decades is extensive (Aghabozorgi et al., 2015). A distance function can be specified to account for one or more temporal aspects of interest, e.g., mean level, changes over time, variability, autocorrelation, spectral components, and entropy. Many dissimilarity metrics are implemented in the **dtwclust** package (Sardá-Espinosa, 2019).

2.3 Regression-based clustering

In regression-based clustering, the longitudinal dataset is modeled by a regression model comprising a mixture of submodels (de la Cruz-Mesía et al., 2008). It is also referred to as latent-class trajectory modeling. This approach comprises a versatile class of (semi-)parametric methods. Most importantly, the shape of the trajectories can be represented using a parametric model, requiring fewer parameters compared to a non-parametric approach. Measurements can be taken at different times between subjects, and covariates can be accounted for. Moreover, users can incorporate assumptions into the modeling of the trajectories and clusters, such as the distribution of the response variable, the within-cluster variability, and heteroskedasticity.

A straightforward example of regression-based clustering involves modeling the population as a mixture of cluster trajectory models. This is referred to as group-based trajectory modeling (GBTM) or latent-class growth analysis (LCGA). It is essentially a mixture of linear regression models, with

$$y_{ij} = \mathbf{x}_{ij}\mathbf{fi}_k + \varepsilon_{ijk} \quad \text{for } i \in I_k, \tag{1}$$

where $\mathbf{x}_{ij}$ is the $N \times B$ design matrix of B covariates, $\mathbf{fi}_k$ are the B group-specific coefficients, and ε_{ijk} is the normally distributed residual error with zero mean and constant variance σ_k^2 which may be specified to differ between clusters. The design matrix contains covariates of time, enabling the model to describe the change in response over time. External covariates can be included to further explain the dependent variable. The expected values of a trajectory, assuming the trajectory belongs to cluster k, is given by

$$E(y_{ij}|C_i = k) = \mathbf{x}_{ij}\mathbf{fi}_k. \tag{2}$$

GBTM is available, for example, in the packages **lcmm** (Proust-Lima et al., 2017) and **crimCV** (Nielsen, 2023).

A popular form of regression-based clustering that does consider within-cluster variability is growth mixture modeling (GMM) (Muthén, 2004), which represents a mixture of multilevel models. Here, the within-cluster variability is modeled by allowing for subject-specific deviations from the cluster center, e.g., a deviation in the intercept. Using a linear mixed modeling approach, the trajectories for cluster k are given by

$$y_{ij} = \mathbf{x}_{ij}\mathbf{fi}_k + \mathbf{z}_{ij}\mathbf{u}_{ki} + \varepsilon_{ijk} \quad \text{for } i \in I_k. \tag{3}$$

Here, $\mathbf{z}_{ij}$ is the $N \times U$ design matrix for the U random effects, and $\mathbf{u}_{ki}$ are the subject-specific random coefficients for cluster k. The random effects are assumed to be normally distributed with mean zero and variance-covariance matrix Σ_k. The expected values of a trajectory,

[1] Clustering longitudinal data can be regarded as a special case of time series clustering where the time series have a common starting point.

assuming the trajectory belongs to cluster k, is given by

$$E(y_{ij}|C_i = k, \mathbf{u}_i) = \mathbf{x}_{ij}\mathbf{fi}_k + \mathbf{z}_{ij}\mathbf{u}_{ki}. \tag{4}$$

GMM is available in packages such as **lcmm** (Proust-Lima et al., 2017), **mixtools** (Benaglia et al., 2009), and **mixAK** (Komárek, 2009).

2.4 Feature-based clustering

In a feature-based approach, each trajectory is independently represented by a set of temporal characteristics (i.e., features, coefficients), for example, the mean, variability, and change over time (Liao, 2005). The trajectories are then clustered based on the features or coefficients using a cross-sectional cluster algorithm. This can be regarded as a special case of distance-based clustering, but with a domain-tailored distance function. This approach has the advantage of allowing users to easily combine arbitrary features of interest. The approach is used, for example, by the anchored k-medoids algorithm provided by the **akmedoids** package (Adepeju et al., 2020). Here, the trajectories are represented using linear regression models, and are clustered based on the model coefficients.

Compared to the rather time-intensive regression-based clustering approach, the trajectory models only need to be estimated once. A disadvantage compared to regression-based clustering is that the reliability of the trajectory coefficients depends on the available data per trajectory. This approach therefore generally requires a greater number of observations per subject to yield similar results.

2.5 Identifying the number of clusters

Due to the exploratory nature of clustering, the number of clusters is typically not known in advance. Moreover, most of the cluster methods require the user to specify the number of clusters. The preferred number of clusters for the respective method can be determined by estimating the method for an increasing number of clusters, followed by comparing the solutions by means of an evaluation metric. In such a comparison for a particular method, the interpretation of the metric is consistent across the solutions, as they all originate from the same method specification.

Many metrics are available, although they may not be defined for each type of method. For example, in distance-based methods, the solutions are typically evaluated in terms of the separation between clusters. Cluster separation is measured by the distance between trajectories or cluster trajectories, e.g., using the average Silhouette width (ASW) (Rousseeuw, 1987) or the Dunn index (Arbelaitz et al., 2013). In contrast, a regression-based approach typically has no notion of the distance between trajectories, but instead measures the likelihood of the overall regression model on the given data, enabling the use of likelihood-based evaluation such as the Bayesian information criterion (BIC), Akaike information criterion (AIC), or likelihood ratio test (der Nest et al., 2020). Specific to cluster regression methods where the longitudinal observations are modeled at the subject level, assessing the solution in terms of the residual errors of the trajectories may be of interest. Examples of such metrics include the mean absolute error (MAE) and root mean squared error (RMSE). For probabilistic assignments these metrics may be weighted by the posterior probability of the trajectories, denoted as WMAE and WRMSE, respectively.

Overall, the preferred metric depends on the type of method under consideration and the case study domain. In the case of evaluation between different types of methods, a metric should be selected which is defined for both types of method, which may rule out many of the options. Users are advised to follow recommendations from literature for the respective method. Moreover, it is advisable to use the evaluation metric merely as guidance in identifying the preferred solution, as a trade-off between the number of clusters and the interpretability of the solution. Lastly, it is worthwhile to factor in domain knowledge into the selection of cluster solutions (Nagin et al., 2018).

2.6 Comparing methods

The approaches may yield considerably different results, arising from fundamental differences in the temporal representation and similarity criterion of the methods. Moreover, some approaches are more applicable to certain measurement moments, sample sizes, trajectory shapes and cluster sizes than others. To guide users towards an initial choice for a suitable approach, we have listed some of the general strengths and limitations of the different approaches in Table 1. Note that even for methods of the same type of approach, results may differ depending on how the trajectories are represented, trajectory similarity is measured, or how clusters are formed. Considering that the most suitable approach or method is typically not known in advance, it is advisable to evaluate and compare the solutions between methods to identify the most suitable method for the respective case study. The resulting solutions can then be compared using an external evaluation metric.

A useful starting point in comparing the preferred solutions between methods is to evaluate the similarity between the cluster partitions. After all, if both candidate methods find a similar cluster partition, this would indicate that both methods find the same grouping despite representational differences. In contrast, if the cluster partitions are dissimilar, it may suggest that either a hybrid approach could be of interest, or that one method is preferred over the other.

The similarity between cluster partitions of two methods can be assessed using partition similarity metrics such as the adjusted Rand index (ARI) (Hubert and Arabie, 1985), variance of information, or the split-join index. These metrics are applicable to any method and are even applicable when the solutions have a mismatching number of clusters. In some case studies, a ground truth may be available in the form of a reference cluster partition. Partition similarity metrics such as the ARI may then be used to identify the solution that most closely resembles the ground truth. Alternatively, one may obtain a partial ground truth by manually annotating a subset of the trajectories based on domain knowledge.

Solutions may be compared further by assessing the compactness of the clusters or the separation between clusters on a common distance metric, for example using the average Silhouette width or the Dunn index. This is useful to identify the method that is best at identifying distinct subgroups.

Table 1: Summary of the general strengths and limitations of the different approaches to longitudinal clustering.

Approach	Strengths	Limitations
Cross-sectional	• Suitable for initial exploration due to no assumptions on the shape of the cluster trajectories • Low sample size requirement • Very fast to estimate	• Requires time-aligned trajectories of equal length • Requires complete data • Does not account for the temporal relation of observations
Distance-based	• Flexible in the choice of distance metric(s) • Trajectory distance matrix only needs to be computed once • Fast to estimate	• Distance matrix computation is not practical for a large number of trajectories • Pairwise comparison of trajectories is more sensitive to noise • Many distance metrics require time-aligned trajectories

Approach	Strengths	Limitations
Regression-based	• Low sample size requirements due to inclusion of parametric assumptions (Martin and von Oertzen, 2015) • Can handle missing data • Can handle trajectories of unequal length and variable time • Can account for covariates • Relatively robust to trajectories that do not fit the representation	• May be challenging to estimate (convergence problems) (Den Teuling et al., 2021b) • Computationally intensive to estimate
Feature-based	• Temporal features only needs to be computed once • Very fast to estimate • Fast alternative to regression-based approach given a sufficiently large sample size (Den Teuling et al., 2021b)	• Sensitive to trajectories that do not fit the representation • Trajectory-independent feature estimation is more sensitive to observational outliers

3 Software design

We begin by providing a high-level description of the framework, outlining the main functionality of the classes. A step-by-step demonstration of the framework is given in the next section. The software is built on an object-oriented paradigm using the S4 system, available in the **methods** package (R Core Team, 2024). The framework is designed to provide a standardized way of specifying, estimating, and evaluating different longitudinal cluster methods. This is achieved by defining two interfaces: the `lcMethod` interface is used for defining the specification and estimation logic of a method. The `lcModel` interface represents the result of an estimated method. Using these two interfaces, we can then define method-agnostic estimation procedures for applying a specified method to a given dataset, yielding a method result. This estimation procedure is implemented by the `latrend()` function. For example, users can specify a growth mixture model (GMM) through a `lcMethodLcmmGMM` object, specifying the GMM and the estimation settings. The resulting estimated GMM is represented by a `lcModelLcmmGMM` object.

A key advantage of having stand-alone estimation procedures is that it ensures all methods take the same data format as input, and allows for more procedures to be implemented which automatically support all implemented methods. There are additional procedures implemented in the package, including repeated estimation via `latrendRep()`, batch estimation via `latrendBatch()`, and standard non-parametric bootstrap sampling via `latrendBoot()`.

3.1 Dataset input

We have selected the `data.frame` in long format as the preferred representation for longitudinal datasets. Here, each row represents an observation for a trajectory at a given time, possibly for multiple covariates. The trajectory and time of an observation are indicated in separate columns. This format can represent irregularly timed measurements, a variable number of observations per trajectory, and an arbitrary number of covariates of different types. Since not all datasets are readily available in this format, the `latrend()` estimation procedures handle data input by calling the generic `transformLatrendData()` function. Currently, this transformation is only defined for `matrix` input. Users can implement the method to add support for other longitudinal data types.

3.2 The `lcMethod` class

The `lcMethod` class has two purposes. The first purpose is to record the method specification, defined by the method parameters and other settings, referred to as the method arguments.

The second purpose is to provide the logic for estimating the method for the specified arguments and given data. `lcMethod` objects are immutable. Users only interact with a `lcMethod` object for retrieving method arguments, or for creating a new specification with modified arguments. This functionality is provided by the base `lcMethod` class.

The base class also stores the method arguments in a list, inside the `arguments` slot. The method arguments can be of any type. The names of subclasses are prefixed by "*lcMethod*". Subclasses can validate the model arguments against the data by overriding the `validate()` function. Due to the specific internal structure of a `lcMethod` object, constructors are defined for creating `lcMethod` objects of a specific class for a given set of arguments. In `lcMethod` implementations that are a wrapper around an existing cluster package function, the method arguments are simply passed to the package function. The required arguments and their default values are obtained from the formal function arguments of the package function at runtime.

The evaluation of the method arguments is delayed until the method estimation process. This enables a `lcMethod` object to be printed in an easily readable way, where the original argument expressions or calls are shown, instead of the evaluation result. This is useful when an argument takes on a function or complex data structure, and it reduces the memory footprint when a large set of method permutations is generated and serialized, such as in a simulation study.

The method estimation process is implemented through six generic functions: `prepareData()`, `compose()`, `validate()`, `preFit()`, `fit()`, and `postFit()`. The purpose of each step is explained in Section 5. There are several advantages to this design. Firstly, the structure enables the method estimation process to be checked at each step. Secondly, splitting the estimation logic into processing steps encourages shorter functions with clearer functionality, resulting in more readable code. Thirdly, the steps enable optimizations in the case of repeated method estimation, for which the `prepareData()` function only needs to be called once. Lastly, in case of an update to the `lcModel` post-processing step, the `postFit()` function can be applied to previously obtained `lcModel` objects.

Supported methods

An overview of the currently available methods that can be specified is given in Table 2. The `lcMethodGCKM` class implements a feature-based approach, based on representing the trajectories through a linear mixed model specified in the **lme4** package (Bates et al., 2015).

Table 2: The list of currently supported methods for clustering longitudinal data, in alphabetical order. The methods in the bottom row represent generic approaches which can be adapted. Class names are prefixed by "lcMethod".

Class (`lcMethod`)	Method	Package
`Akmedoids`	Anchored *k*-medoids	**akmedoids** (Adepeju et al., 2020)
`CrimCV`	Group-based trajectory modeling of count data	**crimCV** (Nielsen, 2023)
`Dtwclust`	Dynamic time warping	**dtwclust** (Sardá-Espinosa, 2019)
`Flexmix`	Interface to FlexMix framework	**flexmix** (Grün and Leisch, 2008)
`FlexmixGBTM`	Group-based trajectory modeling	**flexmix** (Grün and Leisch, 2008)
`FunFEM`	funFEM	**funFEM** (Bouveyron, 2021)
`GCKM`	Feature-based clustering using growth curve modeling and *k*-means	**lme4** (Bates et al., 2015)
`KML`	longitudinal *k*-means	**kml** (Genolini et al., 2015)

Class (`lcMethod`)	Method	Package
`LcmmGBTM`	Group-based trajectory modeling	**lcmm** (Proust-Lima et al., 2017)
`LcmmGMM`	Growth mixture modeling	**lcmm** (Proust-Lima et al., 2017)
`LMKM`	Feature-based clustering using linear regression and k-means	
`MclustLLPA`	Longitudinal latent profile analysis	**mclust** (Scrucca et al., 2016)
`MixAK_GLMM`	Mixture of generalized linear mixed models	**mixAK** (Komárek, 2009)
`MixtoolsGMM`	Growth mixture modeling	**mixtools** (Benaglia et al., 2009)
`MixtoolsNPRM`	Non-parametric repeated measures clustering	**mixtools** (Benaglia et al., 2009)
`MixTVEM`	Mixture of time-varying effects models	
`Random`	Random partitioning	
`Stratify`	Stratification rule	
`Feature`	Feature-based clustering	

Additionally, a partitioning of trajectories can be specified without an estimation step through the `lcModelPartition` and `lcModelWeightedPartition` classes, providing trajectories with a cluster membership or membership weight, respectively.

3.3 The `lcModel` class

The `lcModel` class represents the estimated cluster solution. It is designed to function as any other model fitted in R. Here, the word "model" should be taken in the broadest sense of the word, where any resulting cluster partitioning represents the data, and thereby is regarded as a model of said data. Users can apply the familiar functions from the **stats** package (R Core Team, 2024) where applicable, including the `predict()`, `plot()`, `summary()`, `fitted()`, and `residuals()` functions. Furthermore, `lcModel` objects support functions for obtaining the cluster representation, such as the cluster proportions, sizes, names, and trajectories.

The base `lcModel` class facilitates basic functionality such as providing a solution summary and providing functionality for computing predictions or fitted values. The two most important functions that characterize the class are the `predict()` and `postprob()` functions. These functions are used to derive the cluster trajectories, the posterior probabilities of the trajectories, and cluster proportions.

The base class stores information regarding the model, including the estimated `lcMethod` object, the `call` that was used to estimate the method, the date and time when the method was estimated, the total estimation time, and a text label for differentiating solutions. Users should not update the slots of the base class directly, except for the `tag` slot, which is intended as a convenient way of assigning custom meta data to the `lcModel`.

The names of subclasses are prefixed by "*lcModel*". Subclasses generally have little need for adding new slots, as most of the functionality resides inside the class functions, such that results and statistics are computed dynamically. This enables fitted `lcModel` objects to be modified retroactively, e.g., for correcting implementation errors that are discovered at a later stage.

In the `lcModel` subclass implementations that are based on an underlying R package, the subclass serves as a wrapper around the underlying package model. The underlying model is exposed via the `getModel()` function so that users can still benefit from the specialized functionality provided by the underlying package.

3.4 The metric interfaces

There is a vast number of metrics available in literature. To provide access to as many metrics as possible, and to enable users to add missing metrics as needed, we define an interface for the computation of metrics. Users can replace or extend the metrics with custom implementations. To ensure a consistent output across all metrics, the output of metric functions must be scalar. Note that some metrics may be undefined for certain types of methods, in which case `NA` is returned (e.g., likelihood-based metrics such as the AIC and BIC are only defined for model-based methods). Currently, the framework supports any of the applicable metrics from the packages **clusterCrit** (Desgraupes, 2023) and **mclustcomp** (You, 2021). The list of supported internal and external metrics is obtained via the `getInternalMetricNames()` and `getExternalMetricNames()` functions, respectively. Metrics can be added or updated via the `defineInternalMetric()` and `defineExternalMetric()` functions.

4 Using the package

We illustrate the main capabilities of the package through a step-by-step exploratory cluster analysis on the longitudinal dataset named `PAP.adh` which is included with the package. This synthetic dataset was simulated based on the real-world study reported by Yi et al. (2022), who investigated the longitudinal CPAP therapy usage patterns of patients with obstructive sleep apnea since the start of their treatment. They identified three distinct patterns of therapy adherence: patients who were adherent to the therapy and stable in their usage ("Adherent"), patients who were consistently non-adherent ("Non-adherent"), and patients who improved their usage over time ("Improvers"). We used the growth mixture model fit reported by the authors to simulate new patients, yielding the `PAP.adh` dataset.

The goal of the analysis is to identify the common patterns of adherence and to establish the most suitable method for the data out of those considered. For brevity, the description of the package function arguments used in the demonstration below is limited to the main arguments. We refer users to the package documentation to learn more about other optional arguments.

The `PAP.adh` dataset comprises records of the weekly average hours of therapy usage of 301 patients in their first 13 weeks of therapy. Therapy usage ranges between 0 and 9.5 hours, with a mean of 4.5 hours. The `PAP.adh` dataset is represented by a `data.frame` in long format, with each row representing the observation of a patient at a specific week (1 to 13).

```
library("latrend")
data("PAP.adh")
head(PAP.adh, n = 3)
```

```
#>   Patient Week UsageHours    Group
#> 1       1    1   6.298703 Adherers
#> 2       1    2   5.916080 Adherers
#> 3       1    3   5.022241 Adherers
```

The `Patient` column indicates the trajectory to which the observation belongs. The `UsageHours` column represents the averaged hours of usage in the respective therapy week, denoted by the `Week` column. The true cluster membership per trajectory is indicated by the `Group` column.

Throughout the analysis, there are several occasions during which the trajectory identifier and time columns would need to be specified. Instead of passing the column names to each function, we can set the default index columns using the `options` mechanism. Keep in mind that this is only recommended during interactive use.

```
options(latrend.id = "Patient", latrend.time = "Week")
```

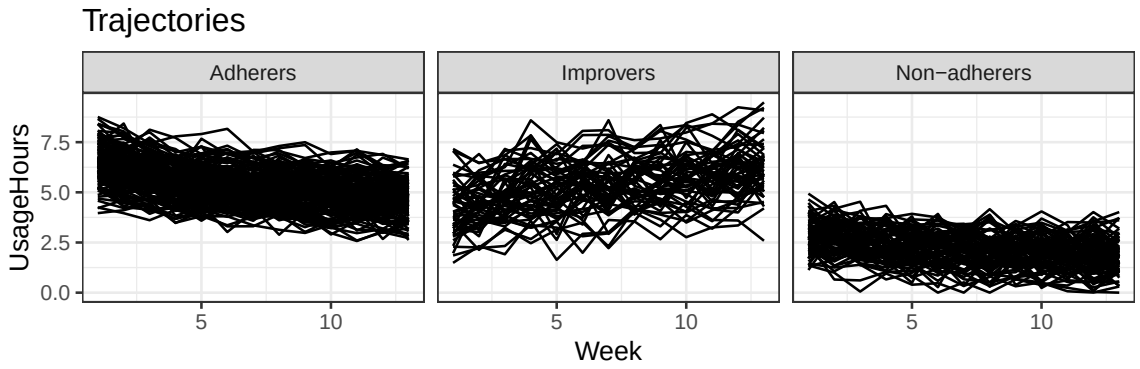


Figure 1: The trajectories from the 'PAP.adh' dataset, by reference group.

We can visualize the patient trajectories using the `plotTrajectories()` function, shown in Figure 1. As the ground truth is known in our synthetic example, we specified the cluster membership of the trajectories via the `cluster` argument, resulting in a stratified visualization.

```
plotTrajectories(PAP.adh, response = "UsageHours", cluster = "Group")
```

4.1 Specifying methods

We first specify the methods to be evaluated. The first method of interest in this case study is KmL, selected for its flexibility in identifying patterns of any shape. The KmL method is available in the framework through the `lcMethodKML` class, which serves as a wrapper around the `kml()` function of the **kml** package (Genolini et al., 2015). The KmL method is specified through the `lcMethodKML()` constructor function.

```
kmlMethod <- lcMethodKML(response = "UsageHours", nClusters = 2)
kmlMethod
```

```
#> lcMethodKML specifying "longitudinal k-means (KML)"
#>  time:           getOption("latrend.time")
#>  id:             getOption("latrend.id")
#>  nClusters:      2
#>  nbRedrawing:    20
#>  maxIt:          200
#>  imputationMethod:"copyMean"
#>  distanceName:   "euclidean"
#>  power:          2
#>  distance:       function() {}
#>  centerMethod:   meanNA
#>  startingCond:   "nearlyAll"
#>  nbCriterion:    1000
#>  scale:          TRUE
#>  response:       "UsageHours"
```

Note that any unspecified arguments have been set to the default values defined by the **kml** package. The method arguments can be accessed using the `$` or `[[` operator. Requested arguments are evaluated unless disabled by the argument `eval = FALSE`. As can be seen in the method output below, the time index column is obtained from the `options` mechanism by default.

```
kmlMethod$time
```

```
#> [1] "Week"

kmlMethod["time", eval = FALSE]

#> getOption("latrend.time")
```

Next, we specify the other methods of interest. We use a variety of approaches that are applicable to this type of data. We evaluate a feature-based approach based on LMKM as implemented in `lcMethodLMKM`, a distance-based dynamic time warping approach via `lcMethodDtwclust` based on the **dtwclust** package, and the regression-based approaches via the `lcMethodLcmmGBTM` and `lcMethodLcmmGMM` methods based on the **lcmm** package (Proust-Lima et al., 2017). We specify the distance-based approach using dynamic time warping. For LMKM, GBTM and GMM, we model the trajectories using an intercept and slope[2]. Moreover, GBTM and GMM are specified to use a shared diagonal variance-covariance matrix. The GMM defines a random patient intercept.

```
dtwMethod <- lcMethodDtwclust(response = "UsageHours", distance = "dtw_basic")
lmkmMethod <- lcMethodLMKM(formula = UsageHours ~ Week)
gbtmMethod <- lcMethodLcmmGBTM(fixed = UsageHours ~ Week,
  mixture = ~ Week, idiag = TRUE)
gmmMethod <- lcMethodLcmmGMM(fixed = UsageHours ~ Week,
  mixture = ~ Week, random = ~ 1, idiag = TRUE)
```

The method arguments of a `lcMethod` object cannot be modified. Instead, a new specification is created from the existing one with the updated method arguments. Any `lcMethod` object can be used as a prototype for creating a new specification with new, modified, or removed arguments using the `update()` function. As an example, if we would like to respecify KmL to identify three clusters, this can be done by updating the existing specification as follows:

```
kml3Method <- update(kmlMethod, nClusters = 3)
```

As the number of clusters is generally not known in advance, we need to fit the methods for a range of number of clusters. Generating specifications for a series of argument values can be done via the `lcMethods()` function, which outputs a `list` of updated `lcMethod` objects from a given prototype. We specify each method for up to six clusters[3] using:

```
kmlMethods  <- lcMethods(kmlMethod,  nClusters = 1:6)
lmkmMethods <- lcMethods(lmkmMethod, nClusters = 1:6)
dtwMethods  <- lcMethods(dtwMethod,  nClusters = 2:6)
gbtmMethods <- lcMethods(gbtmMethod, nClusters = 1:4)
gmmMethods  <- lcMethods(gmmMethod,  nClusters = 1:4)
length(gmmMethods)

#> [1] 4
```

4.2 Fitting methods

Using the previously created method specifications, we can estimate the methods for the `PAP.adh` data. For estimating a single method, we can use the `latrend()` function. The function optionally accepts an `environment` through the `envir` argument for evaluating the method arguments within a specific environment. The output of the function is the fitted `lcModel` object.

[2]For methods supporting formula input, the response variable is automatically determined from the response of the formula.

[3]Only one to four clusters were estimated for GBTM and GMM due to the relatively excessive computation time

```
lmkm2 <- latrend(lmkmMethod, data = PAP.adh)
summary(lmkm2)

#> Longitudinal cluster model using lmkm
#> lcMethodLMKM specifying "lm-kmeans"
#>  time:           "Week"
#>  id:             "Patient"
#>  nClusters:      2
#>  center:         function (x) {    mean(x, na.rm = TRUE)}
#>  standardize:    `scale`
#>  method:         "qr"
#>  model:          TRUE
#>  y:              FALSE
#>  qr:             TRUE
#>  singular.ok:    TRUE
#>  contrasts:      NULL
#>  iter.max:       10
#>  nstart:         1
#>  algorithm:      `c("Hartigan-Wong", "Lloyd", "Forgy", "M
#>  formula:        UsageHours ~ Week
#>
#> Cluster sizes (K=2):
#>           A           B
#> 135 (44.9%) 166 (55.1%)
#>
#> Number of obs: 3913, strata (Patient): 301
#>
#> Scaled residuals:
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -2.52815 -0.67127 -0.06772  0.00000  0.54587  4.04438
```

Instead of needing to update a method prior to calling `latrend()`, the arguments to be updated can also be passed directly to `latrend()`. Here, we estimate the LMKM method for three clusters.

```
lmkm3 <- latrend(lmkmMethod, nClusters = 3, data = PAP.adh)
```

Alternatively, we can achieve the same result by updating the previously estimated two-cluster solution.

```
lmkm3 <- update(lmkm2, nClusters = 3)
```

Batch estimation

The `latrendBatch()` function estimates a list of method specifications. This is useful for evaluating a method for a range of number of clusters, as we have defined above using the `lcMethods()` function. Another use case is the improvement of model convergence and the estimation time by tuning the control parameters. Optimizing such parameters may yield considerably improved convergence or considerably reduced estimation time on larger datasets. Many of the methods have settings for the number of random starts, maximum number of iterations, and convergence criteria. However, because such control settings are specific to each method, we will not cover this.

The inputs to the `latrendBatch()` function are a list of `lcMethod` objects, and a list of datasets. The output is an `lcModels` object, representing a list of the fitted `lcModel` objects for each dataset. A seed is specified to ensure reproducibility of the examples.

```
lmkmList <- latrendBatch(lmkmMethods, data = PAP.adh, seed = 1)
lmkmList

#> List of 6 lcModels with
#>   .name .method       seed nClusters
#> 1     1    lmkm  762473831         1
#> 2     2    lmkm 1762587819         2
#> 3     3    lmkm 1463113723         3
#> 4     4    lmkm 1531473323         4
#> 5     5    lmkm 1922000657         5
#> 6     6    lmkm 1985277999         6
```

When printing a `lcModels` object, the content is shown as a table of method specifications. By default, only arguments which differ between the models are shown. The table can also be obtained as a `data.frame` by calling `as.data.frame()`. We now fit the other methods in the same manner.

```
dtwList <- latrendBatch(dtwMethods, data = PAP.adh, seed = 1)
```

For the repeated estimation of more computationally intensive methods, we can speed up the process by using parallel computation. By setting `parallel = TRUE`, the `latrendBatch()` function will use the parallel back-end of the **foreach** package (Microsoft and Weston, 2022). To make use of this functionality, we first need to configure the parallel back-end:

```
nCores <- parallel::detectCores(logical = FALSE)
if (.Platform$OS.type == "windows") {
  doParallel::registerDoParallel(parallel::makeCluster(nCores))
} else {
  doMC::registerDoMC(nCores)
}
```

The methods can then be estimated in parallel using:

```
kmlList  <- latrendBatch(kmlMethods,
  data = PAP.adh, parallel = TRUE, seed = 1)
gbtmList <- latrendBatch(gbtmMethods,
  data = PAP.adh, parallel = TRUE, seed = 1)
gmmList  <- latrendBatch(gmmMethods,
  data = PAP.adh, parallel = TRUE, seed = 1)
```

4.3 Evaluation

Assessing a cluster result

A cluster result is useful only when it describes the data adequately. There are various aspects on which the cluster result can be evaluated, depending on the method and analysis domain:

- The identified solution may not be reliable when the method estimation procedure did not converge. Convergence can be checked via the `converged()` function.
- The cluster solution may comprise empty clusters or clusters with a negligible proportion of trajectories. In such a case, re-estimating the method may yield a better solution. Alternatively, one should consider fitting the method with a lower number of clusters.
- The cluster trajectories may be assessed visually to determine whether the identified patterns are sufficiently distinct.

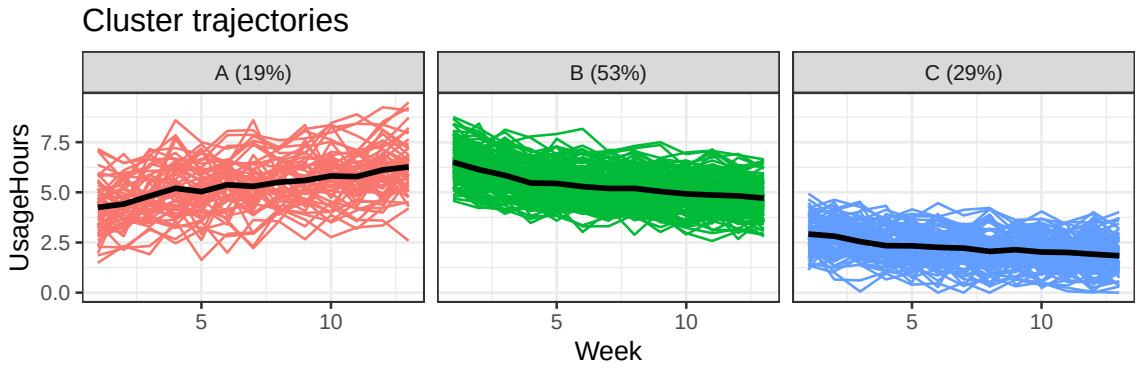


Figure 2: The cluster trajectories of the three-cluster solution identified by LMKM, created by running plot(lmkm3).

- The prediction error may help to determine to what degree trajectories are represented by one of the clusters.

As shown in the previous section, the summary of an `lcModel` object shows the method arguments values, cluster sizes, cluster proportions, cluster names, and the standardized residuals. By default, the residuals are computed from the difference between the reference values and the predictions outputted by `fitted()`, conditional on the most likely trajectory assignments. For methods that do not provide trajectory-specific predictions, the fitted values are determined from the cluster trajectories.

The cluster trajectories can be obtained using the `clusterTrajectories()` function, returning a `data.frame`. The cluster trajectories can be plotted via `plot()` or `plotClusterTrajectories()`. The three-cluster LMKM solution is visualized in Figure 2. For parametric cluster methods, a more concise representation of the model can be obtained from the model coefficients, using `coef()`.

```
plot(lmkm3, linewidth = 1)
```

Assigning descriptive names to the clusters can help to increase the readability of the cluster result, which is especially useful for solutions with many clusters. The `clusterNames()` function can be used to retrieve or change the cluster names.

```
clusterNames(lmkm3) <- c("Struggling", "Increasing", "Decreasing")
```

The most likely cluster for each of the trajectories is obtained using the `trajectoryAssignments()` function, which outputs a `factor` with the cluster names as its levels. For soft-cluster representations, the cluster assignments are determined by the cluster with the highest probability, based on the posterior probability matrix. An alternative approach can be specified through the `strategy` argument. For example, the `which.weight()` function assigns a random cluster weighted by the proportions. The `which.is.max()` function from the **nnet** package (Venables and Ripley, 2002) returns the most likely cluster, breaking ties at random.

The posterior probability matrix can be obtained from the `postprob()` function[4]. For probabilistic methods, it can be used to gauge the cluster separation, i.e., the certainty of assignment. The posterior probability is also important in the post-hoc analysis for accounting for the uncertainty in cluster assignment.

When it comes to longitudinal representation, the minimum functionality that is available for all `lcModel` objects is the prediction of the cluster trajectories at the given moments in time. The prediction has been implemented for underlying packages that lack this functionality. For non-parametric methods such as KmL or LLPA, linear interpolation is used when

[4]For methods that only support modal assignment, the posterior probability matrix only comprises 0 and 1.

time points are requested which are not represented by the cluster centers.The available functionality differs between methods.

All `lcModel` objects support the standard model functions from the standard **stats** package, including `fitted()`, `residuals()`, and `predict()`. These functions are primarily of interest for methods that have a notion of a group or individual trajectory prediction error, such as for the regression-based approaches like GBTM and GMM. The `fitted()` function returns the expected values for the response variable for the data on which the model was estimated. By default, only the values for the most likely cluster are given. However, for `clusters = NULL`, a matrix of predictions is outputted, where each column represents the predictions of the respective cluster.

The `predict()` function computes trajectory- and cluster-specific predictions for the given input data.

```
predict(lmkm3, newdata = data.frame(Week = c(1, 10), Cluster = "Decreasing"))

#>        Fit
#> 1 2.919423
#> 2 2.024865
```

The `predictPostprob()` and `predictAssignments()` functions compute the posterior probability and cluster membership for new trajectories, respectively. As this is not a common use case for cluster methods, most of the underlying packages do not provide this functionality. For demonstration purposes, we have implemented the functionality for the `lcModelKML` class.

Using the metric interface defined in Section 2, we can compute a variety of internal metrics through the `metric()` function:

```
metric(lmkm3, c("MAE", "RMSE", "Dunn", "ASW"))

#>        MAE       RMSE       Dunn        ASW
#> 0.74262252 0.94094913 0.09173111 0.35605235
```

Identifying the number of clusters

Using one or more internal metrics of interest, we can assess how the data representation of a method improves or worsens for an increasing number of clusters. In this case study, we will use the Dunn index as the primary metric for the choice of the number of clusters.

The change in metrics for an increasing number of clusters can be visualized via the `plotMetric()` function, and can help to determine the preferred solution. For brevity, we will only provide a detailed view for the KmL method. We plot the Dunn index, WMAE, and estimation time (in seconds) for the six KmL solutions as follows:

```
plotMetric(kmlList, c("Dunn", "WMAE", "estimationTime"))
```

The resulting plot is shown in Figure 3. The Dunn index and WMAE show a rather convincing improvement for an increasing number of clusters[5].

Moreover, we observe that the estimation time increases with the number of clusters. This can be a practical consideration when deciding on the preferred method to use. For much larger datasets, it may be useful to conduct a preliminary analysis on a subset of the data for possibly ruling out methods which are too computationally intensive in relation to the results.

We can obtain the metric values for each of the models by calling the `metric()` function.

```
metric(kmlList, c("Dunn", "WMAE", "estimationTime"))
```

[5]The Dunn index is not defined for a one-cluster solution.

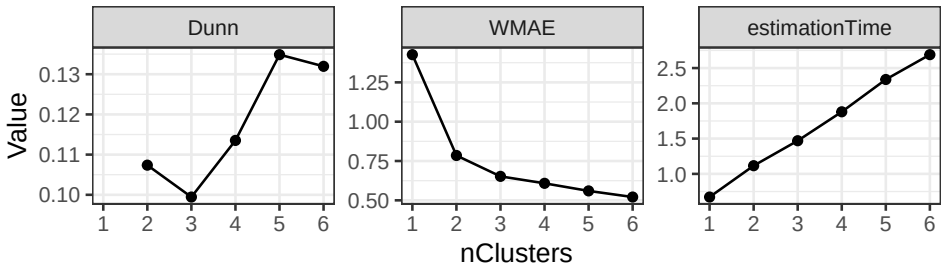


Figure 3: The Dunn index (higher is better), and WMAE (lower is better) metrics for each of the KmL solutions from 1 to 6 clusters

```
#>         Dunn      WMAE estimationTime
#> 1         NA 1.4261264          0.672
#> 2 0.10737225 0.7850566          1.116
#> 3 0.09944419 0.6523208          1.470
#> 4 0.11353357 0.6081128          1.880
#> 5 0.13487175 0.5598639          2.338
#> 6 0.13196444 0.5209264          2.690
```

As the preferred solution corresponds to the highest Dunn index, we can obtain the respective model by calling the `max()` function on the `lcModels` list object.

```
kmlBest <- max(kmlList, "Dunn")
```

Alternatively, we can select the preferred model using the `subset()` function. By specifying the `drop = TRUE`, the `lcModel` object is returned instead of a `lcModels` object.

```
kmlBest <- subset(kmlList, nClusters == 5, drop = TRUE)
```

The identification of the number of clusters is a form of model selection. The same approach can therefore be used for identifying the best cluster representation, e.g., evaluating different formulas for a parametric model, or selecting a different method initialization strategy.

Comparing methods

The optimal number of clusters according to the internal metric can be different for other methods or specifications thereof. Depending on the cluster representation, some methods may require fewer or more clusters to represent the heterogeneity to the same degree. By concatenating the lists of fitted methods, we can create a metric plot that is grouped by the type of method as follows:

```
allList <- lcModels(lmkmList, kmlList, dtwList, gbtmList, gmmList)
plotMetric(
  allList,
  name = c("Dunn", "WMAE", "BIC", "estimationTime"),
  group = '.method'
)
```

The WMAE and BIC between GBTM and KmL are almost exactly the same, possibly indicating that the methods find a similar solution. If the solutions are found to be practically identical, then one could actually prefer KmL due to its considerably favorable computational scaling with the number of clusters.

We explore the best solution of each method further to better understand how the cluster representations differ between the methods. We can select the preferred `lcModel` object

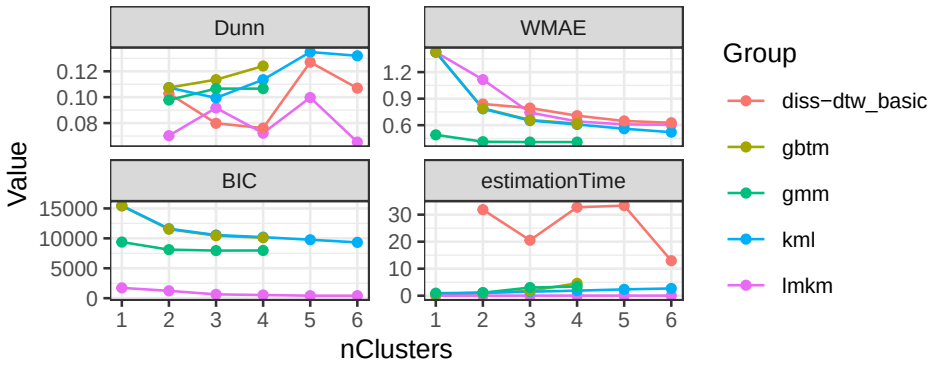


Figure 4: The Dunn index (higher is better), WMAE (lower is better) and BIC (relatively lower is better) for each of the methods and number of clusters

corresponding to the selected number of clusters for each of the methods using the `subset()` function.

```
kmlBest  <- subset(kmlList,  nClusters == 5, drop = TRUE)
dtwBest  <- subset(dtwList,  nClusters == 5, drop = TRUE)
gbtmBest <- subset(lmkmList, nClusters == 4, drop = TRUE)
lmkmBest <- subset(lmkmList, nClusters == 3, drop = TRUE)
gmmBest  <- subset(gmmList,  nClusters == 3, drop = TRUE)
```

We can then assess the pairwise ARI between each method using the `externalMetric()` function. Calling this function on a `lcModels` list returns a `dist` object representing a distance matrix. We therefore create a list of the best `lcModel` for each method, by which we can then determine the pairwise ARI as follows:

```
bestList <- lcModels(KmL = kmlBest, DTW = dtwBest,
  LMKM = lmkmBest, GBTM = gbtmBest, GMM = gmmBest)
externalMetric(bestList, name = "adjustedRand") |> signif(2)
```

```
#>       KmL  DTW LMKM GBTM
#> DTW  0.41
#> LMKM 0.50 0.40
#> GBTM 0.66 0.31 0.67
#> GMM  0.49 0.40 0.99 0.68
```

With all pairwise ARI being at least 0.31, all methods demonstrate some degree of similarity between each other. In particular, the very high ARI of approximately 0.99 between GMM and LMKM implies that the methods grouped the trajectories in a highly similar way.

Secondly, we evaluate the similarity of the cluster trajectories between the methods using the weighted minimum mean absolute error (WMMAE) (Den Teuling et al., 2021b). This metric computes the mean absolute error between each cluster trajectory and its nearest cluster trajectory of the other method, weighted by the size of the respective cluster.

```
externalMetric(bestList, name = "WMMAE") |> signif(2)
```

```
#>        KmL   DTW  LMKM  GBTM
#> DTW  0.096
#> LMKM 0.130 0.130
#> GBTM 0.063 0.130 0.091
#> GMM  0.130 0.130 0.036 0.099
```

The mean absolute error of 0.091 between the cluster trajectories of GBTM and LMKM is negligible compared to the residual error estimated by GBTM (SD = 0.8), which indicates that both methods have identified practically the same cluster trajectories. The same applies to GMM and LMKM.

4.4 Cluster validation

Assessing the stability and reproducibility of a cluster method can help to determine whether the identified cluster solution generalizes beyond the data that was used to estimate the method. This is especially relevant for more complex cluster methods involving many parameters, which may not generalize well to new data. This primarily pertains to the number of clusters the method is estimated for, as the number of parameters increases linearly with the number of clusters. Even relatively simple methods can overfit the data when the representation comprises too many clusters in relation to the sample size.

Cluster stability using repeated estimation

Many of the cluster methods can yield a slightly different solution during each run, depending on the starting conditions. In such cases, by doing a repeated estimation, we can gauge the stability, i.e., consistency, of the method. Comparing repeated estimation results is also useful for selecting the best solution for a given method.

Repeated estimation can be done via the `latrendRep()` function, where the number of repetitions is specified via the `.rep` argument. Similar to `latrend()`, the method arguments can be updated within the function. The function returns a `lcModels` object, comprising a `list` of `lcModel` objects. Here, we only use five repeated estimations to limit the computation time. In practice, a higher number such as 10 or 25 is advisable, depending on the magnitude of instability.

```
kmlRepList <- latrendRep(kmlMethod, data = PAP.adh,
  nClusters = 5, .rep = 5, .parallel = TRUE)
summary(metric(kmlRepList, c("Dunn", "WMAE")))
```

```
#>       Dunn             WMAE
#>  Min.   :0.1286   Min.   :0.5617
#>  1st Qu.:0.1286   1st Qu.:0.5619
#>  Median :0.1286   Median :0.5658
#>  Mean   :0.1311   Mean   :0.5643
#>  3rd Qu.:0.1349   3rd Qu.:0.5658
#>  Max.   :0.1349   Max.   :0.5665
```

The result suggests that the solutions found by KmL are highly consistent on this dataset, in the sense that both metrics demonstrate a negligible level of variability between repeated estimations.

Cluster stability using bootstrap sampling

Instead of assessing the cluster stability across repeated estimation on the same dataset, we can obtain a more generalizable estimate of the cluster stability in a nonparametric way by measuring the cluster stability across different datasets. This form of bootstrap sampling, also referred to as bootstrapping, involves the repeated estimation on simulated datasets generated from the original dataset. It is primarily used for assessing the stability of a method, as measured by one or more internal metrics. Here, complete trajectories are selected at random with replacement from the dataset to generate a new dataset of equal size. Each simulated dataset, referred to as a bootstrap sample in this context, will yield a slightly different solution. This variability between samples can provide an indication of the stability

of the cluster method on the overall dataset (Hennig, 2007). Since the repeated estimation is done on new datasets that only partially overlap[6], this restricts the available external metrics to only those that can compare between different datasets, e.g., the WMMAE.

The `latrendBoot()` function applies bootstrapping to the given method specification. The `samples` argument determines the number of times the data is resampled, and a model is estimated. Setting the `seed` argument ensures that the same sequence of bootstrap samples is generated when redoing the bootstrapping procedure. The output is a `lcModels` list containing the model for each sample. The estimated methods each have a different `call` for the `data` argument such that the original bootstrap training sample can be recreated as needed. This avoids the need for models to store the training data. As an example, we compute 10 bootstrap samples[7] (i.e., repeated fits) in parallel as follows:

```
kmlMethodBest <- update(kmlMethod, nClusters = 5)
kmlBootModels <- latrendBoot(kmlMethodBest, data = PAP.adh,
  samples = 10, seed = 1, parallel = TRUE)
head(kmlBootModels, n = 3)
```

```
#> List of 3 lcModels with
#>   .name .method                                          data       seed
#> 1     1     kml  bootSample(PAP.adh, "Patient", 762473831L) 1062140483
#> 2     2     kml bootSample(PAP.adh, "Patient", 1762587819L)  185557490
#> 3     3     kml bootSample(PAP.adh, "Patient", 1463113723L)  934902099
```

We can now assess the stability of the solutions across the models in terms of metrics of interest. Here, we assess the mean convergence rate, and the quantiles of the WMAE and Dunn metrics.

```
bootMetrics <- metric(kmlBootModels, c("converged", "Dunn", "WMAE"))
mean(bootMetrics$converged)
```

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

```
summary(bootMetrics$Dunn)
```

```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.1193  0.1469  0.1512  0.1530  0.1645  0.1852
```

```
summary(bootMetrics$WMAE)
```

```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.5326  0.5489  0.5542  0.5530  0.5576  0.5728
```

As can be seen from the output, there is quite some variability between the estimated solutions across bootstrap samples. This suggests that we should consider estimation with repeated random starts to identify a better and more stable solution.

Lastly, we can compute a similarity matrix for an external metric of interest, containing the pairwise similarity for each model pair.

```
wmmaeDist <- externalMetric(kmlBootModels[1:10], name = "WMMAE")
summary(wmmaeDist)
```

```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#> 0.01594 0.05696 0.07259 0.06952 0.08264 0.11011
```

Showing that there is only a small degree of discrepancy in the cluster trajectories between bootstrap samples.

[6] In addition to the challenge of the cluster representations being in a different order between runs, also referred to as label switching.

[7] In practice, a much greater number of bootstrap samples is recommended (at least 100).

Comparison to ground truth

We now consider the case where a method is evaluated in a simulation study. In such a study, the ground truth is known, and we can directly evaluate whether the trajectories are clustered correctly. A useful and intuitive measure is the split-join distance (van Dongen, 2000), which is an edit distance that measures the number of trajectory reassignments that are needed to go from one partitioning to another. In case of a ground truth, we are only interested in the edit distance from the reference partitioning[8].

We can obtain the vector of trajectory cluster membership of the `PAP.adh` from the `Group` column by selecting the first cluster name of each trajectory, since the cluster membership is stable over time. We then create a `lcModelPartition` from the computed membership vector. By default, the `lcModelPartition` generates the cluster representations from the means of the trajectories assigned to the respective cluster.

```
refAssignments <- aggregate(Group ~ Patient, data = PAP.adh, FUN = head, n = 1L)
refAssignments$Cluster = refAssignments$Group

refModel <- lcModelPartition(data = PAP.adh,
  trajectoryAssignments = refAssignments, response = "UsageHours")
refModel
```

```
#> Longitudinal cluster model using part
#> lcMethod specifying "undefined"
#> no arguments
#>
#> Cluster sizes (K=3):
#>     Adherers    Improvers Non-adherers
#>  162 (53.8%)   56 (18.6%)   83 (27.6%)
#>
#> Number of obs: 3913, strata (Patient): 301
#>
#> Scaled residuals:
#>      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
#> -3.894748 -0.643671 -0.009533  0.000000  0.634893  3.590377
```

We can now compare our selected method solutions to the reference solution using the one-way split-join distance to the reference:

```
externalMetric(bestList, refModel, name = "splitJoin.ref", drop = FALSE)
```

```
#>      splitJoin.ref
#> KmL             23
#> DTW             61
#> LMKM             3
#> GBTM             1
#> GMM              2
```

This shows that, for the `PAP.adh` dataset, LMKM, GBTM, and GMM achieve a nearly perfect recovery of the cluster memberships, but that GBTM needs more clusters to represent the dataset.

5 Implementing new methods

One of the main strengths of the framework is the standard way in which methods are specified, estimated, and evaluated. These aspects make it easy to compare newly implemented

[8] In the one-way edit distance, a solution that has more clusters than the reference can still obtain an edit distance of zero if the extra clusters are a subset of the cluster of the reference.

methods with existing ones. Using the base classes `lcMethod` and `lcModel`, new methods can be implemented with a relatively minimal amount of code, enabling rapid prototyping. These classes provide basic functionality, from which the user can extend certain functions as needed by creating a subclass.

5.1 Stratification

The simplest form of clustering is the stratification of the dataset based on a known factor. This can be the response variable, or any other measure available for each trajectory. This is useful for case studies where there is prior knowledge or expert guidance on how the trajectories should be grouped; either by another factor (e.g., age or gender), or a characteristic of the trajectory (e.g., the intercept, slope, average, or variance).

A stratification approach can be specified using the `lcMethodStratify()` function, which takes an R expression as input. The expression is evaluated within the `data.frame` at the trajectory level during the method estimation, so any column present in the data can be used. The expression should resolve to a number or category, indicating the stratum for the respective trajectory.

As an example, we stratify the trajectories by thresholding on the mean hours of usage. This expression returns a `logical` value which determines the cluster assignment. For categorizing trajectories into more than two clusters, the `cut()` function can be used. The cluster trajectories are computed by aggregating the trajectories of each cluster at the respective time points. By default, the average is computed, but an alternative center function can be specified via the `center` argument.

```
stratMethod <- lcMethodStratify(
  response = "UsageHours",
  stratify = mean(UsageHours) > 4
)
stratModel <- latrend(stratMethod, data = PAP.adh)
clusterProportions(stratModel)
```

```
#>         A         B
#> 0.3156146 0.6843854
```

5.2 Feature-based clustering

Feature-based clustering is a flexible and fast approach to clustering longitudinal data, with an essentially limitless choice of trajectory representations. The framework includes a generic feature-based clustering class named `lcMethodFeature` for quickly implementing this approach.

A `lcMethodFeature` specification requires two functions: A representation function outputting the trajectory representation `matrix`, and a cluster function that applies a cluster algorithm to the matrix, returning an `lcModel` object.

To illustrate the method, we represent each trajectory using a linear model, and we cluster the model coefficients using *k*-means. In the representation step, `lm()` is applied to each trajectory, and the model coefficients are combined into a `matrix` with the trajectory-specific coefficients on each row. We parameterize the `lcMethod` implementation by obtaining the model formula from `method$formula`. During the method specification, the user therefore needs to define the `formula` argument. The representation function is as follows:

```
repStep <- function(method, data, verbose) {
  repTraj <- function(trajData) {
    lm.rep <- lm(method$formula, data = trajData)
    coef(lm.rep)
  }
```

```
  dt <- as.data.table(data)
  coefData <- dt[, as.list(repTraj(.SD)), keyby = c(method$id)]
  coefMat <- as.matrix(subset(coefData, select = -1))
  rownames(coefMat) <- coefData[[method$id]]
  coefMat
}
```

We implement the cluster step to return a `lcModelPartition` object based on the cluster assignments outputted by `kmeans()`. We have parameterized the function by obtaining the number of clusters for *k*-means from the `nClusters` model argument. The cluster function is as follows:

```
clusStep <- function(method, data, repMat, envir, verbose) {
  km <- kmeans(repMat, centers = method$nClusters)
  lcModelPartition(response = responseVariable(method), method = method,
    data = data, trajectoryAssignments = km$cluster, center = mean)
}
```

We can now specify and estimate the feature-based method, including the additionally required arguments. Comparing the estimated model to the preferred KmL model, we see that the solutions have a relatively high degree of overlap.

```
tsMethod <- lcMethodFeature(response = "UsageHours", formula = UsageHours ~ Week,
  representationStep = repStep, clusterStep = clusStep)
tsModel <- latrend(tsMethod, data = PAP.adh, nClusters = 5)
externalMetric(tsModel, kmlBest, "adjustedRand")
```

```
#> adjustedRand
#>    0.6283228
```

```
externalMetric(tsModel, kmlBest, "WMMAE")
```

```
#>      WMMAE
#> 0.06505086
```

5.3 Implementing a method

The framework is designed to support the implementation of new methods, so that users can extend or implement new methods to address their use case. In this section, we describe the high-level steps that are involved in adding support for a method to the framework. Considering the number of lines of code for even a relatively simple cluster method, we do not cover a complete example here. Instead, we only outline the typical set of functions that need to be implemented, together with any relevant input and output assumptions of these functions. For complete examples, see the `lcMethod`-interface implementations based on external packages, e.g., `lcMethodKML` or `lcMethodLcmmGMM`. A step-by-step example of implementing a statistical method in the framework can be found in the vignette included with the package, which can be viewed by running `vignette("implement", package = "latrend")`.

The estimation process of a method is divided into six steps, involving the processing of the method arguments, preparing and validating the data, and fitting the specified method. All steps except for `fit()` are optional.

1. The `prepareData()` function transforms the training data into the required format for the internal method estimation code. By default, data is provided in long format in a `data.frame`. For most implementations, no transformation is therefore needed. Cluster methods for repeated-measures data typically require data to be transformed to `matrix` format, however.

2. The `compose()` function evaluates the method arguments and returns an updated `lcMethod` object with the evaluated method arguments. The function can also be used for modifying or even replacing the original `lcMethod` object for the remainder of the estimation process. This is useful when a method is a special case of a more general method and intends to conceal derivative or redundant arguments from the base class.
3. The `validate()` function enables evaluated method arguments to be checked against the input data. This can be used, for example, for checking whether the data contains the covariates specified in the method formula, or whether an argument has a valid value. For implementations which wrap an underlying package function, this validation is usually not needed as the underlying package already performs validation of the input.
4. The `preFit()` function is intended for processing any arguments prior to fitting. In order for these results to be persistent, they should be returned in an `environment` object, which will be passed as an input to the `fit()` function.
5. The `fit()` function is where the internal method is estimated for the given specification to obtain the cluster result. This function is also responsible for creating the corresponding `lcModel` object. The running time of this function is used to determine the method estimation time.
6. The `postFit()` function takes the outputted `lcModel` from `fit()` as input, enabling post-processing to be done. This is used, for example, for computing derivative statistics, or for reducing the memory footprint by stripping redundant data fields from the internal model representation. Preferably, this function is implemented such that it can be called repeatedly, allowing for updates to fitted methods without requiring re-estimation.

The implementation of a method requires defining a new `lcMethod` class. Usually, a new `lcModel` class needs to be implemented to handle the result and representation of the fitted method. If the new method only outputs a partitioning, then the `lcModelPartition` class may be used instead.

6 Summary and outlook

The **latrend** package facilitates the standardized yet flexible exploration of heterogeneity in longitudinal datasets, with a minimal amount of coding effort. The framework provides functionality for specifying, estimating, and assessing models for clustering longitudinal data. The package builds upon the efforts of the R community by providing an interface to the many methods for clustering longitudinal data across packages. Perhaps most importantly, the **latrend** package makes it easy to compare between any two cluster methods, enabling users to identify the most suitable method to their use case. To ensure transparent and reproducible research, all decisions and settings that are relevant to the analysis should be reported. A useful checklist for reporting on latent-class trajectory studies is provided by van de Schoot et al. (2017), which is also relevant to longitudinal cluster analyses in general.

Users can implement new methods within the framework or add support for other packages, enabling rapid prototyping for the case study at hand. Additionally, the standard functionality provided by the framework also reduces the effort needed in implementing a longitudinal cluster model.

We encourage the framework to be used as a first exploratory step in clustering longitudinal data, after which the identified preferred method can then be applied directly from the original package, which typically provides special tools or options not provided by the framework. To illustrate one such limitation, consider the initialization or prior specification of a longitudinal cluster model. This is generally an important aspect of model estimation that can improve the identified model solution but is challenging to facilitate in a standardized way.

The framework is currently focused towards the modeling of a single continuous response variable, whereas some of the supported cluster packages already support multitra-

jectory modeling. The possible support for multitrajectory modeling has been accounted for in the design of the software. Similarly, while the single response is required to be numerical, support could be added for categorical outcomes such as those used in longitudinal latent class analysis.

Overall, we intend the framework to bridge the different approaches to clustering longitudinal data that exist from the various areas of research. We encourage users and package developers to create interfaces for their methods, as the availability of a standard framework for performing a longitudinal cluster analysis lowers the barrier to evaluating and comparing methods for applied researchers.

Computational details

The examples and figures in this paper were obtained using R 4.5.1 (R Core Team, 2024) with the packages **latrend** 1.6.2, **ggplot2** 4.0.0.9000 (Wickham, 2016), and **data.table** 1.17.8 (Barrett et al., 2024). The KmL method was estimated with the **kml** 2.5.0 package. The distance-based method used the **dtwclust** 6.0.0 package. The GBTM and GMM analyses were performed using the **lcmm** 2.2.1 package, with the parallel computation achieved using the **foreach** 1.5.2 package (Microsoft and Weston, 2022).

R and all packages used within the article and the **latrend** package are available from the Comprehensive R Archive Network (CRAN) at (`https://CRAN.R-project.org`).

Acknowledgments

This work was supported by Philips Research, Eindhoven, the Netherlands. Niek Den Teuling and Steffen Pauws are employees of Philips. We are grateful for the feedback and insightful suggestions provided by the anonymous reviewers. The development of this framework builds upon the work of the R community in the area of longitudinal data, clustering, and cluster metrics.

References

M. Adepeju, S. Langton, and J. Bannister. Akmedoids R package for generating directionally-homogeneous clusters of longitudinal data sets. *Journal of Open Source Software*, 5(56):2379, 2020. doi: 10.21105/joss.02379. URL `https://doi.org/10.21105/joss.02379`. [p109, 112, 115]

S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah. Time-series clustering - a decade review. *Information Systems*, 53:16–38, 2015. doi: 10.1016/j.is.2015.04.007. [p111]

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. ISSN 0031-3203. doi: 10.1016/j.patcog.2012.07.021. [p112]

S. F. Babbin, W. F. Velicer, M. S. Aloia, and C. A. Kushida. Identifying longitudinal patterns for individuals and subgroups: An example with adherence to treatment for obstructive sleep apnea. *Multivariate Behavioral Research*, 50(1):91–108, 2015. doi: 10.1080/00273171.2014.958211. [p108]

T. Barrett, M. Dowle, A. Srinivasan, J. Gorecki, M. Chirico, T. Hocking, and B. Schwendinger. *data.table: Extension of 'data.frame'*, 2024. URL `https://CRAN.R-project.org/package=data.table`. R package version 1.16.4. [p132]

D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48, 2015. doi: 10.18637/jss.v067.i01. [p115]

T. Benaglia, D. Chauveau, D. R. Hunter, and D. Young. mixtools: An R package for analyzing finite mixture models. *Journal of Statistical Software*, 32(6):1–29, 2009. URL http://www.jstatsoft.org/v32/i06/. [p109, 112, 116]

C. Bouveyron. *funFEM: Clustering in the Discriminative Functional Subspace*, 2021. URL https://CRAN.R-project.org/package=funFEM. R package version 1.2. [p109, 115]

E. A. Cayanan, D. J. Bartlett, J. L. Chapman, C. M. Hoyos, C. L. Phillips, and R. R. Grunstein. A review of psychosocial factors and personality in the treatment of obstructive sleep apnoea. *European Respiratory Review*, 28(152), 2019. doi: 10.1183/16000617.0005-2019. [p108]

R. de la Cruz-Mesía, F. A. Quintana, and G. Marshall. Model-based clustering for longitudinal data. *Computational Statistics & Data Analysis*, 52(3):1441–1457, 2008. doi: 10.1016/j.csda.2007.04.005. [p111]

N. G. P. Den Teuling, E. R. V. den Heuvel, M. S. Aloia, and S. C. Pauws. A latent-class heteroskedastic hurdle trajectory model: Patterns of adherence in obstructive sleep apnea patients on CPAP therapy. *BMC Medical Research Methodology*, 21(1):1–15, 2021a. doi: 10.1186/s12874-021-01407-6. [p108]

N. G. P. Den Teuling, S. C. Pauws, and E. R. V. den Heuvel. A comparison of methods for clustering longitudinal data with slowly changing trends. *Communications in Statistics - Simulation and Computation*, 2021b. doi: 10.1080/03610918.2020.1861464. [p114, 125]

G. V. der Nest, V. L. Passos, M. J. Candel, and G. J. V. Breukelen. An overview of mixture modelling for latent evolutions in longitudinal data: Modelling approaches, fit statistics and software. *Advances in Life Course Research*, 43:100323, 2020. ISSN 1040-2608. doi: 10.1016/j.alcr.2019.100323. [p112]

B. Desgraupes. *clusterCrit: Clustering Indices*, 2023. URL https://CRAN.R-project.org/package=clusterCrit. R package version 1.3.0. [p117]

J. J. Dziak, R. Li, X. Tan, S. Shiffman, and M. P. Shiyko. Modeling intensive longitudinal data with mixtures of nonparametric trajectories and time-varying effects. *Psychological Methods*, 20(4):444–469, 2015. ISSN 1939-1463. doi: 10.1037/met0000048. [p109]

C. Genolini, X. Alacoque, M. Sentenac, and C. Arnaud. kml and kml3d: R packages to cluster longitudinal data. *Journal of Statistical Software*, 65(4):1–34, 2015. URL http://www.jstatsoft.org/v65/i04/. [p109, 110, 115, 118]

B. Grün and F. Leisch. FlexMix version 2: Finite mixtures with concomitant variables and varying and constant parameters. *Journal of Statistical Software*, 28(4):1–35, 2008. doi: 10.18637/jss.v028.i04. URL http://www.jstatsoft.org/v28/i04/. [p109, 115]

E. L. Hamaker. Why researchers should think "within-person": A paradigmatic rationale. In M. R. Mehl and T. S. Conner, editors, *Handbook of Research Methods for Studying Daily Life*, pages 43–61. Guilford Publications, 2012. ISBN 1462513050. [p108]

C. Hennig. Cluster-wise assessment of cluster stability. *Computational Statistics & Data Analysis*, 52(1):258–271, 2007. ISSN 0167-9473. doi: 10.1016/j.csda.2006.11.025. URL http://www.sciencedirect.com/science/article/pii/S0167947306004622. [p127]

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. doi: 10.1007/BF01908075. [p113]

A. Komárek. A new R package for Bayesian estimation of multivariate normal mixtures allowing for selection of the number of components and interval-censored data. *Computational Statistics & Data Analysis*, 53(12):3932–3947, 2009. doi: 10.1016/j.csda.2009.05.006. [p109, 112, 116]

T. W. Liao. Clustering of time series data — a survey. *Pattern Recognition*, 38(11):1857–1874, 2005. ISSN 0031-3203. doi: 10.1016/j.patcog.2005.01.025. [p110, 112]

D. P. Martin and T. von Oertzen. Growth mixture models outperform simpler clustering algorithms when detecting longitudinal heterogeneity, even with small sample sizes. *Structural Equation Modeling: A Multidisciplinary Journal*, 22(2):264–275, 2015. ISSN 1070-5511. doi: 10.1080/10705511.2014.936340. [p114]

P. D. McNicholas and T. B. Murphy. Model-based clustering of longitudinal data. *Canadian Journal of Statistics*, 38(1):153–168, 2010. [p110]

Microsoft and S. Weston. *foreach: Provides Foreach Looping Construct*, 2022. URL https://CRAN.R-project.org/package=foreach. R package version 1.5.2. [p121, 132]

B. Muthén. Latent variable analysis: Growth mixture modeling and related techniques for longitudinal data. In *The SAGE Handbook of Quantitative Methodology for the Social Sciences*, pages 346–369. SAGE Publications, Inc., 2004. doi: 10.4135/9781412986311.n19. [p110, 111]

D. S. Nagin, B. L. Jones, V. L. Passos, and R. E. Tremblay. Group-based multi-trajectory modeling. *Statistical Methods in Medical Research*, 27(7):2015–2023, 2018. doi: 10.1177/0962280216673085. [p112]

J. D. Nielsen. *crimCV: Group-Based Modelling of Longitudinal Data*, 2023. URL https://CRAN.R-project.org/package=crimCV. R package version 1.0.0. [p109, 111, 115]

C. Proust-Lima, V. Philipps, and B. Liquet. Estimation of extended mixed models using latent classes and latent processes: The R package lcmm. *Journal of Statistical Software*, 78 (2):1–56, 2017. doi: 10.18637/jss.v078.i02. [p109, 111, 112, 116, 119]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2024. URL https://www.R-project.org/. [p109, 114, 116, 132]

P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: 10.1016/0377-0427(87)90125-7. [p112]

A. Sardá-Espinosa. Time-series clustering in R using the dtwclust package. *The R Journal*, 2019. doi: 10.32614/RJ-2019-023. [p109, 111, 115]

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): 205–233, 2016. URL https://journal.r-project.org/archive/2016-1/scrucca-fop-murphy-etal.pdf. [p109, 110, 116]

R. van de Schoot, M. Sijbrandij, S. D. Winter, S. Depaoli, and J. K. Vermunt. The GRoLTS-checklist: Guidelines for reporting on latent trajectory studies. *Structural Equation Modeling: A Multidisciplinary Journal*, 24(3):451–467, 2017. doi: 10.1080/10705511.2016.1247646. [p131]

S. van Dongen. Performance criteria for graph clustering and Markov cluster experiments. Technical Report INS-R0012, CWI (Centre for Mathematics and Computer Science), 2000. [p128]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, 4th edition, 2002. URL https://www.stats.ox.ac.uk/pub/MASS4/. [p122]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2nd edition, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p132]

H. Yi, X. Dong, S. Shang, C. Zhang, L. Xu, and F. Han. Identifying longitudinal patterns of CPAP treatment in OSA using growth mixture modeling: Disease characteristics and psychological determinants. *Frontiers in Neurology*, 13:1063461, 2022. doi: 10.3389/fneur.2022.1063461. [p108, 117]

K. You. *mclustcomp: Measures for Comparing Clusters*, 2021. URL https://CRAN.R-project.org/package=mclustcomp. R package version 0.3.3. [p117]

Eindhoven University of Technology, and Philips Research

https://github.com/niekdt
ORCiD: 0000-0003-1026-5080
niek.den.teuling@philips.com

Steffen Pauws
Tilburg University, and Philips Research

ORCiD: 0000-0003-2257-9239
s.c.pauws@tilburguniversity.edu

Edwin van den Heuvel
Eindhoven University of Technology

ORCiD: 0000-0001-9157-7224
e.r.v.d.heuvel@tue.nl

Structured Bayesian Regression Tree Models for Estimating Distributed Lag Effects: The R Package dlmtree

by Seongwon Im, Ander Wilson, and Daniel Mork

Abstract When examining the relationship between an exposure and an outcome, there is often a time lag between exposure and the observed effect on the outcome. A common statistical approach for estimating the relationship between the outcome and lagged measurements of exposure is a distributed lag model (DLM). Because repeated measurements are often autocorrelated, the lagged effects are typically constrained to vary smoothly over time. A recent statistical development on the smoothing constraint is a tree structured DLM framework. We present an R package dlmtree, available on CRAN, that integrates tree structured DLM and extensions into a comprehensive software package with user-friendly implementation. A conceptual background on tree structured DLMs and a demonstration of the fitting process of each model using simulated data are provided. We also demonstrate inference and interpretation using the fitted models, including summary and visualization. Additionally, a built-in shiny app for heterogeneity analysis is included.

1 Introduction

In many fields, there is interest in estimating the lagged relationship between an exposure (or treatment) and an outcome. In such cases, the length of the lag or how the exposure effect varies is often unknown. The lagged relationship can take either of two forms. The first form is the association between the exposure at one time point and an outcome distributed across several subsequent times. For example, the impact of advertisement persists beyond that single time point of the investment (Koyck, 1954; Palda, 1965) or exposure to an environmental pollutant affects mortality on the same day and each of the following days (Schwartz, 2000). The second form is an exposure assessed longitudinally and an outcome assessed post-exposure. Examples of this form are maternal exposure to environmental chemicals during pregnancy on birth outcomes and children's health and development (Wilson et al., 2017a; Chiu et al., 2023; Hsu et al., 2023) and the effect of training and recovery activities over multiple days on athlete wellness (Schliep et al., 2021). A popular statistical method to estimate the time-varying association between exposure and outcome is a distributed lag model (DLM).

The DLM regresses a scalar outcome on the exposure measured at preceding time points (Schwartz, 2000; Gasparrini et al., 2010). The DLM framework is particularly useful as it allows for estimating time-resolved exposure effects and quantifies the temporal relationship between the exposure and the outcome. Because the repeated measurements of exposure are often correlated, DLMs typically include a smoothing constraint to add temporal structure to the estimated time-specific effects and to regularize the exposure effects in the presence of multicollinearity across the measurements. Constraints include polynomials, splines, and Gaussian processes (Zanobetti et al., 2000; Warren et al., 2012). Notably, Mork and Wilson (2023) introduced an approach for constrained DLM using regression tree structures based on the Bayesian additive regression tree (BART) framework (Chipman et al., 2010). More recent methods have extended the tree structured DLM framework in several directions, including nonlinear exposure-response relationships (Mork and Wilson, 2022), models with lagged interactions among multiple exposures (Mork and Wilson, 2023), and models with heterogeneous lag effects (Mork et al., 2024).

There are several related methods and packages for DLM implementation. Table 1 lists currently available R packages for DLMs that are generally appropriate for the type of epidemiology studies considered in this paper. The **dlnm** package contains software for estimating a DLM with linear or nonlinear exposure-response relationships using splines for

the smoothing constraint (Gasparrini, 2011). Mork and Wilson (2022, 2023) compare model performance between the spline-based and tree structured DLM implementations. The **dlnm** package does not consider heterogeneity or multiple exposures. There are several DLM methods for the linear effect of a single exposure with modification by a single covariate. The **dlim** package allows for modification by a single continuous factor (Dematteis et al., 2024) and the **bdlim** package allows for modification by a single categorical factor (Wilson et al., 2017b). However, these packages do not allow for multiple candidate modifying factors or multiple exposures. The package **DiscreteDLM** extends Bayesian estimation to categorical outcomes but does not consider mixtures or heterogeneity (Dempsey and Wyse, 2025). Hence, the **dlmtree** package fills several gaps including heterogeneous effects of multiple candidate modifiers and analysis of mixture or multivariate lagged exposures.

Table 1: Available DLM related R packages with functionalities

Package	GLM	Nonlinearity	Mixture	Heterogeneity
bdlim	✓			✓
DiscreteDLM	✓			
dlim	✓			✓
dlmtree	✓	✓	✓	✓
dlnm	✓	✓		

Note: This table excludes packages designed for autoregressive DLMs, which are used in a different context from the models proposed here.

In this article, we introduce a comprehensive R package **dlmtree** which consolidates a wide range of tree structured DLMs. The package offers a user-friendly and computationally efficient environment to fit tree structured DLMs and to address multiple potential research assumptions including nonlinearity, monotonicity and prior information, multiple simultaneous exposures, and heterogeneous lag effects. We first provide a conceptual review of a regression tree as a smoothing constraint in a DLM framework and an overview of the extensions of tree structured DLMs. We present a decision tree to help users select an appropriate model for their analysis. We illustrate the model fitting process with detailed descriptions and syntax of functions for implementing models, obtaining the summary output and inferential information, and plotting the fitted models for visualization. The package is available in the comprehensive R archive network (CRAN), and the installation instructions are provided at `https://danielmork.github.io/dlmtree/`.

2 Tree structured DLMs

2.1 DLM tree as a smoothing constraint

In this section, we review the regression tree approach to constrained DLM estimation, a key idea underlying the tree structured DLMs in the **dlmtree** package. Classically, the DLM model is applied to time-series studies, and time is defined in terms of the outcome time. Let y_t be the observed outcome and x_t the observed exposure at time t for $t = 1, \dots, T$. For clarity, we assume a continuous outcome when presenting the models. We discuss extensions to generalized linear models in Section 2.4. The DLM applied to time-series is

$$y_t = \sum_{l=1}^{L} x_{t-l}\beta_l + \boldsymbol{z}_t'\boldsymbol{\gamma} + \varepsilon_t, \tag{1}$$

where β_l is the linear effect of exposure at lag l (l days prior to the outcome assessment), $\boldsymbol{z}_t$ is a vector of covariates including the intercept, $\boldsymbol{\gamma}$ is a vector of regression coefficients for the covariates, ε_t is independent error assumed to follow Normal$(0, \sigma^2)$.

In this paper, we focus on an alternative and more general representation of the DLM that does not assume repeated assessments of an outcome in a time-series design. Consider a vector of outcomes $\boldsymbol{y} = (y_1, \dots, y_n)$ for a sample $i = 1, \dots, n$. Suppose we are interested in

the lagged association between the outcome and a single longitudinally assessed exposure $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{iT})$ measured at equally spaced time points, $t = 1, \ldots, T$. Here, exposure time is not explicitly defined relative to the outcome. Most commonly, the exposures are in the T time points prior to outcome assessment, as in (1), but that is not required. In the illustrations in this manuscript, we consider exposure during the first T weeks of pregnancy and a birth or early childhood health outcome as our motivating example. Using this representation, the DLM is

$$y_i = f(\boldsymbol{x}_i) + \boldsymbol{z}_i'\boldsymbol{\gamma} + \varepsilon_i, \quad f(\boldsymbol{x}_i) = \sum_{t=1}^{T} x_{it}\theta_t, \tag{2}$$

where $\boldsymbol{z}_i$, $\boldsymbol{\gamma}$ and ε_i are as defined above, and f is a distributed lag function parameterized by θ_t representing the linear effect of exposure at time t. The time-series design can also be modeled using this format, and we discuss the data processing needed to convert time-series data to this format in Section 4.2.

The estimation of θ_t for $t = 1, \ldots, T$ in (2) requires an appropriate temporal structure, such as a smooth or piecewise-smooth constraint on θ_t, to account for autocorrelation within the exposure measurements. Mork and Wilson (2023) introduced a regression tree-structure, shown in Figure 1, referred to as a *DLM tree*. A DLM tree, denoted $\mathcal{T}$, is a binary tree that

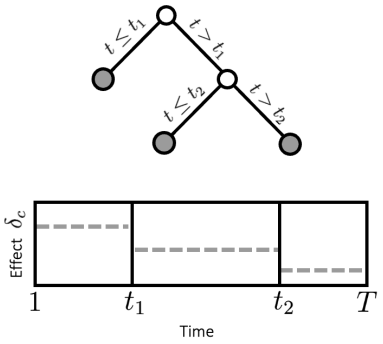


Figure 1: A DLM tree, $\mathcal{T}$. A binary tree splits the time span into non-overlapping intervals, here resulting in three terminal nodes representing three time segments (gray nodes). Each terminal node is assigned a constant effect (gray dashed lines).

splits the exposure time span into non-overlapping segments. We denote the terminal nodes of $\mathcal{T}$ as λ_c for $c = 1, \ldots, C$ where C is the total number of terminal nodes. Each terminal node of the DLM tree is assigned a scalar parameter δ_c that represents the effect of the time lags contained in that terminal node. We denote a set of scalar parameters of the DLM tree $\mathcal{T}$ as $\mathcal{D} = \{\delta_1, \ldots, \delta_C\}$. A DLM tree defines a function of a time lag,

$$g(t|\mathcal{T}, \mathcal{D}) = \delta_c \quad \text{if } t \in \lambda_c. \tag{3}$$

The treed distributed lag model (TDLM) is a tree structured DLM in its simplest form and is a Bayesian additive model that consists of an ensemble of A DLM trees, indexed as $\mathcal{T}_a$, with a corresponding set of scalar parameters $\mathcal{D}_a$ for $a = 1, \ldots, A$. TDLM defines θ_t in (2) as

$$\theta_t = \sum_{a=1}^{A} g(t|\mathcal{T}_a, \mathcal{D}_a). \tag{4}$$

The representation of the DLM tree ensemble provides several advantages. Each DLM tree provides a temporal structure on the exposure-time-response function with data-driven learning of the change points and time spans related to the outcome. The ensemble structure allows for flexibility to approximate smoothness in the exposure-time-response function as each DLM tree in the ensemble splits the time span differently.

2.2 DLM tree pair for lagged multivariate exposures

A distributed lag mixture model extends the DLM framework to a multivariate exposure (also referred to as *mixture exposures* in the environmental literature) assessed longitudinally. For mixture exposures with $M \geq 2$ exposures, we denote the vector of longitudinally assessed measurements of the m^{th} exposure, for $m = 1, \ldots, M$, as $\boldsymbol{x}_{im} = (x_{im1}, \ldots, x_{imT})$. The exposure-time-response function $f(\boldsymbol{x}_i)$ in (2) is replaced with a mixture-exposure-time-response function that incorporates the main effect of each exposure and the pairwise lagged interaction effects between exposures. The mixture-exposure-time-response function is

$$f(\boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{iM}) = \sum_{m=1}^{M} \sum_{t=1}^{T} x_{imt}\theta_{mt} + \sum_{m_1=1}^{M} \sum_{m_2=m_1}^{M} \sum_{t_1=1}^{T} \sum_{t_2=1}^{T} x_{im_1t_1} x_{im_2t_2} \theta_{m_1m_2t_1t_2}, \tag{5}$$

where θ_{mt} is a main effect of exposure m at time t and $\theta_{m_1m_2t_1t_2}$ is an interaction effect of exposure m_1 at time t_1 and exposure m_2 at time t_2.

Estimating the function in (5) is challenging due to autocorrelation across repeated exposure measurements, correlation at the same time lag between exposures, and a high-dimensional parameter space. Mork and Wilson (2023) introduced the treed distributed lag mixture model (TDLMM) that structures the parameters in (5) using a *DLM tree pair*. Figure 2 illustrates a single DLM tree pair, denoted $\{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_1 \times \mathcal{T}_2\}$ with corresponding parameters sets $\{\mathcal{D}_1, \mathcal{D}_2, \Omega\}$, where $\mathcal{D}_1$ and $\mathcal{D}_2$ represent the scalar main effects for $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, and Ω contains the scalar interaction effects for interaction surface, denoted $\mathcal{T}_1 \times \mathcal{T}_2$. Each DLM tree in the pair is associated with one component of the mixture

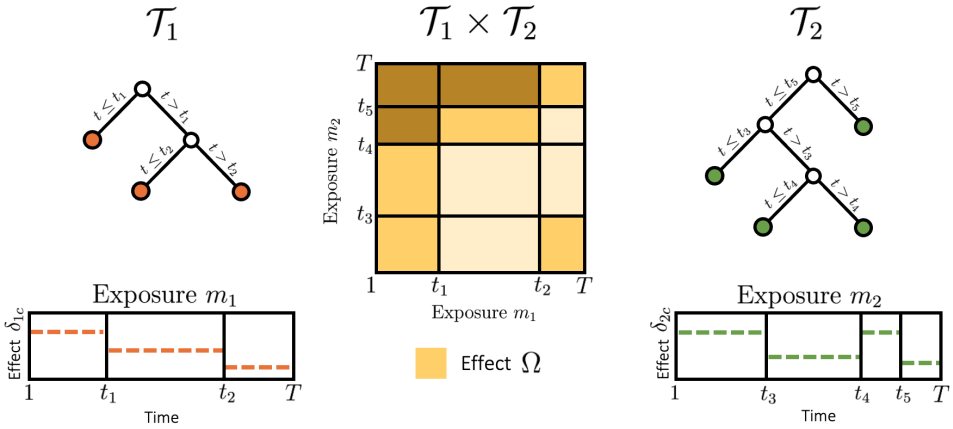


Figure 2: A DLM tree pair. Two DLM trees split the time span of the assigned exposure into non-overlapping intervals, here resulting in three time segments for exposure m_1 and four time segments for exposure m_2 (colored nodes). Each terminal node is assigned a scalar parameter that represents a constant effect of the assigned exposure (colored dashed lines). The interaction surface is fully defined by two DLM trees and each combination of time segments is assigned a scalar parameter (color-shaded boxes).

exposures. Similar to the DLM tree described above, each DLM tree in the pair partitions the time span of its assigned component. Each DLM tree in a tree pair is defined similarly to (3) such that

$$g(t|\mathcal{T}_p, \mathcal{D}_p) = \delta_{pc} \quad \text{if } t \in \lambda_{pc}, \quad p = 1, 2. \tag{6}$$

The DLM tree pair structure allows for an interaction surface to model lagged interactions between exposures. The interaction surface $\mathcal{T}_1 \times \mathcal{T}_2$ shown in the middle of Figure 2, is fully defined by the two DLM trees in a pair. Each time interval of the first DLM tree is paired with every interval of the second DLM tree. Each combination is assigned a scalar parameter $\omega_{c_1c_2}$ that represents the lagged interaction effect where c_1 and c_2 are indices for terminal nodes of the first and second DLM tree, respectively. The interaction surface of a DLM tree

pair defines a function

$$g_I(t_1, t_2|\mathcal{T}_1 \times \mathcal{T}_2, \Omega) = \omega_{c_1c_2} \quad \text{if } t_1 \in \lambda_{c_1}, t_2 \in \lambda_{c_2}. \tag{7}$$

A DLM tree pair with an interaction surface provides a structure to regularize the exposure-time-response function for two components and their lagged interaction effects.

TDLMM employs an ensemble representation of A DLM tree pairs denoted $\{\mathcal{T}_{a1}, \mathcal{T}_{a2}, \mathcal{T}_{a1} \times \mathcal{T}_{a2}\}$ with the corresponding set of scalar parameters $\{\mathcal{D}_{a1}, \mathcal{D}_{a2}, \Omega_a\}$, using the formulation in (6) and (7). Each DLM tree $\mathcal{T}_{ap}$ is also associated with exposure m, indicated by $S_{ap} = m$. With TDLMM, the main effect of exposure m at time t in (5) is

$$\theta_{mt} = \sum_{a=1}^{A} \sum_{p=1}^{2} g(t|\mathcal{T}_{ap}, \mathcal{D}_{ap}) \mathbb{I}(S_{ap} = m). \tag{8}$$

The pairwise interaction effect between exposure m_1 at time t_1 and exposure m_2 at time t_2 in (5) is

$$\theta_{m_1m_2t_1t_2} = \sum_{a=1}^{A} g_I(t_1, t_2|\mathcal{T}_{a1} \times \mathcal{T}_{a2}, \Omega_a) \mathbb{I}(S_{a1} = m_1, S_{a2} = m_2). \tag{9}$$

TDLMM has three representations depending on different assumptions of lagged interactions between exposures. The simplest form, TDLMMadd, assumes $\theta_{m_1m_2t_1t_2} = 0$ in (5), implying no interaction between exposures, resulting in an additive model of the main effects of exposures. The second form, TDLMMns, accounts for lagged interaction between exposures but not within exposures. The last, TDLMMall, allows for all lagged interactions between and within exposures, implying a nonlinear effect of exposure.

2.3 Extensions to nonlinear exposure-time-response functions

Mork and Wilson (2022) introduced the treed distributed lag nonlinear model (TDLNM) to estimate the nonlinear association between a single longitudinally assessed exposure and an outcome. The DLM tree, as shown in (3), assumes a linear association between exposure at each time point and the outcome. To relax the linearity assumption for a distributed lag nonlinear model framework, TDLNM modifies the DLM tree to additionally split exposure concentration levels along with the exposure time span, partitioning the bi-dimensional space of exposure concentration and time lags. Mork and Wilson (2024) further extended TDLNM to include a monotonicity assumption, where the exposure-response is constrained to be non-decreasing at each time point. See Mork and Wilson (2022, 2024) for details.

2.4 Extensions to generalized linear models

In various applications of DLMs, the response variables may be binary or counts. Examples of binary outcomes include the occurrence of conditions such as asthma or preterm birth, and an example of a count-valued outcome is the daily number of deaths in a county. The linear representation of the DLM framework allows the tree structured DLMs to extend to the generalized linear model setting. TDLM, TDLNM, and TDLMM have been extended to incorporate binary response variables via logistic regression (Mork and Wilson, 2022, 2023). Further extensions on these models allow for count data via negative binomial regression, including an option for zero-inflated negative binomial data. The extensions to binary and count data rely on a framework based on the Pólya-Gamma data augmentation approach (Polson et al., 2013; Neelon, 2019).

2.5 Extensions to heterogeneous models

Another extension to the DLM framework is to assume heterogeneous exposure effects. The exposure effects may be heterogeneous due to a single modifying factor or a set of factors. For example, the impact of prenatal exposure to air pollution may be governed by genetic

factors such as fetal sex (Rosa et al., 2019). The set of factors, referred to as *modifiers*, may be continuous, categorical, or ordinal. The general approach is to introduce an additional tree, known as a *modifier tree*, that partitions the modifier space and has a DLM tree or a DLM tree pair affixed to each terminal node. Mork et al. (2024) extended TDLM to the

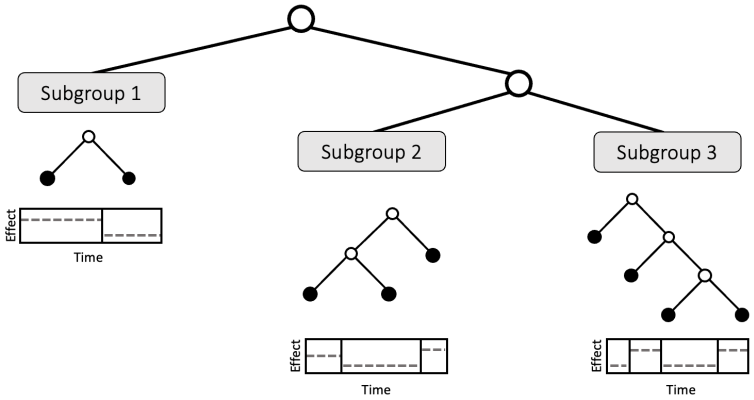


Figure 3: A nested tree structure for heterogeneous tree structured DLMs. The top tree is a modifier tree that is applied to candidate modifiers, here resulting in three subgroups. DLM trees are affixed to the terminal nodes of a modifier tree to estimate the exposure-time-response relationship specific to the subgroups.

heterogeneous distributed lag model (HDLM) by introducing a nested tree structure, as shown in Figure 3. In a nested tree, a modifier tree is applied to a set of candidate modifiers, defining mutually exclusive subgroups of the sample at each terminal node. A DLM tree is then attached to each terminal node of the modifier tree to define subgroup-specific effects with unique parameters for each subgroup. Other extensions include a heterogeneous distributed lag mixture model (HDLMM) that extends TDLMM to incorporate heterogeneity based on a set of multiple candidate modifiers using an ensemble of tree triplet structures.

3 Implementation

The tree structured DLMs are classified with three main criteria: 1) linear or nonlinear exposure-time-response function, 2) one lagged component or a mixture of more than one lagged component, and 3) homogeneous or heterogeneous exposure-time-response relationship. Figure 4 illustrates a decision tree as a guide to choosing an appropriate tree structured DLM.

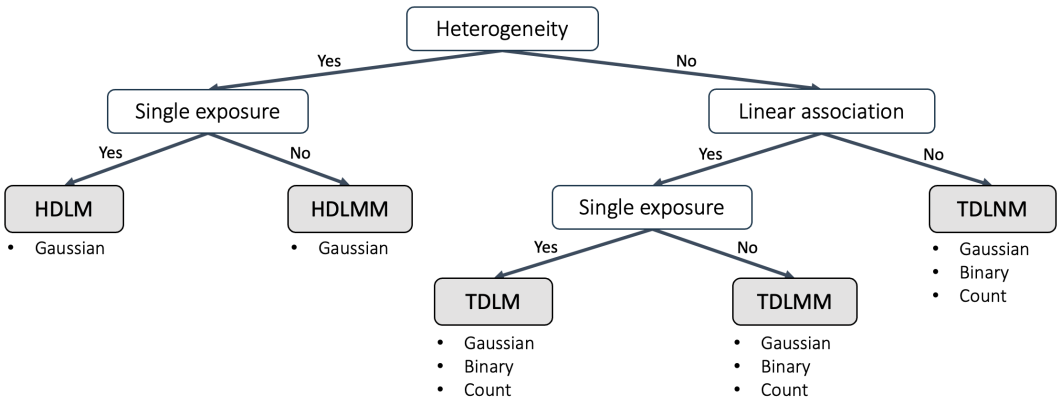


Figure 4: A decision tree for choosing tree structured DLMs. The bullet points below the models list the data types of response variables that each model can incorporate.

The **dlmtree** package offers tree structured DLMs shown in Figure 4. A main function `dlmtree` is designed to fit various tree structured DLMs, offering customizable analysis

through its arguments. The function `dlmtree` with its main arguments is as follows.

```
dlmtree(formula, data, exposure.data, family, dlm.type, mixture, het)
```

The function requires a formula specifying an outcome and covariates for the fixed effect, a data source, the data type of response variable, and three arguments for model specification. The data for the `formula`, including covariates and the outcome, are provided as an $(n \times p)$ data frame in the `data` argument. The exposures are not specified in the `formula` and are not needed in the `data` argument. Rather, the exposure data is specified separately as an $(n \times T)$ matrix of exposure measurements or a list of $(n \times T)$ matrices of exposure measurements in the `exposure.data` argument.

Table 2: Arguments for `dlmtree` function

Argument	Description
`formula`	Object of class `formula` for the fixed effect
`data`	A data frame containing covariates and an outcome used in `formula`
`exposure.data`	A numerical matrix of exposure data with the same length as data. For a mixture setting, a named list containing equally sized numerical matrices of exposure data having the same length as the data
`family`	'gaussian' for a continuous response, 'logit' for binomial, 'zinb' for count data
`dlm.type`	DLM type specification: 'linear', 'nonlinear', 'monotone'
`mixture`	logical; A flag for mixture exposures if `TRUE`
`het`	logical; A flag for heterogeneity if `TRUE`

Table 2 provides descriptions of the arguments. All tree structured DLMs allow for continuous response variables, while TDLM, TDLNM, and TDLMM additionally allow for binary and count-valued response variables. Additional parameters, omitted here, include MCMC sampling parameters, the number of trees in the ensemble, effect shrinkage, sparsity parameters for exposure and modifier selection, and model-specific hyperparameters. These parameters can be fine-tuned and passed as a list to the corresponding `control` helper functions, each suffixed with its purpose (e.g., `control.mcmc`, `control.hyper`).

A fitted model is assigned a class, determined by the model-specifying arguments. The classes are: `tdlm`, `tdlmm`, `tdlnm`, `hdlm`, and `hdlmm`. Each class uses an S3 object oriented system with `summary` method. The `summary` method returns the model information, estimates and credible intervals for the fixed effects and lag effects, and time points of a significant effect, where the 95% credible intervals of the lag effects do not contain zero (in some applications this is referred to as *critical window*) of the exposure or mixture exposures. The `plot` method on the summary object further returns the visualization of the estimated main exposure effects (and interaction effects if applicable) with 95% credible intervals. Additionally for classes of models with heterogeneity: `hdlm` and `hdlmm`, the `shiny` method is built for various types of statistical inference. The **shiny** app interface provides tools for identifying important modifiers and their splitting points that contribute to heterogeneity. It also includes features for evaluating personalized exposure effects with a set of user-specified modifiers, and exposure effects specific to a subgroup defined by a set of modifiers of interest.

4 Example usage

We illustrate the example usage of tree structured DLMs through a set of vignettes based on simulated data. We demonstrate data preparation and the model fitting process for TDLM, TDLMM, HDLM, and HDLMM. An example of fitting TDLNM is provided in the supplementary material.

4.1 Simulated dataset

We use the simulated dataset 'sbd_dlmtree', which is publicly available in the **dlmtree** package GitHub repository. The dataset contains 10,000 simulated mother-child dyads with descriptions of maternal and birth information. The maternal covariates include maternal age, height, prior weight, prior body mass index (BMI), race, Hispanic designation, education attainment, smoking habits, marital status, and yearly income. Additionally, the birth information includes birth weight for gestational age z-scores (BWGAZ), gestational age, sex of a child, and estimated date of conception. The dataset contains five environmental chemicals measured at 37 weeks preceding the birth for each dyad: fine particulate matter ($PM_{2.5}$), temperature, sulfur dioxide (SO_2), carbon monoxide (CO), and nitrogen dioxide (NO_2). Each exposure measurement is scaled by its interquartile range value. The dataset is constructed to have realistic distributions and correlations of the covariates and exposures. It contains a complex exposure-response relationship that includes interactions and heterogeneous effects.

In the following examples, we examine the distributed lag effects of maternal exposure to the environmental mixtures during the 37 weeks of gestation on BWGAZ. The results provided in the example usage are solely for demonstrative purposes of the model fitting process using the model parameters and simulated data and do not represent any actual findings.

4.2 Data preparation for model fitting

We first load the required packages: **dlmtree** and **dplyr**. We used **dplyr** for a better presentation of the data processing. We set the seed for reproducibility and load the external dataset 'sbd_dlmtree' from **dlmtree** package repository using an embedded function `get_sbd_dlmtree`.

```
# Libraries and seed
library(dlmtree)
library(dplyr)
set.seed(1)

# Download data as 'sbd'
sbd <- get_sbd_dlmtree()
```

The data frame `sbd` has a lagged format where each row has columns of lagged measurements of each exposure (e.g. wide format data). Often, datasets have a time-series format, which contains a single column of dates and multiple columns of corresponding measurements of response variables and exposures on that specific date only. This is particularly common in time-series studies, whereas the wide format data is more common in cohort studies. The model fitting function `dlmtree` is designed to use a data frame in a wide format; hence, a data frame with a time-series format must be pivoted to a wide format. An example data frame of time-series format and a function for data pivoting are provided in the supplementary materials. In addition to the wide format, the software in this package requires that all individuals and all exposures (for multi-exposure models) have the same number of exposure time points.

To prepare the birth data in `sbd` for model fitting, we create a data frame including the covariates and the response variable, BWGAZ. We note that the categorical columns in the dataset are of class `factor`. We consider five components for exposure data: $PM_{2.5}$, temperature, SO_2, CO, and NO_2, and store them as a list of exposure matrices with a size of $(10{,}000 \times 37)$. Each matrix is already in wide format, where rows represent observations and columns represent exposure measurements at different lags.

```
# Response and covariates
sbd_cov <- sbd %>% select(bwgaz, ChildSex, MomAge, GestAge, MomPriorBMI, Race,
```

```
                          Hispanic, MomEdu, SmkAny, Marital, Income,
                          EstDateConcept, EstMonthConcept, EstYearConcept)

# Exposure data
sbd_exp <- list(PM25 = sbd %>% select(starts_with("pm25_")),
                TEMP = sbd %>% select(starts_with("temp_")),
                SO2 = sbd %>% select(starts_with("so2_")),
                CO = sbd %>% select(starts_with("co_")),
                NO2 = sbd %>% select(starts_with("no2_")))

sbd_exp <- sbd_exp %>% lapply(as.matrix)
```

Each matrix of the exposure data list can be centered and scaled with caution to different interpretations of the resulting estimates. Specifically, when using the TDLMM with lagged interaction, it is crucial to avoid centering the exposure data as it can lead to an inaccurate estimate of the marginal exposure effect when considering co-exposures.

4.3 TDLM: Estimating linear relationship between an outcome and a single exposure

Model fitting and summary

We first assume that we are interested in the linear association between BWGAZ and weekly exposure to $PM_{2.5}$ during the first 37 gestational weeks. We include the following covariates to control for fixed effects: child sex, maternal age, BMI, race, Hispanic designation, smoking habits, and month of conception. We fit TDLM with the following code.

```
tdlm.fit <- dlmtree(
    formula = bwgaz ~ ChildSex + MomAge + MomPriorBMI + Race +
                        Hispanic + SmkAny + EstMonthConcept,
    data = sbd_cov,
    exposure.data = sbd_exp[["PM25"]], # A single numeric matrix
    family = "gaussian",
    dlm.type = "linear",
    control.mcmc = list(n.burn = 2500, n.iter = 10000, n.thin = 5)
)
```

The resulting fitted object of class `tdlm` has attributes of the fitted model information and posterior samples of parameters of interest. The `summary` method applied to the object `tdlm.fit` returns a clear overview of the model fit. The summary of the model is obtained with the following code.

```
tdlm.sum <- summary(tdlm.fit)
print(tdlm.sum)

---
TDLM summary

Model run info:
- bwgaz ~ ChildSex + MomAge + MomPriorBMI + Race + Hispanic + SmkAny + EstMonthConcept
- sample size: 10,000
- family: gaussian
- 20 trees
- 2500 burn-in iterations
- 10000 post-burn iterations
- 5 thinning factor
- exposure measured at 37 time points
```

```
- 0.95 confidence level

Fixed effect coefficients:
                       Mean  Lower  Upper
*(Intercept)          2.289  2.032  2.542
*ChildSexM           -2.105 -2.126 -2.085
MomAge                0.000 -0.001  0.002
*MomPriorBMI         -0.021 -0.022 -0.019
RaceAsianPI           0.069 -0.057  0.192
RaceBlack             0.078 -0.050  0.205
Racewhite             0.059 -0.060  0.181
*HispanicNonHispanic  0.255  0.233  0.278
*SmkAnyY             -0.403 -0.451 -0.356
EstMonthConcept2     -0.049 -0.109  0.010
*EstMonthConcept3    -0.145 -0.211 -0.077
*EstMonthConcept4    -0.230 -0.295 -0.160
*EstMonthConcept5    -0.207 -0.265 -0.147
*EstMonthConcept6    -0.205 -0.260 -0.153
EstMonthConcept7     -0.032 -0.083  0.023
*EstMonthConcept8     0.145  0.081  0.210
*EstMonthConcept9     0.393  0.326  0.460
*EstMonthConcept10    0.372  0.311  0.437
*EstMonthConcept11    0.330  0.271  0.387
*EstMonthConcept12    0.129  0.078  0.181
---
* = CI does not contain zero

DLM effect:
range = [-0.019, 0.008]
signal-to-noise = 0.021
critical windows: 11-20,36-37
             Mean  Lower  Upper
Period 1    0.003 -0.005  0.015
Period 2    0.000 -0.006  0.010
Period 3   -0.002 -0.010  0.004
Period 4   -0.002 -0.010  0.003
Period 5   -0.001 -0.007  0.004
Period 6   -0.001 -0.006  0.005
Period 7   -0.001 -0.006  0.006
Period 8   -0.001 -0.008  0.004
Period 9   -0.002 -0.011  0.003
Period 10  -0.002 -0.012  0.006
*Period 11 -0.016 -0.024 -0.007
*Period 12 -0.017 -0.024 -0.010
*Period 13 -0.017 -0.024 -0.012
*Period 14 -0.017 -0.022 -0.010
*Period 15 -0.017 -0.022 -0.011
*Period 16 -0.017 -0.022 -0.011
*Period 17 -0.017 -0.024 -0.011
*Period 18 -0.019 -0.030 -0.013
*Period 19 -0.018 -0.027 -0.011
*Period 20 -0.015 -0.024 -0.003
Period 21  -0.007 -0.019  0.002
Period 22  -0.002 -0.010  0.006
Period 23  -0.003 -0.012  0.003
Period 24  -0.002 -0.008  0.004
```

```
Period 25   0.000 -0.005  0.006
Period 26   0.000 -0.005  0.006
Period 27  -0.001 -0.005  0.005
Period 28  -0.001 -0.006  0.004
Period 29  -0.001 -0.007  0.004
Period 30  -0.002 -0.008  0.003
Period 31  -0.002 -0.008  0.004
Period 32  -0.001 -0.008  0.005
Period 33   0.002 -0.004  0.010
Period 34   0.004 -0.003  0.012
Period 35   0.006 -0.001  0.014
*Period 36  0.008  0.000  0.017
*Period 37  0.008  0.000  0.019
---
* = CI does not contain zero

residual standard errors: 0.004
---
```

The summary output first presents a section 'Model run info' with the model fitting information including the formula, sample size, data type of the response variable, number of trees in the ensemble, MCMC parameters, number of lags, and a confidence level. The next section, 'Fixed effect coefficients', shows the estimates with credible intervals for the regression coefficients of the covariates. Lastly, the summary output returns 'DLM effect' section including the range of DLM effects, signal-to-noise ratio, and estimated lagged effects with credible intervals. Each lag is marked with an asterisk if it is identified as a critical window based on a pointwise 0.95 probability credible interval. In context, TDLM estimated a negative association between BWGAZ and $PM_{2.5}$ with gestational weeks 11–20 as critical windows.

The cumulative effect of the exposure is defined as the effect of a one unit increment of the exposure across all time points. The following code returns the estimated cumulative effect with its 95% credible interval from an attribute `cumulative.effect` of the summary object `tdlm.sum`.

```
tdlm.sum$cumulative.effect
      mean       2.5%      97.5%
-0.1738753 -0.2124918 -0.1367665
```

In this context, TDLM estimates that a unit increment of exposure to $PM_{2.5}$ across all gestational weeks has a cumulative exposure effect of -0.17 (95% CrI: [-0.21, -0.14]) on BWGAZ.

Visualizing exposure effects

For a more intuitive view of the overall trend of the distributed lag effects, the `plot` method may be applied to the summary object `tdlm.sum` to visualize the effect estimates as the following.

```
plot(tdlm.sum, main = "Estimated effect of PM2.5", xlab = "Time", ylab = "Effect")
```

Figure 5 shows the plot of the estimated exposure effect of $PM_{2.5}$ in the simulated dataset where the x-axis is the lags (or weeks) and the y-axis is the estimated exposure effect. The gray area showing the 95% credible intervals of exposure effects during weeks 11–20 does not cover the red line of a null effect, which indicates a critical window. The `plot` method also includes additional arguments of `main`, `xlab`, and `ylab` for customizing the main title, x-axis, and y-axis label. For a customized plot, the estimated values can be obtained from the attributes of the summary object `tdlm.sum`: `matfit`, `cilower`, and `ciupper`.

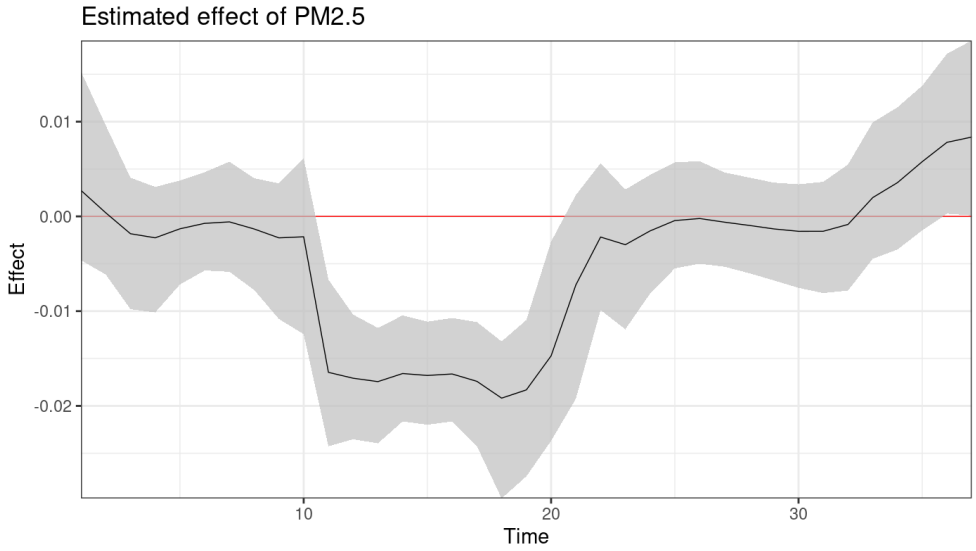


Figure 5: Estimated distributed lag effects of $PM_{2.5}$ on BWGAZ during 37 gestational weeks, using TDLM. These results are based on simulated data.

4.4 TDLMM: Analyzing linear relationship between an outcome and multiple exposures

Model fitting with pairwise lagged interactions

Suppose we are interested in the linear association between BWGAZ and mixture exposures of five exposures: $PM_{2.5}$, temperature, SO_2, CO, and NO_2. We are mainly interested in the marginal effects of each exposure on BWGAZ, the pairwise lagged interactions between exposures, and which exposures are most correlated with BWGAZ. Here, we use all five exposures to illustrate the flexibility and capability of our package to handle multiple exposures. In practice, users should carefully determine the number of exposures included in the model by considering the complexity of the model relative to the number of observations and the expected signal-to-noise ratio. The key differences between the code for the model for multiple exposures below and the code for the previous single exposure model are that the `exposure.data` is provided with a list of matrices of five different exposures and we specify `mixture = TRUE`. The code is as follows.

```
tdlmm.fit <- dlmtree(
    formula = bwgaz ~ ChildSex + MomAge + MomPriorBMI +
                        Race + Hispanic + SmkAny + EstMonthConcept,
    data = sbd_cov,
    exposure.data = sbd_exp,
    family = "gaussian",
    dlm.type = "linear",
    mixture = TRUE,
    control.mix = list(interactions = "noself"),
    control.mcmc = list(n.burn = 2500, n.iter = 10000, n.thin = 5)
)
```

The model assumption regarding lagged interaction can be additionally specified for the TDLMM fitting with `interactions` argument within `control.mix`. The options include no interaction (‘`none`’), no-self interactions (‘`noself`’), and all interactions (‘`all`’). We use TDLMM with no-self lagged interaction, which is the default argument.

Model summary accounting for co-exposures

The summary method can be applied to the fitted object tdlmm.fit. As TDLMM allows for pairwise lagged interaction between mixture exposures, the estimated exposure effect for each exposure varies with the levels of the co-exposures. The summary method on class tdlmm offers additional control argument marginalize to address this. The argument marginalize requires a fixed level used for co-exposure marginalization. The default is the empirical means of co-exposures, which provides the distributed lag function for a single exposure estimated when all other exposures are fixed at their means. This is equivalent to integrating out all co-exposures. The following code returns the summary of the tdlmm.fit with marginalization using the empirical means of co-exposures.

```
# Marginalization with co-exposure fixed at the empirical means
tdlmm.sum <- summary(tdlmm.fit, marginalize = "mean")
```

Other marginalization options are available for conducting different types of inferences. The second option is to specify a number between 0 and 100, representing a percentile of the co-exposures that will be used for marginalization. The following code returns the summary of the tdlmm.fit with marginalization fixing all co-exposures at their 25th percentile values.

```
# Marginalization with co-exposure fixed at 25th percentile
tdlmm.sum.percentile <- summary(tdlmm.fit, marginalize = 25)
```

The last option is to specify the exact levels of co-exposures. This option requires the argument marginalize to be a numeric vector of the same length as the number of exposures used for the model fitting. The specified values in the vector must also follow the order of the exposures in the fitted model. This method of marginalization offers flexibility as these exposure levels can be specified based on pre-informed levels using existing data or to address hypothetical questions. For example, the following code can be used to obtain the marginal exposure effect of $PM_{2.5}$ when temperature, SO_2, CO, and NO_2 are all fixed to 1. The marginalized effects of other exposures are calculated in a similar manner.

```
# Marginalization with co-exposure fixed at exact levels for each exposure
tdlmm.sum.level <- summary(tdlmm.fit, marginalize = c(1, 1, 1, 1, 1))
```

Below is the summary result of tdlmm.sum with the default argument using empirical means.

```
print(tdlmm.sum)
---
TDLMM summary

Model run info:
- bwgaz ~ ChildSex + MomAge + MomPriorBMI + Race + Hispanic + SmkAny + EstMonthConcept
- sample size: 10,000
- family: gaussian
- 20 trees (alpha = 0.95, beta = 2)
- 2500 burn-in iterations
- 10000 post-burn iterations
- 5 thinning factor
- 5 exposures measured at 37 time points
- 10 two-way interactions (no-self interactions)
- 1 kappa sparsity prior
- 0.95 confidence level

Fixed effects:
                      Mean  Lower  Upper
*(Intercept)         0.172  0.043  0.307
*ChildSexM          -2.063 -2.085 -2.041
```

```
 MomAge               0.001 -0.001  0.002
*MomPriorBMI         -0.020 -0.022 -0.019
 RaceAsianPI          0.027 -0.058  0.117
 RaceBlack            0.033 -0.063  0.124
 Racewhite            0.016 -0.067  0.100
*HispanicNonHispanic  0.248  0.224  0.272
*SmkAnyY             -0.393 -0.441 -0.346
 EstMonthConcept2     0.073 -0.003  0.145
*EstMonthConcept3     0.107  0.009  0.211
*EstMonthConcept4     0.158  0.038  0.282
*EstMonthConcept5     0.255  0.126  0.388
*EstMonthConcept6     0.200  0.064  0.333
*EstMonthConcept7     0.223  0.084  0.354
*EstMonthConcept8     0.199  0.068  0.331
*EstMonthConcept9     0.291  0.164  0.418
*EstMonthConcept10    0.182  0.070  0.296
*EstMonthConcept11    0.135  0.040  0.236
 EstMonthConcept12    0.006 -0.062  0.077
---
* = CI does not contain zero

--
Exposure effects: critical windows
* = Exposure selected by Bayes Factor
(x.xx) = Relative effect size

 *PM25 (0.7): 11-20
 *TEMP (0.7): 5-19
 *SO2 (0.21):
 *CO (0.63):
 *NO2 (0.26): 23
--
Interaction effects: critical windows

 PM25/TEMP (0.8):
 12/6-19
 13/6-19
 14/6-20
 15/6-20
 16/6-20
 17/6-21
 18/5-22
 19/5-22
 20/6-21
---
residual standard errors: 0.005
```

The summary output of the TDLMM fit contains similar information to that of the TDLM. The 'Model run info' section in the output includes additional information specific to mixture exposures: the number of exposures included in the model, the number of pairwise interactions, and a sparsity parameter for exposure selection. The summary output does not include the lagged effects for each exposure to prevent overwhelming the output with excessive information. The summary presents the critical window of the marginal effects of each exposure with the relative effect size, indicating the effect size of exposure relative to that of other exposures. The summary also returns the critical window of lagged interaction effects with relative effect size. In our context, all five exposures are considered to be

significantly associated with BWGAZ, based on a Bayes factor threshold of 0.5. The TDLMM estimated gestational weeks 11–20, 5–19, and 23 as critical windows of $PM_{2.5}$, temperature, and NO_2, respectively. The model fit also identified significant $PM_{2.5}$–temperature lagged interaction effects.

Additional statistical inferences using TDLMM

More useful statistical inferences are possible with `tdlmm.fit` and `tdlmm.sum`. First, a function `adj_coexposure` can be used to obtain the marginalized exposure effect while accounting for the expected change in co-exposures. In comparison to using the argument `marginalize` in `summary` method, the function `adj_coexposure` uses a spline-based method to predict the expected changes in co-exposures corresponding with a pre-defined change in an exposure of interest and calculates the marginalized effects of the primary exposure with co-exposure at the predicted levels. The following code shows an example usage of the function with its output omitted.

```
# Lower and upper exposure levels specified as 25th and 75th percentiles
tdlmm.coexp <- adj_coexposure(sbd_exp, tdlmm.fit, contrast_perc = c(0.25, 0.75))
```

The function requires exposure data, the model fit of class `tdlmm`, and an argument `contrast_perc` which can be specified with a vector of two percentiles used for co-exposure prediction. Another argument `contrast_exp` is available for specifying exact exposure levels for each exposure.

Second, the marginal cumulative effect of each exposure can be obtained with the summary object `tdlmm.sum`. The object has a list attribute `DLM`, which contains estimates of marginal exposure effect and cumulative effect with their credible intervals. These estimated effects correspond to the argument `marginalize` specified within `summary` method. The code below returns the estimates of the cumulative effect of $PM_{2.5}$.

```
tdlmm.sum$DLM$PM25$cumulative
$mean
[1] -0.3790024

$ci.lower
      2.5%
-0.5318544

$ci.upper
     97.5%
-0.2305391
```

$PM_{2.5}$ can be replaced with other exposures if desired, e.g., `tdlmm.sum$DLM$TEMP$cumulative` for temperature. In this context, the TDLMM estimates that the marginal cumulative effect of a unit increase of $PM_{2.5}$ across all gestational weeks is -0.38 (95% CrI: [-0.53, -0.23]).

Visualizing main exposure effects and lagged interaction effects

The `plot` method on the summary object `tdlmm.sum` requires a single exposure or a pair of exposures. The `plot` method returns the marginal effect of an exposure when specified with a single exposure. For instance, for the three exposures, $PM_{2.5}$, temperature, and NO_2, we can use the following code.

```
library(gridExtra)

p1 <- plot(tdlmm.sum, exposure1 = "PM25", main = "PM2.5")
```

```
p2 <- plot(tdlmm.sum, exposure1 = "TEMP", main = "Temperature")
p3 <- plot(tdlmm.sum, exposure1 = "NO2", main = "NO2")

grid.arrange(p1, p2, p3, nrow = 1)
```

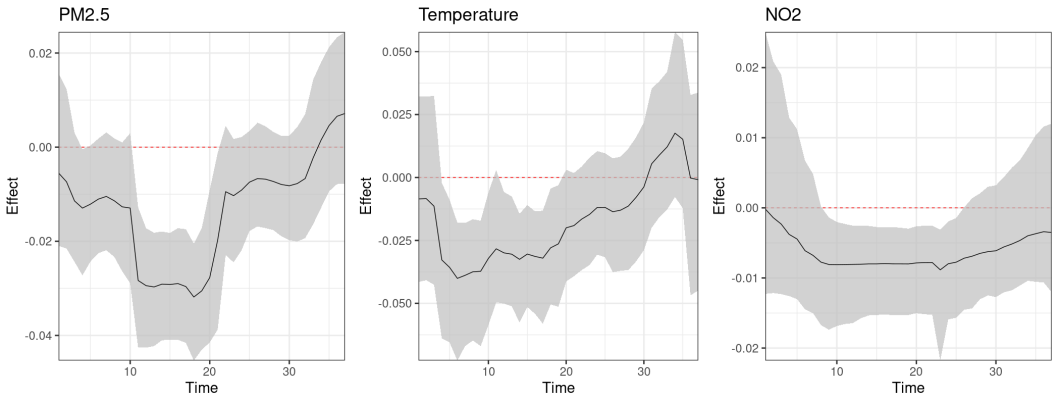


Figure 6: Estimated marginal distributed lag effects of $PM_{2.5}$, temperature, and NO_2 on BWGAZ during 37 gestational weeks, using TDLMM. These results are based on simulated data.

The `plot` method with arguments of two exposures visualizes an interaction surface of two specified exposures. The following code plots an estimated pairwise interaction surface of two specified exposures:

```
plot(tdlmm.sum, exposure1 = "PM25", exposure2 = "TEMP")
```

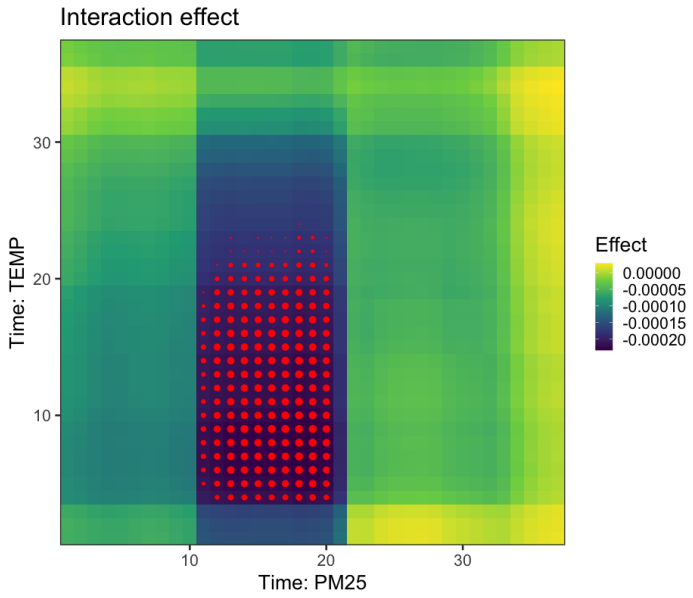


Figure 7: Estimated lagged interaction effects between $PM_{2.5}$ and temperature, using TDLMM. These results are based on simulated data.

Figure 7 shows the estimated interaction surface between $PM_{2.5}$ and temperature in the simulated data, estimated with TDLMM. The gradient colors on the grid indicate the estimated interaction effects where the x-axis is the exposure time span of $PM_{2.5}$ and the y-axis is that of temperature. The red dots indicate that the credible interval of the effect does not contain zero, with larger dots indicating a higher probability of non-null effect. Figure 7 indicates significant negative interaction effects at weeks 12–20 of $PM_{2.5}$ and 6–19 weeks of temperature at 95% confidence level, implying that increased exposure to $PM_{2.5}$ may decrease the exposure effect of temperature, and vice versa.

4.5 HDLM & HDLMM: Introducing heterogeneity to distributed lag effects

We illustrate heterogeneous models for a single exposure (HDLM) and for a mixture exposure (HDLMM) for estimating heterogeneous exposure effects. We focus our demonstration on a **shiny** interface built for HDLM and HDLMM for examining the most significant modifying factors, the personalized exposure effects, and the subgroup-specific exposure effects.

Model fitting and summary with selected modifiers

Suppose we are interested in the linear association between BWGAZ and a single exposure $PM_{2.5}$. We additionally assume that the exposure effect of $PM_{2.5}$ may be modified by child sex, maternal age, BMI, and smoking habits across the population. Specifically, to fit heterogeneous models, additional arguments can be passed to `control.het` as a list. An argument `modifiers` can be set to a vector of modifier names. These modifiers must be included in the data frame provided to the `data` argument (`sbd_cov` in this example). By default, the argument is set to include all covariates included in the `formula` argument as modifiers. Also, the possible number of splitting points of modifiers for heterogeneity can be specified with an integer argument `modifier.splits` to manage the computational cost. We specify `het = TRUE` to fit the HDLM with the following code.

```
hdlm.fit <- dlmtree(
    formula = bwgaz ~ ChildSex + MomAge + MomPriorBMI +
                        Race + Hispanic + SmkAny + EstMonthConcept,
    data = sbd_cov,
    exposure.data = sbd_exp[["PM25"]],
    family = "gaussian",
    dlm.type = "linear",
    het = TRUE,
    control.het = list(
        modifiers = c("ChildSex", "MomAge", "MomPriorBMI", "SmkAny"),
        modifier.splits = 10
    ),
    control.mcmc = list(n.burn = 2500, n.iter = 10000, n.thin = 5)
)
```

The `summary` method applied to the object `hdlm.fit` returns the overview of the model fit. The following code returns the summary.

```
hdlm.sum <- summary(hdlm.fit)
print(hdlm.sum)

---
HDLM summary

Model run info:
- bwgaz ~ ChildSex + MomAge + MomPriorBMI + Race + Hispanic + SmkAny + EstMonthConcept
- sample size: 10,000
- family: gaussian
- 20 trees
- 2500 burn-in iterations
- 10000 post-burn iterations
- 5 thinning factor
- exposure measured at 37 time points
- 0.5 modifier sparsity prior
- 0.95 confidence level
```

```
Fixed effects:
                       Mean  Lower  Upper
*(Intercept)          1.272  0.914  1.629
 ChildSexM            0.127 -0.302  0.566
 MomAge               0.001 -0.002  0.004
*MomPriorBMI         -0.021 -0.024 -0.018
 RaceAsianPI          0.045 -0.074  0.171
 RaceBlack            0.055 -0.070  0.182
 Racewhite            0.034 -0.082  0.157
*HispanicNonHispanic  0.255  0.233  0.278
*SmkAnyY             -0.406 -0.453 -0.359
 EstMonthConcept2    -0.050 -0.104  0.004
*EstMonthConcept3    -0.129 -0.188 -0.070
*EstMonthConcept4    -0.201 -0.265 -0.139
*EstMonthConcept5    -0.195 -0.246 -0.144
*EstMonthConcept6    -0.199 -0.249 -0.144
 EstMonthConcept7    -0.035 -0.087  0.019
*EstMonthConcept8     0.147  0.088  0.208
*EstMonthConcept9     0.395  0.331  0.455
*EstMonthConcept10    0.388  0.328  0.446
*EstMonthConcept11    0.343  0.290  0.397
*EstMonthConcept12    0.141  0.091  0.190
---
* = CI does not contain zero

Modifiers:
              PIP
ChildSex    1.0000
MomAge      0.6305
MomPriorBMI 0.9055
SmkAny      0.0975
---
PIP = Posterior inclusion probability

residual standard errors: 0.004
---
To obtain exposure effect estimates, use the 'shiny(fit)' function.
```

As before, the summary output includes '`Model run info`' and '`Fixed effects`' sections with the estimates and the 95% credible intervals of the regression coefficients of the fixed effect. The summary additionally shows the sparsity hyperparameter set for modifier selection and the posterior inclusion probability (PIP) of the modifiers included in the model. The fitted HDLM identified child sex, maternal age, and BMI to be the modifiers that contributed the most heterogeneity to the exposure effect of $PM_{2.5}$. It is important to note that a high PIP does not necessarily indicate modification. Instead, it suggests that their posterior should be examined to determine if any meaningful modification is present.

A similar model fitting process can be done when examining the heterogeneous exposure effect of a mixture of five exposures on BWGAZ. The following code additionally specifies `mixture = TRUE` and fits HDLMM with the same potential modifiers:

```
hdlmm.fit <- dlmtree(
    formula = bwgaz ~ ChildSex + MomAge + MomPriorBMI +
                      Race + Hispanic + SmkAny + EstMonthConcept,
    data = sbd_cov,
    exposure.data = sbd_exp,
```

```
    family = "gaussian",
    dlm.type = "linear",
    mixture = TRUE,
    het = TRUE,
    control.het = list(
        modifiers = c("ChildSex", "MomAge", "MomPriorBMI", "SmkAny"),
        modifier.splits = 10),
    ),
    control.mcmc = list(n.burn = 2500, n.iter = 10000, n.thin = 5)
)
```

As previously, the `summary` method on `hdlmm` model object similarly returns the summary of the fitted HDLMM. The summary output, omitted here, is similar to that of HDLM, with additional information such as the number of exposures, number of pairwise interactions, and sparsity parameters for modifier selection and exposure selection. Unlike the `summary` method applied to the model of class `tdlmm` in Section 4.4, marginalization methods are unavailable for the fitted model `hdlmm.fit` as the marginalization of co-exposure with heterogeneity is not well defined.

Using shiny app to investigate heterogeneous distributed lag effects

Exposure effects are estimated at the individual level and can be summarized at either the individual or subgroup level. This flexibility makes summarizing and visualizing the estimated effects challenging. A built-in **shiny** app with an object of class `hdlm` and `hdlmm` provides a comprehensive analysis of the exposure effects. HDLM and HDLMM share the same **shiny** interface but the `shiny` method applied to class `hdlmm` additionally includes an option in the panel to select an exposure of interest from mixture exposures. We present the **shiny** app interface using the fitted HDLM for a single exposure, using the same argument specification for fitting the model `hdlmm.fit`. The **shiny** app is launched with the following code.

```
shiny(hdlm.fit)
```

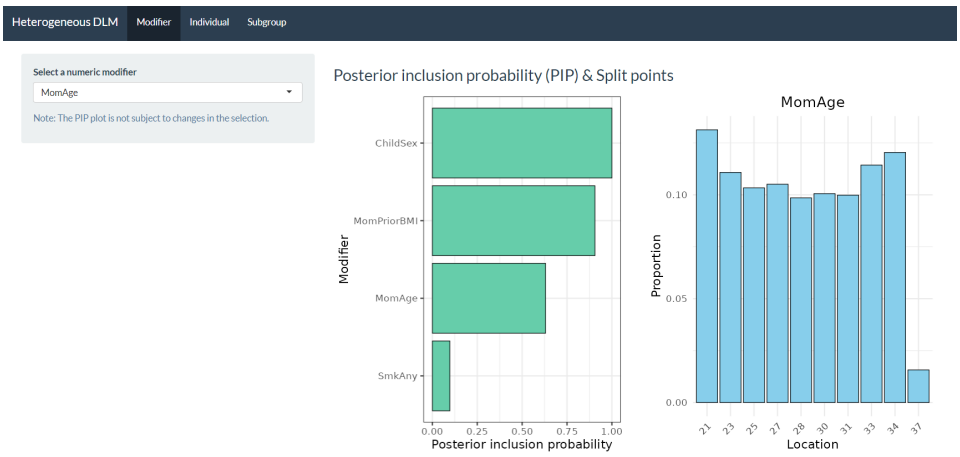


Figure 8: Example of R **shiny** user interface for the fitted HDLM. The displayed results are based on simulated data.

Figure 8 shows the main screen, also the first tab, of the **shiny** app for the fitted HDLM. The **shiny** interface includes three tabs. The first tab labeled 'Modifier' presents two panels including a bar plot of modifier PIPs and the proportions of split points of a user-selected continuous modifier used to split the internal nodes of modifier trees.

In the 'Individual' tab, the user can adjust the levels of modifiers to obtain the individualized estimate of the distributed lag effects. In our context, the **shiny** app provides

the personalized exposure effect and critical windows when the sex of a child, age, BMI, and smoking habits of a mother are specified. Figure 9 shows the estimated personalized exposure effect of $PM_{2.5}$ during gestational weeks for a 29-year-old mother with a BMI of 24, whose child is male and who does not smoke.

Figure 9: Example of estimated personalized exposure effect in R **shiny** app. The displayed results are based on simulated data.

The last tab labeled 'Subgroup' offers subgroup-specific analyses. In the top panel, shown in Figure 10, the user can select one or two modifiers to group the samples into multiple subgroups and obtain personalized exposure effects for individuals in each subgroup. Each line of the resulting plot represents an exposure effect of an individual accounting for all modifiers of that individual. This is useful for simultaneously assessing how much exposure effects vary among individuals and across subgroups. The bottom panel allows users to

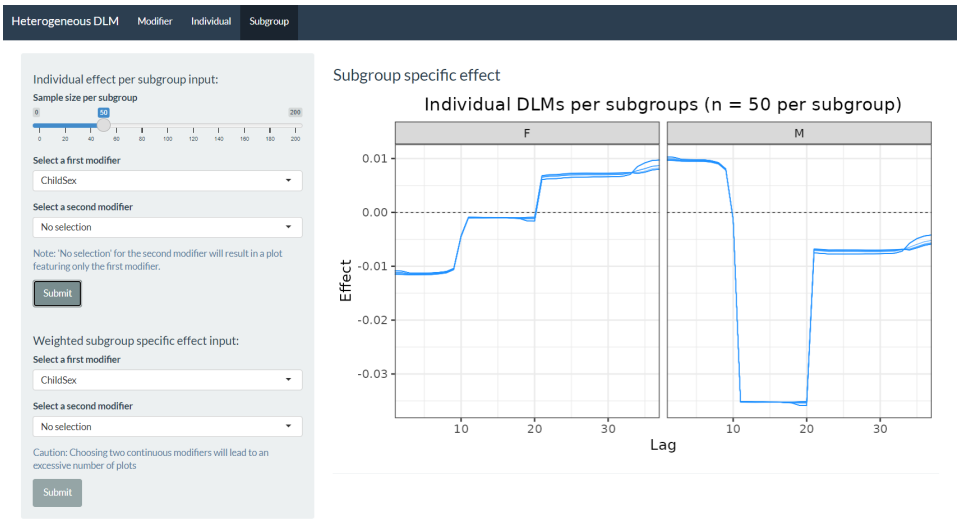


Figure 10: Example of estimated personalized exposure effects with subgroups in R **shiny** app. The displayed results are based on simulated data.

select one or two modifiers for subgroup-specific distributed lag effects. Subgroup-specific distributed lag effects are calculated by marginalizing out the modifiers not specified in the panel. Two modifiers at most can be specified for analyzing how much each modifier affects heterogeneity in the exposure effects. For example, Figure 11 shows the estimated exposure-time-response function for four subgroups, grouped by two categorical modifiers: child sex and smoking habit. The difference in subgroup-specific exposure effects indicates that smoking habit does not contribute much heterogeneity in the exposure effect of $PM_{2.5}$, while child sex introduces a considerable amount, which aligns with the modifier PIPs in

the summary output. Using the simulated dataset, the subgroup-specific effects suggest that mothers with a male child may be more vulnerable to $PM_{2.5}$.

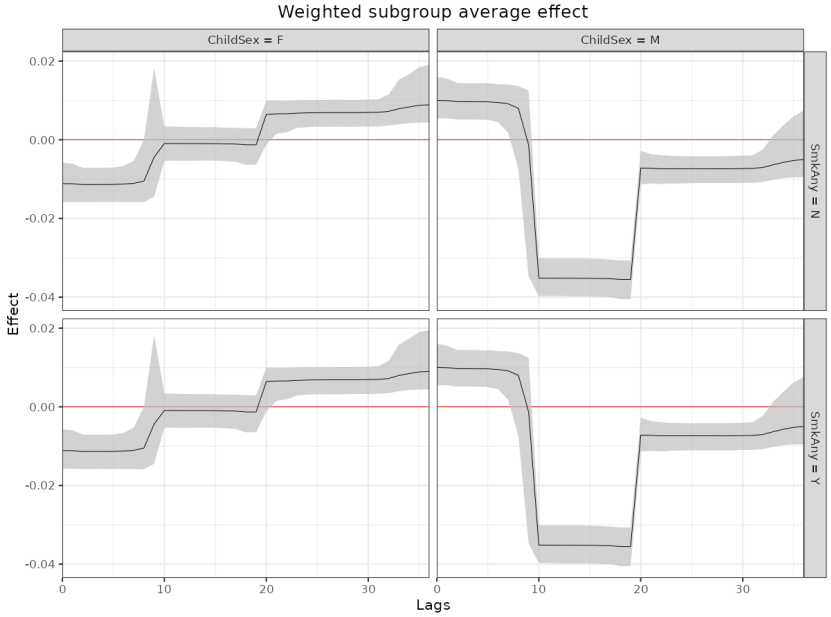


Figure 11: Example of estimated subgroup-specific effects, grouped by child sex and maternal smoking status, in R **shiny** app. These results are based on simulated data.

5 Practical considerations

R package **dlmtree** offers highly flexible and integrative models for analyzing the effects of repeatedly measured exposures. The package allows user specification for adjusting the flexibility of the exposure-lag-response relationship, number of exposures, types of lagged interaction terms, modifiers, and MCMC simulation control. We highlight important considerations and best practices when using the package to obtain reliable results.

The models included in **dlmtree** are very flexible and highly parametric. It is, therefore, important to be mindful of whether such a complex model is relevant to a research question. Factors that increase model complexity are: increased number of time points the exposure is assessed at, more unique exposures and allowing for additional interactions between or within exposures across time, and including heterogeneity and the number of candidate modifiers. The BART framework and shrinkage and selection priors are effective at regularizing these models. However, more complex models can still result in increased variance, identifiability issues where the model becomes sensitive to model specification, and challenges in interpretation. Therefore, we recommend users employ standard best practices in model building, such as carefully assessing what should be included in the model, considering pre-selection of exposures or combining related exposures into a single index, aggregating exposure measurements to a coarser time interval, being mindful of extreme multicollinearity, and conducting sensitivity analyses with simpler models where possible.

When running heterogeneous distributed lag models, potentially important modifiers can be identified using the PIP. When the number of modifiers is small relative to the number of trees, all modifiers will have a higher baseline PIP. Additionally, a large PIP does not necessarily imply meaningful effect modification, and the posterior distribution of the exposure-response function should be explored to better identify important modifiers. Related to model complexity, when two or more modifiers are highly correlated, one or both may be identified as driving heterogeneity. Therefore, considering the mechanisms of effect modification is essential when selecting modifiers and interpreting results.

For practical implementation, we recommend running longer MCMC chains than are used in the examples in this paper. While the examples presented use short iterations for demonstration, longer chains are necessary for stable estimates and to improve mixing and convergence. Given the high dimensionality of the tree structured DLMs, a longer chain will sample more reliable full posterior distributions for inference. Additionally, for logistic and negative binomial models, longer burn-in periods are important to ensure convergence. For example, in our previous large data analyses with this software (Mork and Wilson, 2023; Mork et al., 2024; Mork and Wilson, 2022), we have used upwards of 10,000 burn-in iterations.

Convergence can be assessed in several ways. We have relied on trace plots, which graph the chain of posterior draws for key parameters. When possible, another approach is to run multiple Markov chains with different seeds and compare results across chains. To help users assess model convergence and identify potential issues with mixing, our package provides a `diagnose` S3 method for model summary objects which returns a comprehensive convergence diagnostics panel customized for the treed DLM framework. It provides an overview of key outputs including trace plots and posterior density plots of distributed lag effects, Metropolis-Hastings acceptance rate for tree updates, and changes in tree sizes throughout MCMC sampling. The example usage of `diagnose` is demonstrated in the supplementary materials.

6 Summary

We introduced the R package **dlmtree**, a user-friendly software for addressing a wide range of research questions regarding the relationship between a longitudinally assessed exposure or mixture and a scalar outcome. Our software provides functionality to estimate distributed lag linear or nonlinear models, quantify main and interaction effects, and account for heterogeneity in the exposure-time-response function. Furthermore, the methods in our package have been carefully optimized for computational efficiency under a custom C++ language framework with a convenient R wrapper function, `dlmtree`, designed for accessibility by all researchers. In this paper, we provided an overview of the regression tree approaches used for estimating DLMs, and through a collection of vignettes, we highlighted a variety of tools available for processing data, fitting models, conducting inference, and visualizing the results. Our goal in making this package available is to bring robust data science tools to expand the range of questions that can be asked and answered with longitudinally assessed data.

7 Acknowledgements

Research reported in this publication was supported by National Institute of Environmental Health Sciences of the National Institutes of Health under award numbers ES035735, ES029943, ES028811, AG066793, and ES034021. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

References

H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010. URL https://doi.org/10.1214/09-AOAS285. [p136]

Y.-H. M. Chiu, A. Wilson, H.-H. L. Hsu, H. Jamal, N. Mathews, I. Kloog, J. Schwartz, D. C. Bellinger, N. Xhani, R. O. Wright, et al. Prenatal ambient air pollutant mixture exposure and neurodevelopment in urban children in the northeastern United States. *Environmental Research*, 233:116394, 2023. URL https://doi.org/j.envres.2023.116394. [p136]

D. Dematteis, K. P. Keller, D. Rojas-Rueda, M.-A. Kioumourtzoglou, and A. Wilson. Penalized distributed lag interaction model: Air pollution, birth weight, and neighborhood vulnerability. *Environmetrics*, page e2843, 2024. URL https://doi.org/10.1002/env.2843. [p137]

D. Dempsey and J. Wyse. Bayesian variable selection in distributed lag models: A focus on binary quantile and count data regressions, 2025. URL https://arxiv.org/abs/2403.03646. [p137]

A. Gasparrini. Distributed lag linear and non-linear models in R: The package dlnm. *Journal of Statistical Software*, 43(8):1–20, 2011. URL https://doi.org/10.18637/jss.v043.i08. [p137]

A. Gasparrini, B. Armstrong, and M. G. Kenward. Distributed lag non-linear models. *Statistics in Medicine*, 29(21):2224–2234, 2010. URL https://doi.org/10.1002/sim.3940. [p136]

H.-H. L. Hsu, A. Wilson, J. Schwartz, I. Kloog, R. O. Wright, B. A. Coull, and R. J. Wright. Prenatal ambient air pollutant mixture exposure and early school-age lung function. *Environmental Epidemiology*, 7(2):e249, 2023. URL https://doi.org/10.1097/EE9.0000000000000249. [p136]

L. M. Koyck. *Distributed lags and investment analysis*, volume 4. North-Holland Publishing Company Amsterdam, 1954. [p136]

D. Mork and A. Wilson. Treed distributed lag nonlinear models. *Biostatistics*, 23(3):754–771, 2022. URL https://doi.org/10.1093/biostatistics/kxaa051. [p136, 137, 140, 157]

D. Mork and A. Wilson. Estimating perinatal critical windows of susceptibility to environmental mixtures via structured Bayesian regression tree pairs. *Biometrics*, 79(1):449–461, 2023. URL https://doi.org/10.1111/biom.13568. [p136, 137, 138, 139, 140, 157]

D. Mork and A. Wilson. Incorporating prior information into distributed lag nonlinear models with zero-inflated monotone regression trees. *Bayesian Analysis*, 1(1):1–29, 2024. URL https://doi.org/10.1214/23-BA1412. [p140]

D. Mork, M.-A. Kioumourtzoglou, M. Weisskopf, B. A. Coull, and A. Wilson. Heterogeneous distributed lag models to estimate personalized effects of maternal exposures to air pollution. *Journal of the American Statistical Association*, 119(545):14–26, 2024. URL https://doi.org/10.1080/01621459.2023.2258595. [p136, 141, 157]

B. Neelon. Bayesian zero-inflated negative binomial regression based on pólya-gamma mixtures. *Bayesian Analysis*, 14(3):829, 2019. URL https://doi.org/10.1214/18-ba1132. [p140]

K. S. Palda. The measurement of cumulative advertising effects. *The Journal of Business*, 38 (2):162–179, 1965. [p136]

N. G. Polson, J. G. Scott, and J. Windle. Bayesian inference for logistic models using pólya–gamma latent variables. *Journal of the American Statistical Association*, 108(504):1339–1349, 2013. URL https://doi.org/10.1080/01621459.2013.829001. [p140]

M. J. Rosa, H.-H. L. Hsu, A. C. Just, K. J. Brennan, T. Bloomquist, I. Kloog, I. Pantic, A. M. García, A. Wilson, B. A. Coull, et al. Association between prenatal particulate air pollution exposure and telomere length in cord blood: Effect modification by fetal sex. *Environmental Research*, 172:495–501, 2019. URL https://doi.org/10.1016/j.envres.2019.03.003. [p141]

E. M. Schliep, T. L. J. Schafer, and M. Hawkey. Distributed lag models to identify the cumulative effects of training and recovery in athletes using multivariate ordinal wellness data. *Journal of Quantitative Analysis in Sports*, 17(3):241–254, 2021. URL https://doi.org/10.1515/jqas-2020-0051. [p136]

J. Schwartz. The distributed lag between air pollution and daily deaths. *Epidemiology*, 11(3): 320–326, 2000. URL https://doi.org/10.1097/00001648-200005000-00016. [p136]

J. Warren, M. Fuentes, A. Herring, and P. Langlois. Spatial-temporal modeling of the association between air pollution exposure and preterm birth: Identifying critical windows of exposure. *Biometrics*, 68(4):1157–1167, 2012. URL https://doi.org/10.1111/j.1541-0420.2012.01774.x. [p136]

A. Wilson, Y.-H. M. Chiu, H.-H. L. Hsu, R. O. Wright, R. J. Wright, and B. A. Coull. Potential for bias when estimating critical windows for air pollution in children's health. *American Journal of Epidemiology*, 186(11):1281–1289, 06 2017a. ISSN 0002-9262. doi: 10.1093/aje/kwx184. URL https://doi.org/10.1093/aje/kwx184. [p136]

A. Wilson, Y.-H. M. Chiu, H.-H. L. Hsu, R. O. Wright, R. J. Wright, and B. A. Coull. Bayesian distributed lag interaction models to identify perinatal windows of vulnerability in children's health. *Biostatistics*, 18(3):537–552, 2017b. URL https://doi.org/10.1093/biostatistics/kxx002. [p137]

A. Zanobetti, M. P. Wand, J. Schwartz, and L. M. Ryan. Generalized additive distributed lag models: Quantifying mortality displacement. *Biostatistics*, 1(3):279–292, 2000. URL https://doi.org/10.1093/biostatistics/1.3.279. [p136]

Seongwon Im
Department of Statistics
Colorado State University
United States of America
(0009-0000-8447-5852)
seongwon.im@colostate.edu

Ander Wilson
Department of Statistics
Colorado State University
United States of America
(0000-0003-4774-3883)
ander.wilson@colostate.edu

Daniel Mork
Department of Biostatistics
Harvard T.H. Chan School of Public Health
United States of America
(0000-0002-7924-0706)
dmork@hsph.harvard.edu

CDsampling: An R Package for Constrained D-Optimal Sampling in Paid Research Studies

by Yifei Huang, Liping Tong, and Jie Yang

Abstract In the context of paid research studies and clinical trials, budget considerations often require patient sampling from available populations, which comes with inherent constraints. We introduce the R package CDsampling, which is the first to our knowledge to integrate optimal design theories within the framework of constrained sampling. This package offers the possibility to find both D-optimal approximate and exact allocations for samplings with or without constraints. Additionally, it provides functions to find constrained uniform sampling as a robust sampling strategy when the model information is limited. To demonstrate its efficacy, we provide simulated examples and a real-data example with datasets embedded in the package and compare them with classical sampling methods. Furthermore, the CDsampling package revisits the theoretical results of the Fisher information matrix for generalized linear models (including the regular linear regression model) and multinomial logistic models, offering functions for its computation.

1 Introduction

Paid research studies are essential for determining the influence of new interventions or treatments and for providing quantitative evidence in various domains, such as healthcare, psychology, and politics. However, conducting such studies often involves a limited budget and a large pool of potential volunteers, which poses a challenge in selecting the best sample to meet the research objectives. Poor samples may result in biased or inaccurate estimates, low statistical power, and even misleading conclusions. Therefore, finding a good sampling strategy is crucial for researchers and practitioners.

Consider a constrained sampling problem commonly encountered in paid research studies. Suppose, in a research study to evaluate a new intervention, $N = 500$ volunteers register to participate. Upon registration, the investigators collect basic demographic information, such as gender (female or male) and age groups ($18 \sim 25$, $26 \sim 64$, 65 and above) for each volunteer. Treating gender and age as stratification factors, we obtain $m = 6$ subgroups. However, due to budget limitations, the study could only accommodate $n = 200$ participants. Let N_i denote the frequency of volunteers within the ith subgroup, where $i = 1, \ldots, m$. We call the integer number of participants sampled from each subgroup, n_i, the exact allocation, and the corresponding proportion n_i/N_i the approximate allocation. The goal is to select a sample of 200 participants $\mathbf{n} = (n_1, \ldots, n_m)$, such that $\sum_{i=1}^{m} n_i = 200$ from the pool of $N = 500$ volunteers to evaluate the intervention effect most accurately, subject to the constraint that no subgroup is oversampled beyond the number of available volunteers within that subgroup, that is, $n_i \leq N_i$. This constraint is the most commonly encountered in paid research studies (see Section 3.1 for details, and for constraints of other forms, please refer to Section 3.2).

Commonly used sampling strategies include simple random sampling, stratified sampling, and cluster sampling. Simple random sampling is the most straightforward form of probability sampling, where each element has an equal chance of being selected (Tillé, 2006). Proportionally stratified sampling involves dividing the population into homogeneous subgroups, such as gender and age groups, and applying random sampling to sample proportionally within each subgroup (Lohr, 2019). Cluster sampling, on the other hand, splits the population into heterogeneous clusters, for example, based on the locations of volunteers, and randomly selects some clusters as the sample units. However, these methods have their drawbacks. Cluster sampling is relatively low-cost but less precise than simple random sampling. Stratified sampling can produce more efficient estimators of population

means, but it requires finding well-defined and relevant subgroups that cover the entire population (Levy and Lemeshow, 2008). Moreover, these existing methods are based on assumptions that may not hold if model estimation is the primary goal of the paid research study.

In the proposed **CDsampling** package, we implement the sampling strategy based on optimal design theory (with main functions `liftone_GLM()`, `liftone_constrained_GLM()`, `liftone_MLM()`, and `liftone_constrained_MLM()`), which can improve the accuracy of the intervention effect estimation. Optimal design theory is a branch of experimental design that aims to find the best allocation of experimental units to achieve a specific optimality criterion such as minimizing the variance of estimation or equivalently maximizing the information obtained from the design. For example, D-optimality maximizes the determinant of the information matrix, which minimizes the estimators' expected volume of the joint confidence ellipsoid. A-optimality minimizes the trace of the inverse information matrix, equivalent to minimizing the average of the variances of the estimators. E-optimality minimizes the maximum eigenvalue of the inverse information matrix, which minimizes the largest expected semi-axis of the confidence ellipsoid and protects against the worst possible case (Fedorov, 1972; Atkinson et al., 2007; Yang and Stufken, 2012; Fedorov and Leonov, 2014). In this paper, we focus on D-optimality due to its overall good performance and mathematical simplicity (Zocchi and Atkinson, 1999; Atkinson et al., 2007). According to Huang et al. (2025), the constrained D-optimal sampling strategies work well for paid research studies or clinical trials. To implement the recommended sampling strategy, we develop an R package called **CDsampling** (namely, Constrained D-optimal sampling), available on CRAN at `https://cran.r-project.org/package=CDsampling`. To the best of our knowledge, **CDsampling** is the first R package offering a sampling tool with constraints in paid research studies based on optimal design theory. Package **sampling** implements random samples using different sampling schemes (Tillé and Matei, 2016). The package also provides functions to obtain (generalized) calibration weights, different estimators, as well as some variance estimators. Package **SamplingStrata** determines the best stratification for a population frame that ensures the minimum sample cost with precision thresholds (Barcaroli, 2014). On the other hand, there are existing R packages for optimal designs. Package **AlgDesign** finds exact and approximate allocations of optimal designs under D-, A-, and I-criteria (Wheeler and Braun, 2019). Package **OptimalDesign** enables users to compute D-, A-, I-, and c-efficient designs with or without replications, restricted design spaces, and under multiple linear constraints (Harman et al., 2016). Package **acebayes** finds Bayesian optimal experimental designs with approximate coordinate exchange algorithm (Overstall et al., 2020, 2018). Package **OPDOE** provides functions for optimal designs with polynomial regression models and ANOVA (Grömping, 2011). These packages provide programming tools for finding samplings or optimal designs under different criteria and models. However, they do not address the specific challenges of practical feasibility in the constrained sampling scheme of paid research studies. The proposed **CDsampling** package fills this gap by offering an efficient sampling tool to handle general constraints and common parametric models in paid research studies.

2 Method

2.1 Constrained lift-one algorithm

The lift-one algorithm was initially proposed by Yang et al. (2016) to find D-optimal designs with binary responses. This was extended to generalized linear models (GLMs) by Yang and Mandal (2015) and subsequently adapted for cumulative link models by Yang et al. (2017). The methodology was further extended to multinomial logit models (MLMs) by Bu et al. (2020).

Figure 1 provides a concise summary of the lift-one algorithm applied to general parametric models. The detailed algorithm is provided in Algorithm 3 from the Supplementary Material of Huang et al. (2025). We consider a general study or experiment involving co-

variates $\mathbf{x}_i = (x_{i1}, \ldots, x_{id})^\top$, for $i = 1, \ldots, m$, referred to as experimental settings. In paid research studies, these covariates could be the stratification factors such as the gender and age groups in our motivating example. Here, $m \geq 2$ denotes the number of experimental settings, which corresponds to $m = 6$, the number of gender and age groups in the motivating example. Suppose the responses follow a parametric model $M(\mathbf{x}_i, \boldsymbol{\theta})$, where $\boldsymbol{\theta} \subseteq \mathbb{R}^p$ with $p \geq 2$. Under regularity conditions, the Fisher information matrix of the experiment design can be written as $\mathbf{F} = \sum_{i=1}^m n_i \mathbf{F}_i$, where $\mathbf{F}_i, i = 1, \ldots, m$ is the Fisher information matrix at $\mathbf{x}_i$ and n_i is the number of subjects allocated to the ith experimental setting. In the setting of paid research studies, n_i corresponds to the number of subjects sampled from the ith subgroup. We usually call $\mathbf{n} = (n_1, \ldots, n_m)^\top$ the exact allocation, where $n = \sum_{i=1}^m n_i$, and $\mathbf{w} = (w_1, \ldots, w_m)^\top = (n_1/n, \ldots, n_m/n)^\top$ the approximate allocation, where $w_i \geq 0$ and $\sum_{i=1}^m w_i = 1$ (Kiefer, 1974; Pukelsheim, 2006; Atkinson et al., 2007). The approximate allocation is theoretically more tractable, while the exact allocation is practically more useful. In the statistical literature, approximate allocations are more commonly discussed (Kiefer, 1974). The D-optimal design aims to find the optimal allocation that maximizes $f(\mathbf{w}) = |\mathbf{F}(\mathbf{w})| = |\sum_{i=1}^m w_i \mathbf{F}_i|$. The lift-one algorithm simplifies a complex multivariate optimization problem into a sequence of simpler univariate optimization problems. This is achieved by "lifting" and optimizing one weight w_i, within the approximate allocation vector $\mathbf{w}$. Specifically, in step 3° of the algorithm depicted in Figure 1, the determinant of the Fisher information matrix function concerning the lift-one variable is expressed as $f(z) = f(\mathbf{w}_i(z))$ where the variable z substitutes for w_i in allocation $\mathbf{w}$, and the remaining weights are adjusted proportionally, denoted as $\mathbf{w}_i$. The updated weight vector is given by

$$\mathbf{w}_i(z) = \left(\frac{1-z}{1-w_i} w_1, \ldots, \frac{1-z}{1-w_i} w_{i-1}, z, \frac{1-z}{1-w_i} w_{i+1}, \ldots, \frac{1-z}{1-w_i} w_m \right)^\top.$$

The allocation that results from the convergence in step 6° of the algorithm is identified as the D-optimal approximate allocation.

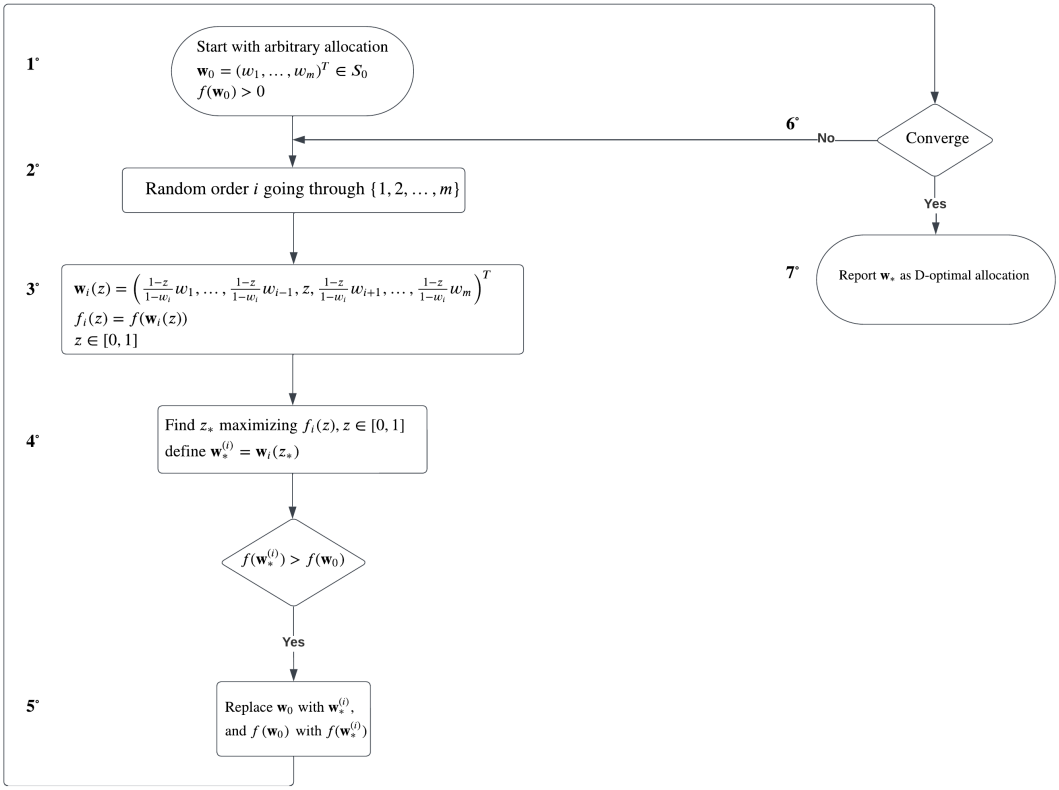


Figure 1: The framework of lift-one algorithm.

In the context of paid research studies, budgetary limitations often necessitate the se-

lection of a subset of participants. We consider the sampling of n subjects from a larger population of N, where $n < N$. A typical constraint in such studies is $n_i \leq N_i$, where N_i represents the number of available subjects from the ith experimental setting group. This effectively places an upper bound on the sample size for each subgroup (or stratum), ensuring the sampling process does not overdraw from any subgroup. Additional constraints may include $n_1 + n_2 + n_3 + n_4 \leq 392$ (see the MLM example in Section 3.2), $4n_1 \geq n_3$ (see Example S2.2 in Supplementary Material Section S2), etc. The constrained lift-one algorithm seeks the approximate allocation $\mathbf{w}$ that maximizes $|\mathbf{F}(\mathbf{w})|$, on a collection of feasible approximate allocations $S \subseteq S_0$ where

$$S_0 := \{(w_1, \ldots, w_m)^\top \in \mathbb{R}^m \mid w_i \geq 0, i = 1, \ldots, m; \sum_{i=1}^{m} w_i = 1\}.$$

The set S is presumed to be either a closed convex set or a finite union of closed convex sets. The framework of the constrained lift-one algorithm is provided in Figure 2. The details of the algorithm can be found in Algorithm 1 of Huang et al. (2025). In the constrained lift-one algorithm, the search for the optimal lift-one weight z in step 3° of Figure 2 is confined within the interval of $[r_{i1}, r_{i2}]$ (Example S2.2 in Supplementary Material Section S2 and Section S.3 in Huang et al. (2025) for details of finding $[r_{i1}, r_{i2}]$). To ensure that the resulting allocation is D-optimal, two additional decision steps, labeled steps 7° and 8°, are incorporated into the algorithm. The reported allocation in step 10° is the constrained D-optimal approximate allocation for the study. To illustrate the difference between the lift-one algorithm and the constrained lift-one algorithm, we provide examples in Supplementary Material Section S2.

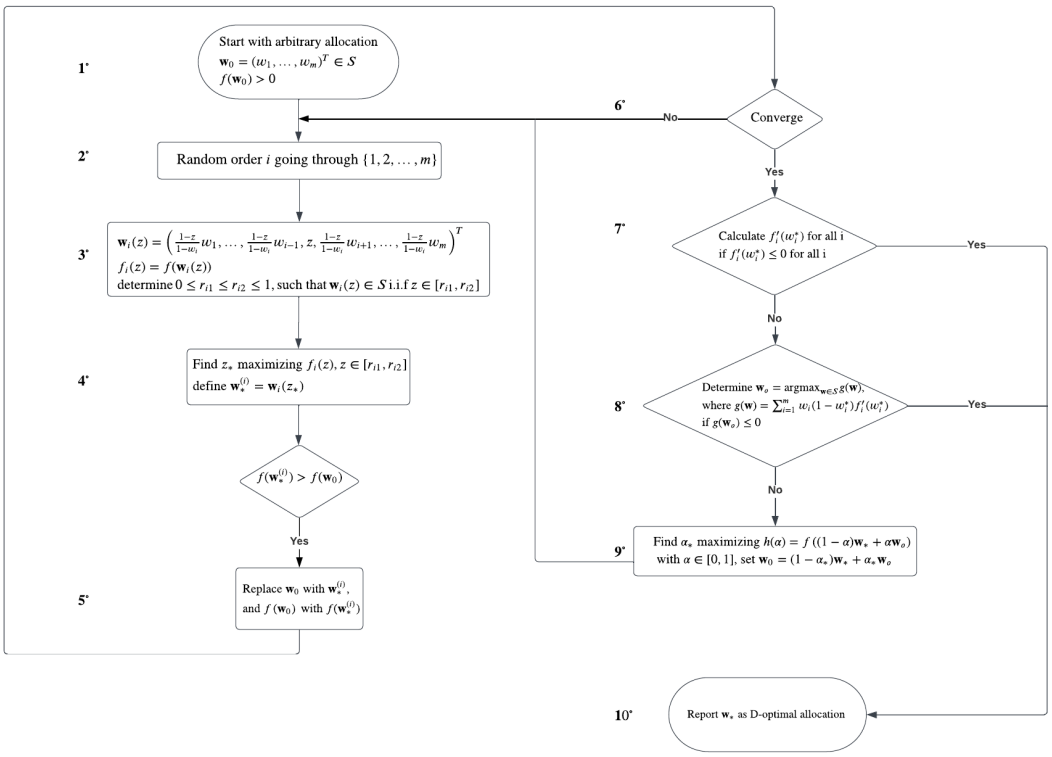


Figure 2: The framework of constrained lift-one algorithm.

Upon obtaining the approximate allocation, we may employ the constrained approximate to exact allocation algorithm outlined in Figure 3 for the conversion of a real-valued approximate allocation to an integer-valued exact allocation. The full details of the algorithm are in Algorithm 2 of Huang et al. (2025). The algorithm begins by assigning a floor integer value to all subgroups $n_i = \lfloor N w_i \rfloor$. Subsequently, each remaining subject is added to the corresponding group in a manner that maximizes the determinant of the Fisher information matrix. This transformation provides a more pragmatic application in the actual sampling

process within paid research studies.

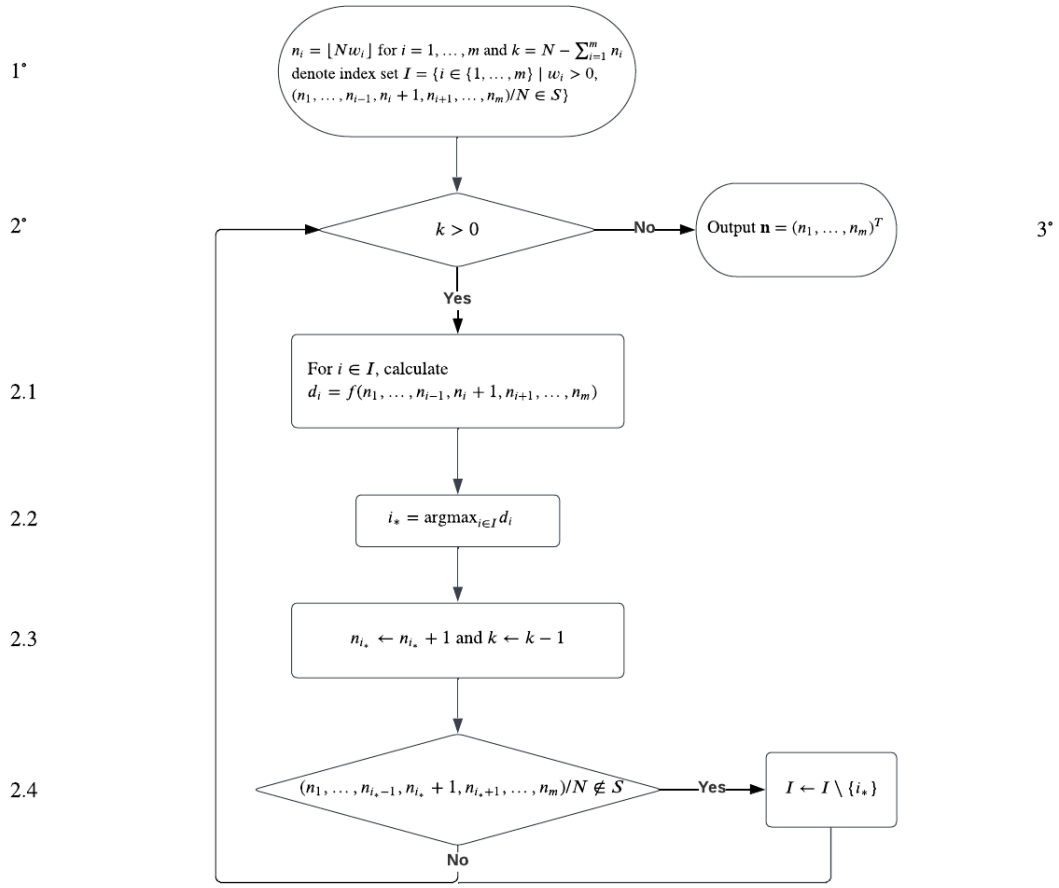


Figure 3: The framework of constrained approximate to exact algorithm.

The calculation of the Fisher information matrix $\mathbf{F}$ and the subsequent maximization of its determinant $|\mathbf{F}|$ are key steps in both the constrained and original (unconstrained) lift-one algorithms. The theoretical details and functions for the computation of $\mathbf{F}$ are provided in Sections 2.2 and Section 2.3.

2.2 Fisher information for generalized linear models

The generalized linear model (GLM) broadens the scope of the traditional linear model. In the standard approach, the response variable is expected to change in direct proportion to the covariate. Yet, this isn't always a practical assumption. Take binary outcomes, for instance, the classic linear model falls short here. Similarly, it's unsuitable for positive-only data like count data. Nelder and Wedderburn (1972) expanded the model to accommodate a wider range of applications.

For a GLM, we assume that response variables $Y_1, \ldots, Y_n$ are independent and from the exponential family. Then we have (Dobson and Barnett, 2018; McCullagh and Nelder, 1989)

$$E(Y_i \mid \mathbf{X}_i) = \mu_i, \quad g(\mu_i) = \eta_i = \mathbf{X}_i^\top \boldsymbol{\beta}$$

where g is a link function, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)^\top$ is the parameter vector of interest, and μ_i is the conditional expectation of response Y_i given predictors $\mathbf{X}_i = \mathbf{h}(\mathbf{x}_i) = (h_1(\mathbf{x}_i), \ldots, h_p(\mathbf{x}_i))^\top$, where $i = 1, \ldots, n$ with known and deterministic predictor functions $\mathbf{h} = (h_1, \ldots, h_p)^\top$. There are various link functions that could be used, for example, logit link $\eta_i = \log \frac{\mu_i}{1-\mu_i}$; probit link $\eta_i = \Phi^{-1}(\mu_i)$, where $\Phi(\cdot)$ is the normal cumulative distribution function; and complementary log-log link $\eta_i = \log\{-\log(1 - \mu_i)\}$. Regular linear models

can be considered as GLMs with the identity link function and normal responses. Suppose we have a design with m design points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ that has an exact allocation $(n_1, \ldots, n_m)$ where $\sum_i n_i = n$ and corresponding approximate allocation $(w_1, \ldots, w_m) = (\frac{n_1}{n}, \ldots, \frac{n_m}{n})$. The Fisher information matrix $\mathbf{F}$ under GLMs can be written as (McCullagh and Nelder, 1989; Khuri et al., 2006; Stufken and Yang, 2012; Yang et al., 2016):

$$\mathbf{F} = n\mathbf{X}^\top \mathbf{W}\mathbf{X} = n \sum_{i=1}^{m} w_i \nu_i \mathbf{X}_i \mathbf{X}_i^\top \tag{1}$$

where $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_m)^\top$ is the $m \times p$ design matrix with $\mathbf{X}_i = \mathbf{h}(\mathbf{x}_i)$, $\mathbf{W} = \text{diag}\{w_1\nu_1, \ldots, w_m\nu_m\}$ is a diagonal matrix with $\nu_i = \frac{1}{\text{Var}(Y_i)}(\frac{\partial \mu_i}{\partial \eta_i})^2$. Here, ν_i represents how much information the design point $\mathbf{x}_i$ contains. The explicit formats of ν_i with different response distributions and link functions can be found in Table 5 of the Supplementary Material of Huang et al. (2025). To calculate the Fisher information matrix $\mathbf{F}$ given the approximate allocation $\mathbf{w}$, we may use the `F_func_GLM()` function in the **CDsampling** package. Additionally, the `W_func_GLM()` function can be used to find the diagonal elements of the matrix $\mathbf{W}$ in the Fisher information matrix (1) of GLM. An example of finding the Fisher information matrix for GLM is provided in Example S1.1 of Supplementary Material.

2.3 Fisher information for multinomial logistic model

The multinomial logistic model (MLM) is an extension of GLM aiming to manage responses that fall into multiple categories, such as rating scales and disease severity levels (Agresti, 2013).

We assume that the responses $\mathbf{Y}_i = (Y_{i1}, \ldots, Y_{iJ})$ follow a multinomial distribution with probabilities $(\pi_{i1}, \ldots, \pi_{iJ})$, where $\pi_{ij} = \mathbb{P}(Y_i = j \mid \mathbf{x}_i)$, $Y_i \in \{1, \ldots, J\}$ is the ith original categorical response, $j = 1, \ldots, J$, and $\sum_j \pi_{ij} = 1$. If the response variables are nominal, in other words, there is no natural ordering among different levels, a commonly used MLM is the baseline-category logit model with the Jth level selected as the baseline category. If the response variable is ordinal, that is, we have a natural ordering of response levels, there are three commonly used MLMs in the literature: cumulative logit model, adjacent logit model, and continuation-ratio logit model (Bu et al., 2020; Dousti Mousavi et al., 2023; Wang and Yang).

In addition to different logit models, the proportional odds (po) assumption is an important concept in MLMs. The po assumption is a parsimonious model assumption, where a model simultaneously uses all $J-1$ logits in a single model with the same coefficients. This means the covariate effect is constant on the odds ratio among different response levels. When the assumption doesn't hold, the model is referred to as a non-proportional odds (npo) model and has more parameters in it. When the assumption only holds for part of the parameters, the model is referred to as a partial proportional odds (ppo) model. Commonly used multinomial logit models with po, npo, or ppo assumptions can be summarized in a unified matrix form (Glonek and McCullagh, 1995; Zocchi and Atkinson, 1999; Bu et al., 2020):

$$\mathbf{C}^\top \log(\mathbf{L}\boldsymbol{\pi}_i) = \mathbf{X}_i\boldsymbol{\theta}$$

where

$$\mathbf{C}^\top = \begin{pmatrix} \mathbf{I}_{J-1} & -\mathbf{I}_{J-1} & \mathbf{0}_{J-1} \\ \mathbf{0}_{J-1}^\top & \mathbf{0}_{J-1}^\top & 1 \end{pmatrix}$$

is a $J \times (2J-1)$ constant matrix, $\mathbf{L}$ is a $(2J-1) \times J$ constant matrix with different formats among the four different multinomial logit models, and $\boldsymbol{\pi}_i = (\pi_{i1}, \ldots, \pi_{iJ})^\top$. The model

matrix $\mathbf{X}_i$ is defined in general as

$$\mathbf{X}_i = \begin{pmatrix} \mathbf{h}_1^\top(\mathbf{x}_i) & & & \mathbf{h}_c^\top(\mathbf{x}_i) \\ & \ddots & & \vdots \\ & & \mathbf{h}_{J-1}^\top(\mathbf{x}_i) & \mathbf{h}_c^\top(\mathbf{x}_i) \\ \mathbf{0}_{p_1}^\top & \dots & \mathbf{0}_{p_{J-1}}^\top & \mathbf{0}_{p_c}^\top \end{pmatrix}_{J\times p}$$

and the parameter $\boldsymbol{\theta} = (\boldsymbol{\beta}_1^\top, \dots, \boldsymbol{\beta}_{J-1}^\top, \boldsymbol{\zeta}^\top)^\top$ consists of $p = p_1 + \cdots + p_{J-1} + p_c$ unknown parameters. Here $\mathbf{h}_j^\top(\cdot) = (h_{j1}(\cdot), \dots, h_{jp_j}(\cdot))$ are known functions to determine p_j predictors associated with unknown parameters $\boldsymbol{\beta}_j = (\beta_{j1}, \dots, \beta_{jp_j})^\top$ in jth response category, and $\mathbf{h}_c^\top(\cdot) = (h_1(\cdot), \dots, h_{p_c}(\cdot))$ are known functions to determine p_c predictors associated with proportional odds parameters $\boldsymbol{\zeta} = (\zeta_1, \dots, \zeta_{p_c})^\top$. If $\mathbf{h}_j^\top(\mathbf{x}_i) \equiv 1$, the model is a po model; if $\mathbf{h}_c^\top(\mathbf{x}_i) \equiv 0$, the model is an npo model.

According to Theorem 2.1 in Bu et al. (2020), the Fisher information matrix under a multinomial logistic regression model with independent observations can be written as

$$\mathbf{F} = \sum_{i=1}^{m} n_i \mathbf{F}_i \tag{2}$$

where $\mathbf{F}_i = (\frac{\partial \boldsymbol{\pi}_i}{\partial \boldsymbol{\theta}^\top})^\top \mathrm{diag}(\boldsymbol{\pi}_i)^{-1} \frac{\partial \boldsymbol{\pi}_i}{\partial \boldsymbol{\theta}^\top}$ with $\frac{\partial \boldsymbol{\pi}_i}{\partial \boldsymbol{\theta}^\top} = (\mathbf{C}^\top \mathbf{D}_i^{-1} \mathbf{L})^{-1}$ and $\mathbf{D}_i = \mathrm{diag}(\mathbf{L}\boldsymbol{\pi}_i)$. Explicit forms of $(\mathbf{C}^\top \mathbf{D}_i^{-1} \mathbf{L})^{-1}$ can be found in Section S.3 of the Supplementary Material of Bu et al. (2020). To calculate the Fisher information matrix $\mathbf{F}$ for the MLM, one may use the function `F_func_MLM()` in the **CDsampling** package. An example of finding the Fisher information matrix for MLM is provided in Example S1.2 of the Supplementary Material.

3 Examples

The methods described in Section 2 are implemented in the proposed R package **CDsampling**. The **CDsampling** package comprises 16 functions, as detailed in Table 1. The primary functions of the **CDsampling** package are `liftone_constrained_GLM()` and `liftone_constrained_MLM()`. Additionally, the package includes the original (unconstrained) lift-one algorithm for general experimental designs, accessible via the `liftone_GLM()` and `liftone_MLM()` functions. Two datasets `trial_data` and `trauma_data` are provided for illustration purposes.

In the remainder of this section, we present two examples to illustrate the usage of the **CDsampling** package for sampling problems in paid research studies, using datasets provided by **CDsampling**. All results in this section were generated using R version 4.3.2 on a macOS Sonoma 14.6.1 system.

3.1 Applications in paid research study: `trial_data` example

The `trial_data` dataset is simulated data for a toy example of paid research studies. This study includes a cohort of 500 patients for a clinical trial, with gender and age as stratification factors. A logistic regression model incorporates these factors as covariates: `gender` (coded as 0 for female and 1 for male) and `age` (coded as two dummy variables *age*_1 and *age*_2 with $(age\_1, age\_2) = (0,0)$ for age group $18 \sim 25$; $(1,0)$ for age group $26 \sim 64$, and $(0,1)$ for age group 65 and above). For simplicity, the study assumes binary gender options and a tripartite age categorization. In total, there are $m = 6$ combinations of covariate factors. In practice, non-binary gender options and a "prefer not to answer" choice may be included to respect gender diversity and protect patient confidentiality. The response Y denotes the treatment's efficacy (0 indicating ineffectiveness, 1 indicating effectiveness). The data is

	Usage	Function
Model	Calculating **W** matrix diagonal elements of generalized linear model (see Section 2.2); providing input for function `liftone_GLM()` and `liftone_constrained_GLM()`.	`W_func_GLM()`
	Calculating Fisher information matrix and its determinant of generalized linear model (see Example S1.1 in Supplementary Material).	`F_func_GLM()`, `Fdet_func_GLM()`
	Calculating Fisher information matrix of multinomial logit model at a specific design point (see Section 2.3); using as input of `liftone_MLM()` and `liftone_constrained_MLM()`.	`Fi_func_MLM()`
	Calculating Fisher information matrix and its determinant of multinomial logit model (see Example S1.2 in Supplementary Material).	`F_func_MLM()`, `Fdet_func_MLM()`
	Using in `approxtoexact_constrained_func()` to find constrained uniform exact allocation.	`Fdet_func_unif()`
Sampling	Finding unconstrained D-optimal approximate allocation for generalized linear model and multinomial logit model (see Section S2 in Supplementary Material).	`liftone_GLM()`, `liftone_MLM()`
	Finding constrained D-optimal approximate allocation for generalized linear model and multinomial logit model (see Section 3).	`liftone_constrained_GLM()`, `liftone_constrained_MLM()`
	Transferring approximate allocation to exact allocation (see Section 3).	`approxtoexact_constrained_func()` `approxtoexact_func()`
	Finding constrained uniform exact allocation for bounded constraint (see Section 3.1).	`bounded_uniform()`
Example Specific	Finding I set for exact allocation conversion in `trial_data` and `trauma_data` examples (see Section 3 and Section S4 in Supplementary Material).	`iset_func_trauma()`, `iset_func_trial()`

Table 1: Functions and corresponding usages in the CDsampling package.

generated by the logistic regression model

$$\text{logit}\{P(Y_{ij} = 1 \mid x_{gender\_i}, x_{age\_1i}, x_{age\_2i})\} = \beta_0 + \beta_1 x_{gender\_i} + \beta_{21} x_{age\_1i} + \beta_{22} x_{age\_2i} \quad (3)$$

with $(\beta_0, \beta_1, \beta_{21}, \beta_{22}) = (0, 3, 3, 3)$, where $i = 1, \ldots, 6$ stands for the ith covariate combination, and $j = 1, \ldots, N_i$ is an index of patients who fall into the ith covariate combination or sampling subgroups. Figure 4 illustrates the distribution of treatment efficacy across different gender and age groups.

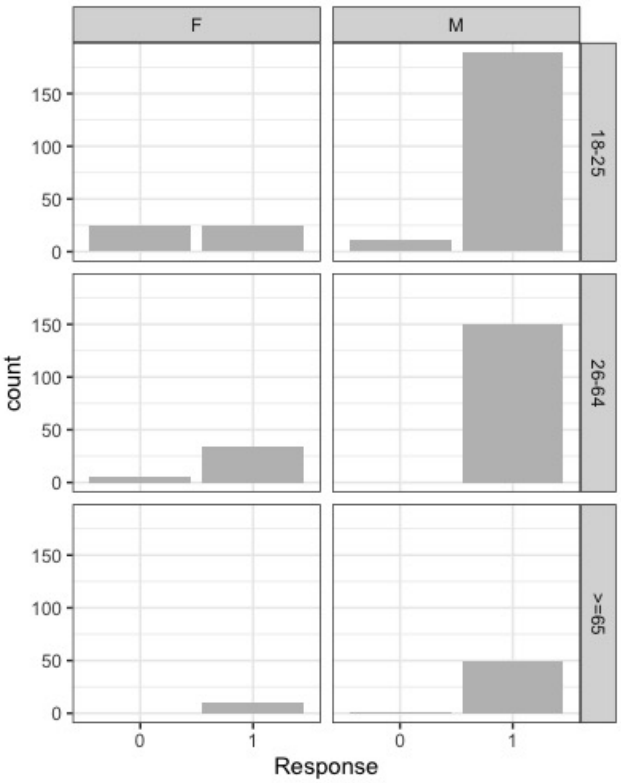


Figure 4: The number of patients from different gender (F, M) and age groups ($18-25$, $26-64$, and ≥ 65) and their responses (0 indicating ineffectiveness, 1 indicating effectiveness) to treatment in `trial_data` of CDsampling package.

In this example, it is posited that a sample of $n = 200$ participants is desired from the population of $N = 500$ volunteers due to budget constraints. The objective is to examine the variation in efficacy rates across gender and age demographics. Should a pilot study or relevant literature provide approximate values for the model parameters, a constrained lift-one algorithm may be employed to find a locally D-optimal design. Conversely, if only partial parameter information is available, the expectation can be deduced from some prior distributions, and the constrained lift-one algorithm can be utilized to determine an EW D-optimal allocation (substituting $\mathbf{W}$ in the Fisher information matrix (1) with $E(\mathbf{W})$).

There are $m = 6$ design points, corresponding to gender and age group combinations $(x_{gender\_i}, x_{age\_1i}, x_{age\_2i}) = (0,0,0), (0,1,0), (0,0,1), (1,0,0), (1,1,0)$, and $(1,0,1)$, respectively. The numbers of available volunteers in the six categories are $(N_1, N_2, \ldots, N_6) = (50, 40, 10, 200, 150, 50)$. Our goal is to find the constrained D-optimal allocation $(w_1, w_2, \ldots, w_6)$ in the set of all feasible allocations $S = \{(w_1, \ldots, w_m)^T \in S_0 \mid nw_i \leq N_i, i = 1, \ldots, m\}$.

We consider the logistic regression model (3), to find the locally D-optimal design, we assume, for illustrative purposes, that the model parameters $(\beta_0, \beta_1, \beta_{21}, \beta_{22}) = (0, 3, 3, 3)$. We may define the parameters and the model matrix as follows. Subsequently, we find the $\mathbf{W}$ matrix in (1) for the Fisher information matrix $\mathbf{F}$.

```
> beta = c(0, 3, 3, 3) #coefficients
> #design matrix
> X=matrix(data=c(1,0,0,0,1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,0,1,1,0,1), ncol=4, byrow=TRUE)
> W=W_func_GLM(X=X, b=beta, link="logit") #find W as input of constrained liftone
```

To define the number of design points, sample size, and constraints with S, we use the following R codes (see Section S3 in the Supplementary Material of Huang et al. (2025) for details on finding r_{i1} and r_{i2} in step 3 of the constrained liftone algorithm in Figure 2):

```
> rc = c(50, 40, 10, 200, 150, 50)/200 #available volunteers/sample size
> m = 6 #design points

> g.con = matrix(0,nrow=(2*m+1), ncol=m) #constraints
> g.con[1,] = rep(1, m)
> g.con[2:(m+1),] = diag(m)
> g.con[(m+2):(2*m+1), ] = diag(m)
> g.dir = c("==", rep("<=", m), rep(">=", m)) #direction
> g.rhs = c(1, rc, rep(0, m)) #righ-hand side

> #lower bound in step 3 of constrained liftone
> lower.bound=function(i, w){
  +   rc = c(50, 40, 10, 200, 150, 50)/200
  +   m=length(w)
  +   temp = rep(0,m)
  +   temp[w>0]=1-pmin(1,rc[w>0])*(1-w[i])/w[w>0];
  +   temp[i]=0;
  +   max(0,temp);
+   }

> #upper bound in step 3 of constrained liftone
> upper.bound=function(i, w){
  +   rc = c(50, 40, 10, 200, 150, 50)/200
  +   min(1,rc[i]);
+   }
```

To identify the subgroups of the output D-optimal allocations, we may add an optional label for each of the $m = 6$ covariatres combination or subgroups as "F, 18-25", "F, 26-64", "F, >=65", "M, 18-2", "M, 26-6", "M, >=65" using the following codes:

```
> label = c("F, 18-25", "F, 26-64", "F, >=65", "M, 18-25", "M, 26-64", "M, >=65")
```

Then, we run the constrained lift-one algorithm with `liftone_constrained_GLM()` to find the constrained D-optimal approximate allocation.

```
> set.seed(092)
> approximate_design = liftone_constrained_GLM(X=X, W=W, g.con=g.con, g.dir=g.dir,
  + g.rhs=g.rhs, lower.bound=lower.bound, upper.bound=upper.bound, reltol=1e-10,
+ maxit=100, random=TRUE, nram=4, w00=NULL, epsilon=1e-8, label=label)
```

The design output is presented below:

```
> print(approximate_design)

Optimal Sampling Results:
================================================================================
Optimal approximate allocation:
F, 18-25 F, 26-64 F, >=65 M, 18-25 M, 26-64 M, >=65
w  0.25     0.2      0.05    0.5      0.0      0.0
w0 0.25     0.0      0.05    0.7      0.0      0.0
--------------------------------------------------------------------------------
maximum :
2.8813e-08
--------------------------------------------------------------------------------
convergence :
TRUE
--------------------------------------------------------------------------------
itmax :
9.0
--------------------------------------------------------------------------------
deriv.ans :
0.0, 3.6017e-08, 4.8528e-07, -1.1525e-07, -1.0310e-07, -7.9507e-08
--------------------------------------------------------------------------------
gmax :
0.0
--------------------------------------------------------------------------------
reason :
"gmax <= 0"
--------------------------------------------------------------------------------
```

The output includes several key components:

- **w**: reports the D-optimal approximate allocation.
- $\mathbf{w}_0$: reports the random initial approximate allocation used to initialize optimization.
- **maximum**: reports the maximum determinant of Fisher information matrix.
- **reason**: reports the termination criterion for the constrained lift-one algorithm including either "all derivative ≤ 0" or "gmax ≤ 0", which corresponds to step 7° and step 8° in the constrained lift-one algorithm in Figure 2.

In practical terms, exact allocations are more beneficial. One may use the constrained approximate to exact allocation algorithm depicted in Figure 3, which is implemented as the `approxtoexact_constrained_func()` function.

```
> exact_design = approxtoexact_constrained_func(n=200, w=approximate_design$w, m=6,
  + beta=beta, link='logit', X=X, Fdet_func=Fdet_func_GLM, iset_func=iset_func_trial,
```

```
+ label=label)
> print(exact_design)

Optimal Sampling Results:
================================================================================
Optimal exact allocation:
F, 18-25 F, 26-64 F, >=65 M, 18-25 M, 26-64 M, >=65
allocation      50.0     40.0     10.0    100.0    0.0      0.0
allocation.real 0.25     0.2      0.05    0.5      0.0      0.0
--------------------------------------------------------------------------------
det.maximum :
46.1012
--------------------------------------------------------------------------------
```

The output provides three key components for the sampling results:

- **allocation**: reports the exact allocation of D-optimal sampling, specifying the number of subjects to sample from each subgroup.
- **allocation.real**: reports the real-number approximate allocation used prior to integer conversion.
- **det.maximum**: reports the maximum determinant of the Fisher information matrix by the optimal design.

In this example, the D-optimal exact allocation is to sample 50 subjects from the "female, $18-25$" subgroup, 40 subjects from the "female, $26-64$" subgroup, 10 subjects from the "female, ≥ 65" subgroup, and 100 subjects from the "male, $18-25$" subgroup. Such a design may not explore all the design space and may lead to extreme design cases. In practice, allocating some subjects to the omitted subgroups "male, $26-64$" and "male, ≥ 65" could improve robustness and reduce the risk of overfitting.

Alternatively, one may aim for EW D-optimal allocations when partial coefficient information is available with the **W** matrix substituted by the expectation (EW stands for the expected weight). To calculate these expectations, one may define prior distributions for the parameters based on available information. For instance, in this scenario, we assume the following independent prior distributions: $\beta_0 \sim \text{uniform}(-2,2)$, $\beta_1 \sim \text{uniform}(-1,5)$, $\beta_{21} \sim \text{uniform}(-1,5)$, and $\beta_{22} \sim$ uniform(-1,5). Subsequently, the diagonal elements of **W** are determined through integration. For $i = 1,\ldots,m$ and $\eta_i = \beta_0 + x_{gender\_i}\beta_1 + x_{age\_i1}\beta_{21} + x_{age\_i2}\beta_{22}$, we calculate the key component $E(v_i)$ of the ith diagonal element of **W** through:

$$\int_{-1}^{5}\int_{-1}^{5}\int_{-1}^{5}\int_{-2}^{2} \frac{\exp(\eta_i)}{(1+\exp(\eta_i))^2}\Pr(\beta_0)\Pr(\beta_1)\Pr(\beta_{21})\Pr(\beta_{22})\, d\beta_0 d\beta_1 d\beta_{21} d\beta_{22}$$

where $\Pr(\cdot)$ stands for the corresponding probability density function. We use the `hcubature()` function in the **cubature** package to calculate the integration as illustrated by the R codes below.

```
> unif.prior <- rbind(c(-2, -1, -1, -1), c(2,  5,  5, 5)) #prior parameters

> #find expectation of W matrix given priors
> W.EW.unif = matrix(rep(0,6))
> for (i in 1:6){
  +  x = X[i,]
  +  W.EW.unif[i] = hcubature(function(beta) dunif(beta[1], min=unif.prior[1,1],
    max=unif.prior[2,1])*dunif(beta[2], min=unif.prior[1,2], max=unif.prior[2,2])*
    dunif(beta[3], min=unif.prior[1,3], max=unif.prior[2,3])*dunif(beta[4],
    min=unif.prior[1,4], max=unif.prior[2,4])*(exp(x[1]*beta[1]+x[2]*beta[2]+x[3]*
```

```
    beta[3]+x[4]*beta[4])/(1+exp(x[1]*beta[1]+x[2]*beta[2]+x[3]*beta[3]+x[4]*beta[4]))^2),
  lowerLimit = unif.prior[1,], upperLimit  = unif.prior[2,])$integral
+  }
```

Given the expectation of **W**, the functions `liftone_constrained_GLM()` and `approxtoexact_constrained_func()` are used for deriving the constrained EW D-optimal approximate allocation and the corresponding exact allocation, respectively. This process follows a similar procedure to that used for local D-optimal approximate allocation.

```
> set.seed(123)
> approximate_design_EW = liftone_constrained_GLM(X=X, W=W.EW.unif,  g.con=g.con,
  + g.dir=g.dir, g.rhs=g.rhs, lower.bound=lower.bound, upper.bound=upper.bound,
+ reltol=1e-12, maxit=100, random=TRUE, nram=4, w00=NULL, epsilon=1e-10, label=label)

> exact_design = approxtoexact_constrained_func(n=200,
  + w=approximate_design_EW$w, m=6, beta=beta, link='logit', X=X, Fdet_func=Fdet_func_GLM,
+ iset_func=iset_func_trial, label=label)
```

The output is summarized with `print()` function and presented below:

```
> print(exact_design_EW)

Optimal Sampling Results:
================================================================================
Optimal exact allocation:
F, 18-25 F, 26-64 F, >=65 M, 18-25 M, 26-64 M, >=65
allocation      48.0     40.0     10.0    43.0     19.0     40.0
allocation.real 0.2406   0.2      0.05    0.2102   0.0991   0.2001
--------------------------------------------------------------------------------
det.maximum :
25.59
--------------------------------------------------------------------------------
```

In situations when the model parameters are unknown, the constrained uniform allocation is applicable. This method entails sampling an equal number of patients from each category within the given constraints. The selection criterion is $n_i = \min\{k, N_i\}$ or $\min\{k, N_i\} + 1$ with k satisfying $\sum_{i=1}^{m} \min\{k, N_i\} \leq n < \sum_{i=1}^{m} \min\{k+1, N_i\}$, where N_i represents the maximum allowable number for each category. This is an example of a bounded design problem, where each category has an upper boundary. The function `bounded_uniform()` can be used to find the constrained uniform allocation with the `trial_data` example and "**allocation**" in the output representing the constrained uniform allocation.

```
> bounded_uniform(Ni=c(50, 40, 10, 200, 150, 50), nsample=200, label=label)

Optimal Sampling Results:
================================================================================
Optimal exact allocation:
F, 18-25 F, 26-64 F, >=65 M, 18-25 M, 26-64 M, >=65
allocation 38.0     38.0     10.0    38.0     38.0     38.0
--------------------------------------------------------------------------------
```

Alternatively, we may also use `approxtoexact_constrained_func()` to find the same constrained uniform exact allocation. This function can be used under fairly general constraints. To find the constrained uniform exact allocation using `approxtoexact_constrained_func()`, we suggest starting with one subject in each stratum or subgroup, which corresponds to the approximate allocation $\mathbf{w_{00}} = (1/200, 1/200, 1/200, 1/200, 1/200, 1/200)$ in this case.

```
> w00 = rep(1/200, 6) #initial approximate allocation
> unif_design = approxtoexact_constrained_func(n=200, w=w00, m=6, beta=NULL,
+ link=NULL, X=NULL, Fdet_func=Fdet_func_unif, iset_func=iset_func_trial, label=label)

> print(unif_design)

Optimal Sampling Results:
================================================================================
Optimal exact allocation:
F, 18-25 F, 26-64 F, >=65 M, 18-25 M, 26-64 M, >=65
allocation      38.0     38.0     10.0    38.0     38.0     38.0
allocation.real 0.005    0.005    0.005   0.005    0.005    0.005
--------------------------------------------------------------------------------
det.maximum :
792351680.0
--------------------------------------------------------------------------------
```

The `iset_func_trial()` function in the **CDsampling** package is specifically designed for the `trial_data` example, which defines the set I in step 1° and step 2.4 in the constrained approximate to exact algorithm depicted by Figure 3. This function serves as a template that users can adapt to their specific constraints by modifying the codes. The package includes two such template functions: `iset_func_trial()` and `iset_func_trauma()` with details provided in Section S4 of Supplementary Material.

To perform a comparison analysis on different sampling strategies, including the constrained D-optimal allocation, the constrained EW D-optimal allocation with uniform priors, the constrained uniform allocation, simple random sample without replacement (SRSWOR), as well as the full data (all the 500 patients enrolled), we simulate their responses Y_{ij}'s based on model (3) with parameter values $(0, 3, 3, 3)$. We apply each sampling strategy to obtain a sample of $n = 200$ observations out of 500, and estimate the parameters using the 200 observations. The exception is the full data method, where estimation is performed using all 500 patients. We use the root mean square error (RMSE) to measure the accuracy of the estimates (see Section 4 of Huang et al. (2025) for more theoretical and technical details). We repeat the procedure 100 times and display the corresponding RMSEs in Figure 5 and simulation codes in Supplementary Material Section S3 (Wickham, 2016; Wickham et al., 2023). Obviously, if we use the full data (all 500 patients) to fit the model, its RMSE attains the lowest. Besides that, the constrained locally D-optimal allocation and the constrained EW D-optimal allocation have a little higher RMSEs than the full data estimates but outperform SRSWOR and the constrained uniform allocation. The sampling strategy based on the constrained uniform allocation doesn't need any model information and is a more robust sampling scheme, which is still better than SRSWOR.

3.2 Applications in paid research study: `trauma_data` example

In the **CDsampling** package, `trauma_data` is a dataset of $N = 802$ trauma patients from Chuang-Stein and Agresti (1997), stratified according to the trauma severity at the entry time of the study with 392 mild and 410 moderate/severe patients enrolled. The study involved four treatment groups determined by the `dose` level, $x_{i1} = 1$ (`Placebo`), 2 (`Low dose`), 3 (`Medium dose`), and 4 (`High dose`). Combining with `severity` grade ($x_{i2} = 0$ for `mild` or 1 for `moderate/severe`), there are $m = 8$ distinct experimental settings with $(x_{i1}, x_{i2}) = (1,0), (2,0), (3,0), (4,0), (1,1), (2,1), (3,1), (4,1)$, respectively. The responses belong to five ordered categories, `Death` (1), `Vegetative state` (2), `Major disability` (3), `Minor disability` (4) and `Good recovery` (5), known as the Glasgow Outcome Scale (Jennett and Bond, 1975). Figure 6 shows the distribution of outcomes over `severity` grades and `dose` levels. In this example, we have $m = 8$ subgroups, which are combinations of the two covariates categories: `dose` levels and `severity` grades. We aim to enroll $n = 600$

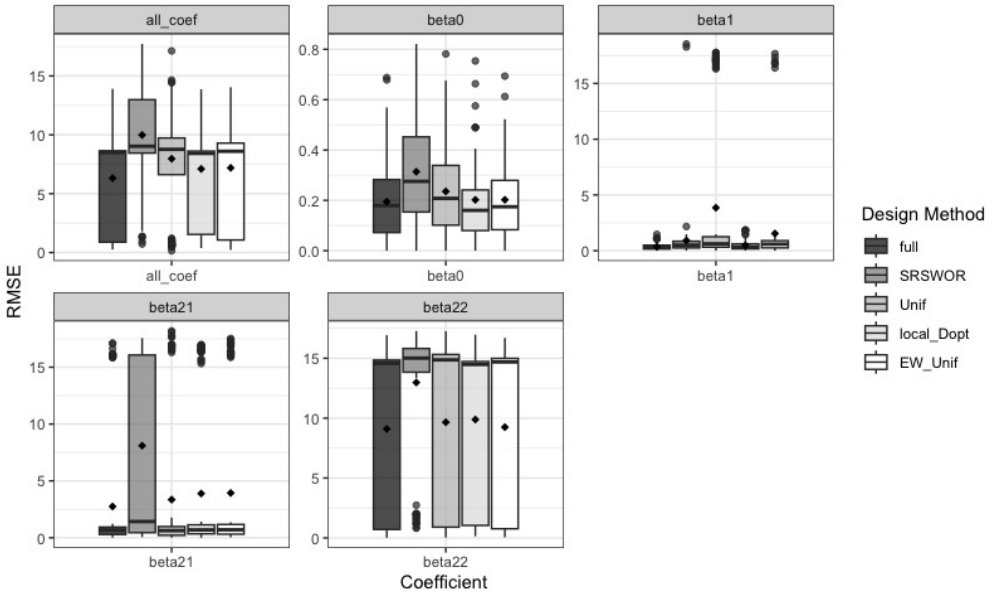


Figure 5: Boxplots of RMSEs obtained from 100 simulations using full data (full), SRSWOR, constrained uniform design (Unif), the constrained locally D-optimal allocation (local_Dopt), and constrained EW D-optimal allocation with uniform priors (EW_Unif), with black diamonds representing average RMSE.

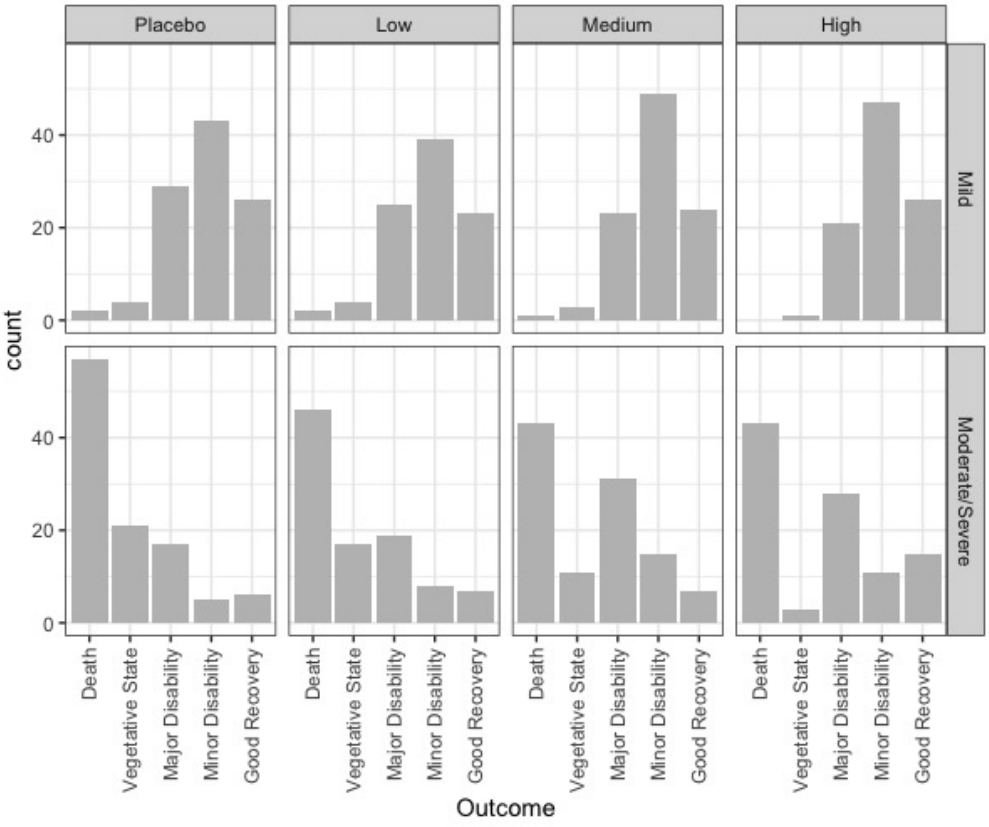


Figure 6: The number of patients from different `dose` levels (Placebo, Low, Medium, and High) and `severity` grades (Mild, Moderate/Severe) groups and their treatment outcomes in `trauma_data` of **CDsampling** package.

patients from the 802 available patients. The collection of feasible allocations is inherently

constrained by the number of patients in different severity grades, defined as

$$S = \{(w_1, \ldots, w_8)^\top \in S_0 \mid n(w_1 + w_2 + w_3 + w_4) \leq 392,\ \ n(w_5 + w_6 + w_7 + w_8) \leq 410\}.$$

The constraints specify that in the sample, the number of patients with mild symptoms must not exceed 392 across all dose levels, while those with moderate/severe symptoms must not exceed 410.

The parameters fitted from the `trauma_data` are

$$\begin{aligned}\boldsymbol{\beta} &= (\hat{\beta}_{11}, \hat{\beta}_{12}, \hat{\beta}_{13}, \hat{\beta}_{21}, \hat{\beta}_{22}, \hat{\beta}_{23}, \hat{\beta}_{31}, \hat{\beta}_{32}, \hat{\beta}_{33}, \hat{\beta}_{41}, \hat{\beta}_{42}, \hat{\beta}_{43})^\top \\ &= (-4.047, -0.131, 4.214, -2.225, -0.376, 3.519, -0.302, -0.237, 2.420, 1.386, -0.120, 1.284)^\top.\end{aligned}$$

The model can be written in the following format:

$$\begin{aligned}\log\left(\frac{\pi_{i1}}{\pi_{i2} + \cdots + \pi_{i5}}\right) &= \beta_{11} + \beta_{12}x_{i1} + \beta_{13}x_{i2} \\ \log\left(\frac{\pi_{i1} + \pi_{i2}}{\pi_{i3} + \pi_{i4} + \pi_{i5}}\right) &= \beta_{21} + \beta_{22}x_{i1} + \beta_{23}x_{i2} \\ \log\left(\frac{\pi_{i1} + \pi_{i2} + \pi_{i3}}{\pi_{i4} + \pi_{i5}}\right) &= \beta_{31} + \beta_{32}x_{i1} + \beta_{33}x_{i2} \\ \log\left(\frac{\pi_{i1} + \cdots + \pi_{i4}}{\pi_{i5}}\right) &= \beta_{41} + \beta_{42}x_{i1} + \beta_{43}x_{i2}\end{aligned}$$

where $i = 1, \ldots, 8$.

We use the R codes below to define the model matrix and coefficients.

```
> J=5; p=12; m=8; #response levels; parameters; subgroups

> #coefficients
> beta = c(-4.047, -0.131, 4.214, -2.225, -0.376, 3.519, -0.302, -0.237,  2.420, 1.386,
+ -0.120,  1.284)

> #define design matrix of 8 subgroups
> Xi=rep(0,J*p*m); dim(Xi)=c(J,p,m);
> Xi[,,1] = rbind(c( 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  +   c( 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0), c( 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0),
+   c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0), c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

> Xi[,,2] = rbind(c( 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  +   c( 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0), c( 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0),
+   c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0), c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

> Xi[,,3] = rbind(c( 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  +   c( 0, 0, 0, 1, 3, 0, 0, 0, 0, 0, 0, 0), c( 0, 0, 0, 0, 0, 0, 1, 3, 0, 0, 0, 0),
+   c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 0), c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

> Xi[,,4] = rbind(c( 1, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  +   c( 0, 0, 0, 1, 4, 0, 0, 0, 0, 0, 0, 0), c( 0, 0, 0, 0, 0, 0, 1, 4, 0, 0, 0, 0),
+   c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 0), c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

> Xi[,,5] = rbind(c( 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  +   c( 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0), c( 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0),
+   c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1), c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

> Xi[,,6] = rbind(c( 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  +   c( 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0), c( 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0),
```

```
+   c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1), c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

> Xi[,,7] = rbind(c( 1, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  +   c( 0, 0, 0, 1, 3, 1, 0, 0, 0, 0, 0, 0), c( 0, 0, 0, 0, 0, 0, 1, 3, 1, 0, 0, 0),
+   c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1), c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

> Xi[,,8] = rbind(c( 1, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
  +   c( 0, 0, 0, 1, 4, 1, 0, 0, 0, 0, 0, 0), c( 0, 0, 0, 0, 0, 0, 1, 4, 1, 0, 0, 0),
+   c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 1), c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))
```

To define the sample size, the constraints, and the functions of lower and upper boundaries r_{i1} and r_{i2}, we may use the following R codes (see Section S3 in the Supplementary Material of Huang et al. (2025) for details on finding r_{i1} and r_{i2}):

```
> nsample=600 #sample size
> constraint = c(392, 410)  #mild:severe

> #lower bound function in step 3 of constrained liftone
> lower.bound <- function(i, w0){
  +   n = 600
  +   constraint = c(392,410)
  +   if(i <= 4){
  +     a.lower <- (sum(w0[5:8])-(constraint[2]/n)*(1-w0[i]))/(sum(w0[5:8]))}
  +   else{
  +     a.lower <- (sum(w0[1:4])-(constraint[1]/n)*(1-w0[i]))/(sum(w0[1:4]))}
+    a.lower}

> #upper bound function in step 3 of constrained liftone
> upper.bound <- function(i, w0){
  +  n = 600
  +  constraint = c(392,410)
  +  if(i <= 4){
  +    b.upper <- ((constraint[1]/n)*(1-w0[i]) - (sum(w0[1:4])-w0[i]))/(1-sum(w0[1:4]))}
  +  else{
  +    b.upper <- ((constraint[2]/n)*(1-w0[i]) - (sum(w0[5:8])-w0[i]))/(1-sum(w0[5:8]))}
+    b.upper}

> #define constraints
> g.con = matrix(0,nrow=length(constraint)+1+m, ncol=m)
> g.con[2:3,] = matrix(data=c(1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1), ncol = m, byrow=TRUE)
> g.con[1,] = rep(1, m)
> g.con[4:(length(constraint)+1+m), ] = diag(1, nrow=m)
> g.dir = c("==", "<=","<=", rep(">=",m))
> g.rhs = c(1, ifelse((constraint/nsample<1),constraint/nsample,1), rep(0, m))
```

Then, we may define an optional label of the sampling subgroups that corresponds to each of the $m = 8$ subgroups using the following code:

```
> label=label = c("Placebo-Mild", "Low-Mild", "Medium-Mild", "High-Mild",
"Placebo-Severe", "Low-Severe", "Medium-Severe", "High-Severe")
```

We then run the constrained lift-one algorithm to find the constrained D-optimal approximate allocation using `liftone_constrained_MLM()` function and convert the approximate allocation to an exact allocation with `approxtoexact_constrained_func` function.

```
> set.seed(123)
> approx_design = liftone_constrained_MLM(m=m, p=p, Xi=Xi, J=J, beta=beta,
```

```
  + lower.bound=lower.bound, upper.bound=upper.bound, g.con=g.con, g.dir=g.dir,
  + g.rhs=g.rhs, w00=NULL, link='cumulative',  Fi.func=Fi_func_MLM, reltol=1e-5,
+ maxit=500, delta=1e-6, epsilon=1e-8, random=TRUE, nram=3, label=label)

> exact_design = approxtoexact_constrained_func(n=600, w=approx_design$w, m=8,
  + beta=beta, link='cumulative', X=Xi, Fdet_func=Fdet_func_MLM,
+ iset_func=iset_func_trauma, label=label)

> print(exact_design)

Optimal Sampling Results:
================================================================================
Optimal exact allocation:
Placebo-Mild Low-Mild Medium-Mild High-Mild Placebo-Severe
allocation      155.0        0.0      0.0         100.0     168.0
allocation.real 0.2593       0.0      0.0         0.1667    0.2796
Low-Severe Medium-Severe High-Severe
allocation      0.0        0.0           177.0
allocation.real 0.0        0.0           0.2944
--------------------------------------------------------------------------------
det.maximum :
1.63163827059162e+23
--------------------------------------------------------------------------------
```

The **allocation** output provides the exact allocation of the sampling across different treatment-severity subgroups, representing the implementable sample sizes for each subgroup. The result is derived by converting the **allocation.real**, which is the D-optimal approximate allocation outcome from `liftone_constrained_MLM()`.

As the `trauma_data` example doesn't have bounded constraints, to find the constrained uniform sampling allocation, we use `approxtoexact_constrained_func()` with one subject in each stratum or subgroup as the input, that is, the approximate allocation $\mathbf{w} = (1/600, 1/600, 1/600, 1/600, 1/600, 1/600, 1/600, 1/600)$. The corresponding I set function is provided in the **CDsampling** package, and it can be easily defined according to other constraints, see Section S4 of Supplementary Material. Note that the determinant provided by `approxtoexact_constrained_func()` for different designs is not comparable, as the criteria `Fdet_func` differs.

```
> unif_design = approxtoexact_constrained_func(n=600, w=rep(1/600,8), m=8,
+ beta=NULL, link=NULL, X=NULL, Fdet_func=Fdet_func_unif, iset_func=iset_func_trauma)

> print(unif_design)

Optimal Sampling Results:
================================================================================
Optimal exact allocation:
Placebo-Mild Low-Mild Medium-Mild High-Mild Placebo-Severe Low-Severe
allocation      75.0         75.0     75.0        75.0      75.0           75.0
allocation.real 0.0017       0.0017   0.0017      0.0017    0.0017         0.0017
Medium-Severe High-Severe
allocation      75.0          75.0
allocation.real 0.0017        0.0017
--------------------------------------------------------------------------------
det.maximum :
1001129150390625
--------------------------------------------------------------------------------
```

4 Summary

The current version of **CDsampling** implements D-optimal allocations within both paid research sampling and general study frameworks with or without constraints. Its primary objective is to optimize sampling allocations for better model estimation accuracy in the studies. The package includes `F_func_GLM()` and `F_func_MLM()` for the computation of the Fisher information matrix of GLMs and MLMs, respectively. It is noteworthy that standard linear regression models are special GLMs with an identity link function and Gaussian-distributed responses, which is also supported by our package. Theoretical results are summarized in Section 2.2 and Section 2.3 while illustrative examples are provided in Supplementary Section S1.

To find standard or unconstrained D-optimal allocations, our package implements the lift-one algorithm through functions `liftone_GLM()` and `liftone_MLM()`. Paid research studies often impose sampling constraints. To address this, the constrained lift-one algorithm can be applied using functions `liftone_constrained_GLM()` and `liftone_constrained_MLM()`. An example illustrating the difference between the lift-one algorithm and the constrained lift-one algorithm is provided in Supplementary Section S2 while Section 3 presents two application examples from paid research studies.

In the absence of model information, `constrained_uniform()` function is available to find a robust constrained uniform allocation with bounded constraints, while the `approxtoexact_constrained_func()` function can be used to find constrained uniform allocation with more general constraints. For transitioning from approximate to exact allocations, the package provides `approxtoexact_constrained_func()` for constrained cases and `approxtoexact_func()` for unconstrained cases. Detailed applications for both GLMs and MLMs are provided in Sections 3.1 and 3.2.

Future enhancements of the package may aim to incorporate a broader spectrum of optimality criteria, such as A-optimality and E-optimality, as well as some models beyond GLMs and MLMs to expand its applicability.

References

A. Agresti. *Categorical Data Analysis*. Wiley, 3 edition, 2013. doi: 10.1007/s00362-015-0733-8. [p165]

A. Atkinson, A. Donev, and R. Tobias. *Optimum Experimental Designs, with SAS*. Oxford University Press, 2007. doi: 10.1093/oso/9780199296590.001.0001. [p161, 162]

G. Barcaroli. Samplingstrata: An R package for the optimization of stratified sampling. *Journal of Statistical Software*, 61:1–24, 2014. doi: 10.18637/jss.v061.i04. [p161]

X. Bu, D. Majumdar, and J. Yang. D-optimal designs for multinomial logistic models. *Annals of Statistics*, 48(2):983–1000, 2020. doi: 10.1214/19-AOS1834. [p161, 165, 166]

C. Chuang-Stein and A. Agresti. Tutorial in biostatistics-a review of tests for detecting a monotone dose-response relationship with ordinal response data. *Statistics in Medicine*, 16:2599–2618, 1997. [p172]

A. Dobson and A. Barnett. *An Introduction to Generalized Linear Models*. Chapman & Hall/CRC, 4 edition, 2018. doi: 10.1201/9781315182780. [p164]

N. Dousti Mousavi, H. Aldirawi, and J. Yang. Categorical data analysis for high-dimensional sparse gene expression data. *BioTech*, 12(3):52, 2023. doi: 10.3390/biotech12030052. [p165]

V. Fedorov. *Theory of Optimal Experiments*. Academic Press, 1972. ISBN 0122507509. [p161]

V. Fedorov and S. Leonov. *Optimal Design for Nonlinear Response Models*. Chapman & Hall/CRC, 2014. doi: 10.1201/b15054. [p161]

G. Glonek and P. McCullagh. Multivariate logistic models. *Journal of the Royal Statistical Society, Series B*, 57:533–546, 1995. URL https://www.jstor.org/stable/2346155. [p165]

U. Grömping. Optimal experimental design with R. *Journal of Statistical Software*, 43:1–4, 2011. doi: 10.18637/jss.v043.b05. [p161]

R. Harman, L. Filova, and M. L. Filova. Package 'optimaldesign', 2016. URL https://cran.r-project.org/package=OptimalDesign. [p161]

Y. Huang, L. Tong, and J. Yang. Constrained d-optimal design for paid research study. *Statistica Sinica*, 2025. doi: 10.5705/ss.202022.0414. To appear. [p161, 163, 165, 168, 172, 175]

B. Jennett and M. Bond. Assessment of outcome after severe brain damage. *Lancet*, 305: 480–484, 1975. doi: 10.1016/S0140-6736(75)92830-5. [p172]

A. Khuri, B. Mukherjee, B. Sinha, and M. Ghosh. Design issues for generalized linear models: A review. *Statistical Science*, 21:376–399, 2006. doi: 10.1214/088342306000000105. [p165]

J. Kiefer. General equivalence theory for optimum designs (approximate theory). *Annals of Statistics*, 2:849–879, 1974. doi: 10.1214/aos/1176342810. [p162]

P. S. Levy and S. Lemeshow. *Sampling of populations: methods and applications*. Wiley, Hoboken, N.J, 4th ed. edition, 2008. ISBN 9780470374597. doi: 10.1002/9780470374597. [p161]

S. Lohr. *Sampling: Design and Analysis*. Chapman and Hall/CRC, 2019. doi: 10.1201/9780429298899. [p160]

P. McCullagh and J. Nelder. *Generalized Linear Models*. Chapman and Hall/CRC, 2 edition, 1989. doi: 10.1007/978-1-4899-3242-6. [p164, 165]

J. A. Nelder and R. W. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 135(3):370–384, 1972. doi: 10.2307/2344614. [p164]

A. M. Overstall, D. C. Woods, M. Adamou, and D. Michaelides. Package 'acebayes'. 2018. URL https://cran.r-project.org/package=acebayes. [p161]

A. M. Overstall, D. C. Woods, and M. Adamou. acebayes: An R package for Bayesian optimal design of experiments via approximate coordinate exchange. *Journal of Statistical Software*, 95(13):1–33, 2020. doi: 10.18637/jss.v095.i13. URL https://www.jstatsoft.org/index.php/jss/article/view/v095i13. [p161]

F. Pukelsheim. *Optimal Design of Experiments*. SIAM, Philadelphia, 2006. doi: 10.1137/1.9780898719109. [p162]

J. Stufken and M. Yang. Optimal designs for generalized linear models. In K. Hinkelmann, editor, *Design and Analysis of Experiments, Volume 3: Special Designs and Applications*, chapter 4, pages 137–164. Wiley, 2012. doi: 10.1002/9781118147634. [p165]

Y. Tillé. *Sampling Algorithms*. Springer, 2006. doi: 10.1007/0-387-34240-0. [p160]

Y. Tillé and A. Matei. Package 'sampling'. *Surv. Sampling. Kasutatud*, 23:2017, 2016. URL https://cran.r-project.org/package=sampling. [p161]

T. Wang and J. Yang. Identifying the most appropriate order for categorical responses. *Statistica Sinica*, 35:411–430. doi: 10.25417/uic.25599399.v1. [p165]

B. Wheeler and M. J. Braun. Package 'algdesign'. *R Proj. Stat. Comput*, 1(0):1–25, 2019. URL https://cloud.r-project.org/web/packages/AlgDesign/AlgDesign.pdf. [p161]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p172]

H. Wickham, R. François, L. Henry, K. Müller, and D. Vaughan. *dplyr: A Grammar of Data Manipulation*, 2023. URL https://dplyr.tidyverse.org. R package version 1.1.4, https://github.com/tidyverse/dplyr. [p172]

J. Yang and A. Mandal. D-optimal factorial designs under generalized linear models. *Communications in Statistics - Simulation and Computation*, 44:2264–2277, 2015. doi: 10.1080/03610918.2013.815773. [p161]

J. Yang, A. Mandal, and D. Majumdar. Optimal designs for 2^k factorial experiments with binary response. *Statistica Sinica*, 26:385–411, 2016. doi: 10.5705/ss.2013.265. [p161, 165]

J. Yang, L. Tong, and A. Mandal. D-optimal designs with ordered categorical data. *Statistica Sinica*, 27:1879–1902, 2017. doi: 10.5705/ss.202016.0210. [p161]

M. Yang and J. Stufken. Identifying locally optimal designs for nonlinear models: A simple extension with profound consequences. *Annals of Statistics*, 40:1665–1681, 2012. doi: 10.1214/12-AOS992. [p161]

S. Zocchi and A. Atkinson. Optimum experimental designs for multinomial logistic models. *Biometrics*, 55:437–444, 1999. doi: 10.1111/j.0006-341X.1999.00437.x. [p161, 165]

Yifei Huang
Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
E-mail: yhuan39@uic.edu

Liping Tong
Advocate Aurora Health
E-mail: liping.tong@aah.org

Jie Yang
Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
E-mail: jyang06@uic.edu

panelPomp: Analysis of Panel Data via Partially Observed Markov Processes in R

by Carles Bretó, Jesse Wheeler, Aaron A. King, and Edward L. Ionides

Abstract Panel data arise when time series measurements are collected from multiple, dynamically independent but structurally related systems. Each system's time series can be modeled as a partially observed Markov process (POMP), and the ensemble of these models is called a PanelPOMP. If the time series are relatively short, statistical inference for each time series must draw information from across the entire panel. The component systems in the panel are called units; model parameters may be shared between units or may be unit-specific. Differences between units may be of direct inferential interest or may be a nuisance for studying the commonalities. The R package panelPomp supports analysis of panel data via a general class of PanelPOMP models. This includes a suite of tools for manipulation of models and data that take advantage of the panel structure. The panelPomp package currently highlights recent advances enabling likelihood based inference via simulation based algorithms. However, the general framework provided by panelPomp supports development of additional, new inference methodology for panel data.

1 Introduction

Collections of time series, known as panel data or longitudinal data, are commonplace across the sciences, medicine, engineering, and business. Examples include biomarkers measured over time across a panel of patients in a clinical trial (Ranjeva et al., 2017, 2019), ecological predator-prey dynamics for lake plankton measured within a season across a panel of years or lakes (Marino et al., 2019), and interactions between self-driving cars and pedestrians (Domeyer et al., 2022). Generically, we say each time series is associated with measurements on a unit. Not infrequently, the time series data available on a single unit are too short or too noisy to adequately identify parameters of interest, and yet large collections of such time series analyzed jointly may suffice. This paper presents software enabling the fitting of general nonlinear nonstationary partially observed stochastic dynamic models to panel data. The methodology provides flexibility in model specification, giving the user considerable freedom to develop models appropriate for the system under study.

Scientific motivations to fit a panel of partially observed Markov processes (PanelPOMP) model to panel data (Bretó et al., 2020) are similar to the motivations to fit a partially observed Markov process (POMP) model to time series data from a single unit. POMP models provide a general framework for mechanistic modeling of nonlinear dynamic systems (Bretó et al., 2009). However, relatively few of the many methodologies developed for time series analysis via POMPs have been extended to panel analysis. The larger datasets and higher-dimensional parameter spaces arising in PanelPOMPs have challenged the Monte Carlo methods that have proved successful for nonlinear time series. Our **panelPomp** package takes advantage of newly developed methodology for PanelPOMP models, as well as building a foundation on which additional methodology for this class of models can be built and tested.

Existing widely-used panel methodology has built on the linear Gaussian model (Croissant and Millo, 2008). Panel analysis of generalized linear models with dependence can be carried out using generalized estimating equations (Halekoh et al., 2006). By contrast, **panelPomp** is designed to facilitate analysis using arbitrary PanelPOMP models, with dynamic relationships and dependence on covariate processes specified according to scientific considerations rather than the constraints of statistical software. Beyond supplying implementations of inference algorithms for PanelPOMP models, the **panelPomp** package provides a framework for developing and sharing new models and methods. This is done by providing a way of writing basic mathematical functions that comprise a general

PanelPOMP model as easy-to-write C-code snippets (King et al., 2016), which makes the **panelPomp** package both computationally efficient and readily adaptable to new developments in methodology for PanelPOMP models.

Analysis of experimental or observational studies often assumes independent outcomes for units given their treatment, and here we take the same approach for panel models. That is, the underlying dynamic process that is used to define a PanelPOMP model is assumed to be independent across units. A collection of POMP models with dynamic dependence between units is called a SpatPOMP, a name motivated by the dependence between nearby locations in spatiotemporal models. Software for general SpatPOMP models is available in the **spatPomp** package (Asfaw et al., 2024). Addressing SpatPOMP models introduces complexities in both software and methodologies that can be avoided when the dynamic processes are independent. The goal of the **panelPomp** package is to take full advantage of the independence inherent in the PanelPOMP model structure.

2 Statistical models

The general scope of the **panelPomp** package requires notation concerning random variables and their densities in arbitrary spaces. The notation below allows us to talk about these things using the language of mathematics, enabling precise description of models and algorithms.

Units of the panel can be identified with numeric labels $\{1, 2, \dots, U\}$, which we also write as $1:U$. Let N_u be the number of measurements collected on unit u, and write the data as $y^*_{u,1:N_u} = \{y^*_{u,1}, \dots, y^*_{u,N_u}\}$ where $y^*_{u,n}$ is collected at time $t_{u,n}$ with $t_{u,1} < t_{u,2} < \cdots < t_{u,N_u}$. The data are modeled as a realization of an observable stochastic process $Y_{u,1:N_u}$ which is dependent on a latent Markov process $\{X_u(t), t_{u,0} \le t \le t_{u,N_u}\}$ defined subsequent to an initial time $t_{u,0} \le t_{u,1}$. Requiring that $\{X_u(t)\}$ and $\{Y_{u,i}, i \ne n\}$ are independent of $Y_{u,n}$ given $X_u(t_{u,n})$, for each $n \in 1:N_u$, completes the partially observed Markov process (POMP) model structure for unit u. For a PanelPOMP we require additionally that all units are modeled as independent.

The latent process at the observation times is written $X_{u,n} = X_u(t_{u,n})$. We suppose that $X_{u,n}$ and $Y_{u,n}$ take values in arbitrary spaces $\mathbb{X}_u$ and $\mathbb{Y}_u$ respectively. Using the independence of units, conditional independence of the observable random variables, and the Markov property of the latent states, the joint distribution of the entire collection of latent variables $\mathbf{X} = \{X_{u,0:N_u}\}_{u=1}^U$ and observable variables $\mathbf{Y} = \{Y_{u,1:N_u}\}_{u=1}^U$ can be written as

$$f_{\mathbf{XY}}(\mathbf{x}, \mathbf{y}) = \prod_{u=1}^{U} f_{X_{u,0}}(x_{u,0}; \theta) \prod_{n=1}^{N_u} f_{Y_{u,n}|X_{u,n}}(y_{u,n}|x_{u,n}; \theta) f_{X_{u,n}|X_{u,n-1}}(x_{u,n}|x_{u,n-1}; \theta),$$

where $\theta \in \mathbb{R}^D$ is a parameter vector. This representation is useful as it demonstrates how any PanelPOMP model can be fully described using three primary components: the unit transition densities $f_{X_{u,n}|X_{u,n-1}}(x_{u,n} \mid x_{u,n-1}\,;\,\theta)$, measurement densities $f_{Y_{u,n}|X_{u,n}}(y_{u,n} \mid x_{u,n}\,;\,\theta)$, and initialization densities $f_{X_{u,0}}(x_{u,0}; \theta)$. Each class of densities are permitted to depend arbitrarily on u and n, allowing non-stationary models and the inclusion of covariate time series. In addition to continuous-time dynamics, the framework includes discrete-time dynamic models by specifying $X_{u,0:N_u}$ directly without ever defining $\{X_u(t), t_{u,0} \le t \le t_{u,N_u}\}$. We also permit the possibility that some parameters may affect only a subset of units, so that the parameter vector can be written as $\theta = (\phi, \psi_1, \dots, \psi_U)$, where the densities described above can be written as

$$\begin{aligned} f_{X_{u,n}|X_{u,n-1}}(x_{u,n} \mid x_{u,n-1}\,;\,\theta) &= f_{X_{u,n}|X_{u,n-1}}(x_{u,n} \mid x_{u,n-1}\,;\,\phi, \psi_u) && (1)\\ f_{Y_{u,n}|X_{u,n}}(y_{u,n} \mid x_{u,n}\,;\,\theta) &= f_{Y_{u,n}|X_{u,n}}(y_{u,n} \mid x_{u,n}\,;\,\phi, \psi_u) && (2)\\ f_{X_{u,0}}(x_{u,0}\,;\,\theta) &= f_{X_{u,0}}(x_{u,0}\,;\,\phi, \psi_u). && (3) \end{aligned}$$

Then, ψ_u is a vector of *unit-specific* parameters for unit u, and ϕ is a *shared* parameter vector.

We suppose $\phi \in \mathbb{R}^A$ and $\psi_u \in \mathbb{R}^B$, so the dimension of the parameter vector θ is $D = A + BU$. In practice, the densities in Eqs. (1)–(3) serve two primary roles in PanelPOMP models: evaluation and simulation. Each function can depend on the unit to which it belongs, and therefore are specified in the unit-specific POMP models that together define the PanelPOMP. This feature is reflected in the software implementation, where the fundamental tasks of simulation and evaluation are defined in the unit `pomp` objects, as described in in Table 1.

Method	Operation	Function
`rprocess`	Simulate from Eq. (1)	$f_{X_{u,n}\|X_{u,n-1}}(x_{u,n} \mid x_{u,n-1}\,;\,\phi,\psi_u)$
`dprocess`	Evaluate Eq. (1)	$f_{X_{u,n}\|X_{u,n-1}}(x_{u,n} \mid x_{u,n-1}\,;\,\phi,\psi_u)$
`rmeasure`	Simulate from Eq. (2)	$f_{Y_{u,n}\|X_{u,n}}(y_{u,n} \mid x_{u,n}\,;\,\phi,\psi_u)$
`dmeasure`	Evaluate Eq. (2)	$f_{Y_{u,n}\|X_{u,n}}(y_{u,n} \mid x_{u,n}\,;\,\phi,\psi_u)$
`rinit`	Simulate from Eq. (3)	$f_{X_{u,0}}(x_{u,0}\,;\,\phi,\psi_u)$
`dinit`	Evalutate Eq. (3)	$f_{X_{u,0}}(x_{u,0}\,;\,\phi,\psi_u)$

Table 1: Methods of a unit model in a `panelPomp` object and their mathematical definitions.

In addition to the functions listed in in Table 1, additional basic mathematical functions can be specified for the unit objects of a `panelPomp` model as needed. For instance, functions `rprior` and `dprior`—which represent the operations of simulating from and evaluating a prior density function, respectively—can be specified as part of the unit objects if desired. The current version of the package (`1.7.0.0`), however, currently emphasizes the likelihood-based, plug-and-play methodologies described in Section 3, which do not require the specification of a prior distribution.

2.1 Model representation in panelPomp

The package uses the S4 functional object oriented programming approach in R (Wickham, 2019) in order to represent PanelPOMP models as `panelPomp` objects. Because a PanelPOMP model comprises multiple POMP models, a `panelPomp` object is essentially a structured collection of `pomp` objects from the **pomp** package (King et al., 2016). The `panelPomp` class contains three attributes: `unit_objects`, a list containing the models for each unit; `shared`, a named numeric vector of parameters that are shared across all units; and `specific`, a matrix of parameters that are unique to each unit.

Typically, creating `panelPomp` objects involves supplying a dataset for each unit and explicitly defining the mathematical functions in Table 1. The functions can be defined either using `R` functions or C-code snippets, the latter offering the advantage of reduced computing times. The construction of `pomp` objects using both `R` functions and C-code snippets is a topic discussed in detail in King et al. (2016) and various online tutorials for the **pomp** package. To avoid redundancy with this existing material, we focus instead on the novel features of the **panelPomp** package. Specifically, we illustrate how to construct a `panelPomp` object when a collection of `pomp` objects is already available.

For demonstration purposes, we consider a stochastic version of the discrete-time Gompertz population model (Winsor, 1932). This model is a popular choice for representing the exponential growth and decay observed in numerous ecological populations (Auger-Méthé et al., 2021; Smith et al., 2023; Lindén and Knape, 2009). It serves as a useful example due to its nonlinear, non-Gaussian nature, which can be transformed into a linear Gaussian model through log transformations. Consequently, the model is a popular choice for demonstrating the capabilities of plug-and-play algorithms on a nonlinear, non-Gaussian model where the likelihood can be exactly calculated using linear Gaussian techniques (Bretó et al., 2020). It further serves as an illustrative case study for **panelPomp**, as researchers may use similar models to describe population dynamics at distinct observation sites or experimental treatments. This approach allows for formal statistical testing to determine whether characteristics of the population dynamics are shared across populations or are unique to each unit.

For each unit u, the latent population density $X_{u,n}$ at time n is recursively modeled according to Eq. (4).

$$X_{u,n+1} = \kappa_u^{1-e^{-r_u}} X_{u,n}^{e^{-r_u}} \epsilon_{u,n}, \tag{4}$$

where $\epsilon_{n,u}$ are independent and identically-distributed log-normal random variables with $\log \epsilon_{u,n} \sim \mathrm{N}(0, \sigma_u^2)$. The observed population $Y_{u,n}$ at time n is assumed to follow a log-normal distribution, independently and identically distributed, conditioned on the value $X_{u,n}$,

$$\log Y_{u,n} | X_{u,n} \sim \mathrm{N}\big(\log X_{u,n}, \tau_u^2\big).$$

Below, we use the `pomp::gompertz` constructor function to build these unit-specific models. These constructors simultaneously build the unit models described above, and store a simulation from the model in the `data` attribute. The `times` argument in the constructor indicates the observation times for each unit; these times do not need to be the same when constructing `panelPomp` objects, as highlighted below.

```
gomp_u1 <- pomp::gompertz(
  K = 1, r = 0.1, sigma = 0.1, tau = 0.1, X_0 = 1, times = 1:20, seed = 111
)

gomp_u2 <- pomp::gompertz(
  K = 1.5, r = 0.1, sigma = 0.1, tau = 0.07, X_0 = 1, times = 1:21, seed = 222
)

gomp_u3 <- pomp::gompertz(
  K = 1.2, r = 0.1, sigma = 0.1, tau = 0.15, X_0 = 1, times = 3:25, seed = 333
)
```

In this example, we construct three unit objects, `gomp_u1`, `gomp_u2`, and `gomp_u3`, each of which is a `pomp` object. Each object contains a vector of parameters `K`, `r`, `sigma` `tau`, and `X_0` that correspond to the parameters $\kappa_u, r_u, \sigma_u, \tau_u$ and $X_{u,0}$, respectively. To build a `panelPomp` object, we pass a list of these models and specify which parameters are `shared` across units and which are `specific` to each unit.

```
gomp <- panelPomp(
  object = list(gomp_u1, gomp_u2, gomp_u3),
  shared = c("r" = 0.1, "sigma" = 0.1),
  specific = c("K", "tau", "X_0")
)
```

The parameter values of the `shared` vector need to be explicitly defined, replacing the separate values in the parameter vectors for each unit object. The unit-specific parameters are extracted from the constituent `pomp` objects. In the above construction, we are creating a `panelPomp` model that has shared parameters `r` and `sigma`. Mathematically, this is equivalent to assuming that, for all $u \in 1:U$, $r_u = r = 0.1$ and $\sigma_u = \sigma = 0.1$. The choice of which parameters are shared and which are unit-specific can be modified, as the helper function described in the following subsection enables changing the original parameter specifications.

Several pre-built models are also included in the package via constructor functions, including:

- `contacts()` creates a dynamic model for the variation in sexual contacts for the data of Vittinghoff et al. (1999). The model supposes each individual has a latent rate of making sexual contacts that evolves over time, allowing for heterogeneity between and within individuals, auto-correlation in individual rates over time, and a trend in rates over the time of the study (Romero-Severson et al., 2015).

- `panelMeasles()` creates a PanelPOMP model for measles incidence data in several UK cities, based on the model of He et al. (2010). This model is a stochastic compartmental model that describes Measles incidence data using a susceptible, exposed, infected, recovered (SEIR) model. The source code provides a useful demonstration of how users can build compartmental models for panel data from an infectious disease outbreak.
- `panelRandomWalk()` constructs a collection of independent latent Gaussian random walk models. After building the panel model, the constructor populates the `data` slots for the created `unitObjects` by simulating from the created model.
- `panelGompertz()` creates a collection of the stochastic Gompertz population models described in this section. The `data` slots for the `unitObjects` are populated by using simulations from the model.

The primary purpose of these pre-built constructor functions is to provide source code demonstrating how users may create a variety of different `panelPomp` objects, and to enable quick testing and comparison of newly developed PanelPOMP methodology using existing models and data. The parent **pomp** package also contains a number of pre-built models and datasets for similar purposes. In cases where there may be confusion between the constructor functions in these packages, the `panel` prefix is used to clarify the distinction.

2.2 Generic methods for `panelPomp` objects

The created object from the previous section, `gomp`, is a PanelPOMP model with 3 independent units, each with a unique number of observations specified by the length of the `times` argument in their constructor function. In this model κ_u, τ_u and $X_{u,0}$ are treated as unit-specific, and $r_u = r, \sigma_u = \sigma$ are shared for all unit objects. Internally, each `unit_object` treats both types of parameters the same, and thereby the construction of the unit objects does not require renaming unit-specific parameters to allow for distinction across units. For example, though κ_u may be unique for each unit u, the internal functions representing the process model (Eq. (1)) can use the variable name `kappa` to represent the parameter in all unit objects, rather than needing unique variable names for each unit. This important feature allows for changing which parameters are treated as shared and which are unit-specific without having to redefine each of the unit objects.

Model parameters can be extracted and modified using the `coef()` generic function. The default treatment of this function is the convention `<parameter name>[<unit name>]` for unit-specific parameters, and `<parameter name>` for shared parameters. This format is intended to closely reflect the standard mathematical notation for unit-specific parameters. For instance, to change the value of κ_2, we could refer to the corresponding parameter `K[unit2]` in the `gomp` object:

```
coef(gomp)['K[unit2]'] <- 0.9
coef(gomp)
```

```
#>          r      sigma   K[unit1] tau[unit1] X_0[unit1]   K[unit2] tau[unit2]
#>       0.10       0.10       1.00       0.10       1.00       0.90       0.07
#> X_0[unit2]   K[unit3] tau[unit3] X_0[unit3]
#>       1.00       1.20       0.15       1.00
```

It can be more convenient to view unit-specific parameters as a matrix and shared parameters as a vector. This can be done using the `format = 'list'` argument of the `coef()` function, or by alternatively using the `shared()` and `specific()` functions to extract only the shared or unit-specific parameters, respectively.

```
coef(gomp, format = 'list')
```

```
#> $shared
```

```
#>     r sigma
#>   0.1   0.1
#>
#> $specific
#>      unit
#> param unit1 unit2 unit3
#>   K     1.0  0.90  1.20
#>   tau   0.1  0.07  0.15
#>   X_0   1.0  1.00  1.00

shared(gomp)

#>     r sigma
#>   0.1   0.1
```

The `shared()<-` and `specific()<-` setter functions are also convenient for modifying which model parameters are considered shared and unit-specific.

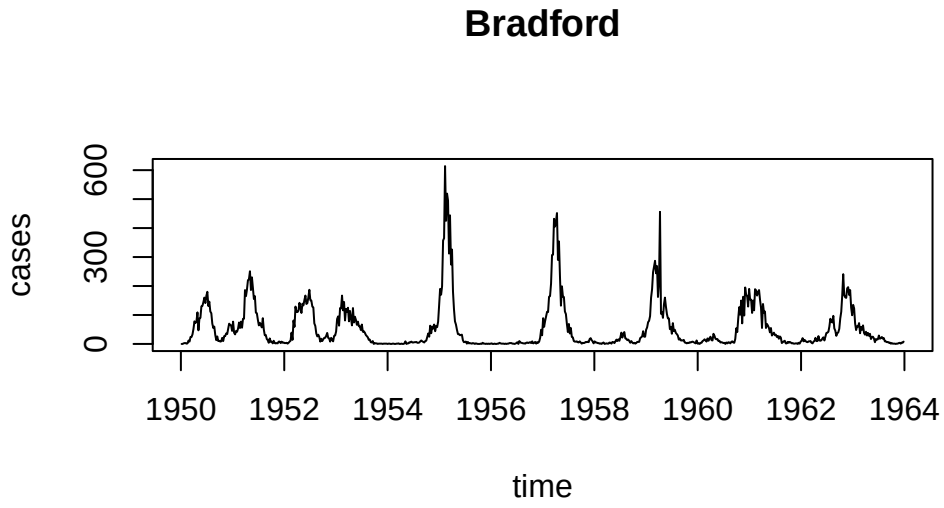
```
shared(gomp) <- c("tau" = 0.15)
shared(gomp)

#>   tau     r sigma
#>  0.15  0.10  0.10

specific(gomp)

#>      unit
#> param unit1 unit2 unit3
#>   K       1   0.9   1.2
#>   X_0     1   1.0   1.0
```

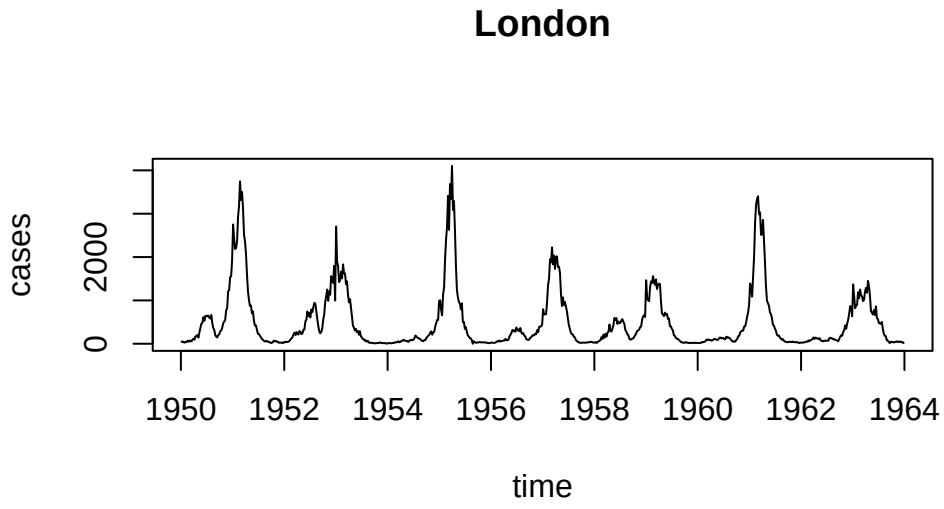
Before altering the definitions of existing parameters, these functions verify whether the parameters are already present in the model. They enable changing parameter values and adjusting whether a parameter is unit-specific or shared, but do not allow for the creation or deletion of model parameters.

A non-exhaustive list of useful methods that are frequently applied to `panelPomp` objects in the course of a data analysis includes:

- `simulate()`. For `panelPomp` objects that contain units with the functions `rinit`, `rprocess`, and `rmeasure`, the `simulate()` function can generate a number of simulations, specified by the `nsim` argument, which defaults as `nsim = 1`. The resulting object is a `panelPomp` object where the simulated results are saved as data in the unit objects.
- `plot()`. This function plots each unit of a `panelPomp` object sequentially. For derived classes, such as `pfilterd.ppomp` resulting from applying `pfilter` to a `panelPomp`, or `mif2d.ppomp` resulting from an application of `mif2`, diagnostic plots are produced for each unit. For example, calling `plot()` on the measles model constructed by the `panelMeasles()` function will plot the measles data available for the specified UK cities:

```
plot(panelMeasles(), units = c('Bradford', 'London'))
```





- `as()`. Many essential features of the basic model components of a `panelPomp` object have already undergone extensive development and testing in the context of the **pomp** package. To avoid unnecessary duplication, the `as()` function offers a way to convert a `panelPomp` object into a list of `pomp` objects. For example, `as(gomp, "list")` and `as(gomp, "pompList")` will convert the `gomp` `panelPomp` object into a `list` or `pompList`, respectively. This is particularly useful when it is appropriate to consider the individual components of the `panelPomp` object.

3 Inference methodology

All POMP methods can in principle be extended to PanelPOMPs; three different ways to represent a PanelPOMP as a POMP were identified by Romero-Severson et al. (2015). The ability to represent a PanelPOMP model as a POMP model, however, does not imply that methodology for POMP models will be feasible for PanelPOMP models. In particular, sequential Monte Carlo algorithms can have prohibitive scaling difficulties with the high dimensionality that arises in PanelPOMP models.

The current version of **panelPomp** emphasizes plug-and-play methods (Bretó et al., 2009; He et al., 2010), also known as likelihood-free (Marjoram et al., 2003; Sisson et al., 2007), that are applicable to dynamic models for which a simulator is available even when the transition densities are unavailable. In the terminology used in this article, this means that `dprocess` (Eq. (1)) is not needed in order to perform inference. This class of algorithms includes methods such as a particle filter (Arulampalam et al., 2002) and the panel iterated filter (PIF) (Bretó et al., 2020), which are methods that can be used to evaluate and maximize model likelihoods, respectively. In this section, we describe and demonstrate a plug-and-play likelihood-based inference workflow using the Stochastic Gompertz population model described previously. To highlight the capability of **panelPomp** to perform inference on high-dimensional models, we reconstruct a model with $U = 50$ measurement units and $N = N_u = 100$ observations per unit using the `panelGompertz()` constructor function.

```
gomp <- panelGompertz(N = 100, U = 50)
```

3.1 Log-likelihood evaluation via particle filters

The particle filter, also known as sequential Monte Carlo, is a standard tool for log-likelihood evaluation on nonlinear non-Gaussian POMP models. The log-likelihood function is a central component of Bayesian and frequentist inference. Due to the dynamic independence between units that is a defining feature of a PanelPOMP model, particle filtering can be carried out separately on each unit. The `pfilter` method for `panelPomp` objects is therefore a direct extension of the `pfilter` method for `pomp` objects from the `pomp` package. All mathematical functions needed to carry out the particle filter for a PanelPOMP model are represented by the functions `rinit`, `rprocess`, `rmeasure` and `dmeasure` (Eqs. (1)–(3)).

Methods applied to `panelPomp` objects—like the `pfilter` function—typically create new objects of the same class or a child class of the original object. For instance, we can perform a particle filter on the `gomp` object that contains 50 units, and 100 observations per unit in the following way:

```
gomp_pfd <- pfilter(gomp, Np = 1000)
```

In this case, we used `Np` = 1000 particles to obtain a single stochastic estimate of the log-likelihood of the `gomp` model. The resulting object `gomp_pfd` is of class `pfilterd.ppomp`, which is a child class of `panelPomp` and contains the estimated log-likelihood of each unit object, as well as the log-likelihood of the entire panel; these can be accessed using the `unitLogLik()` and `logLik()` functions, respectively.

In practice it is advisable to repeat this Monte Carlo approximation in order to reduce and quantify the error associated with the estimate. Because sequential Monte Carlo algorithms can be computationally expensive, we obtain replicated estimates by taking advantage of multicore computation using the **foreach** (Microsoft and Weston, 2022b) and **doParallel** (Microsoft and Weston, 2022a) packages:

```
pf_results <- foreach(i = 1:10) %dopar% {
    pfilter(gomp, Np = 1000)
}
```

This took 2.68 seconds to run the 10 replicates in parallel, resulting in a list of objects of class `pfilterd.ppomp`. We can use the `logLik` function to extract the Monte Carlo estimate of the log-likelihood $\lambda^{[i]}$ for each replicate i, and `unitLogLik` to extract the vector of component Monte Carlo log-likelihood estimates $\lambda_u^{[i]}$ for each unit $u = 1, \ldots, U$, where $\lambda^{[i]} = \sum_{u=1}^{U} \lambda_u^{[i]}$. For a POMP model, replicated log-likelihood evaluations via the particle filter are usually averaged on the natural scale, rather than the log scale, to take advantage of the unbiasedness of the particle filter likelihood estimate. Thus, we have

$$\hat{\lambda}_1 = \log\Big(\frac{1}{I}\sum_{i=1}^{I}\exp\big\{\sum_{u=1}^{U}\lambda_u^{[i]}\big\}\Big),$$

which can be implemented as

```
lambda_1 <- logmeanexp(
  sapply(pf_results,logLik), se = TRUE
)
```

giving $\hat{\lambda}_1 = 2065.4$ with a jack-knife standard error of 0.5. Taking advantage of the independence of the units in the panel structure, Bretó et al. (2020) showed it is preferable to average the replicates of marginal likelihood for each unit before taking a product over units. This corresponds to

$$\hat{\lambda}_2 = \log\Big(\prod_{u=1}^{U}\frac{1}{I}\sum_{i=1}^{I}\exp\big\{\hat{\lambda}_u^{[i]}\big\}\Big),$$

which can be obtained using

```
lambda_2 <- panel_logmeanexp(
  sapply(pf_results,unitLogLik), MARGIN = 1, se = TRUE
)
```

giving $\hat{\lambda}_2 = 2068.5$ with a jack-knife standard error of 1.1. For this model, a Kalman filter log-likelihood evaluation gives an exact answer, $\lambda = 2068.2$.

3.2 Maximum likelihood estimation

Although particle filters can effectively approximate the log-likelihood of non-linear models, it is well known that obtaining a maximum likelihood estimate for fixed parameters using these filters is difficult in practice. To maximize the likelihood, we employ iterated filtering algorithms, which perform repeated particle filtering on an extended version of the model that incorporates time-varying parameter perturbations. With each iteration, the magnitude of these perturbations is reduced, allowing the algorithm to approach a local maximum of the likelihood function. An example of this type of algorithm for general POMP models includes the IF2 iterated filtering algorithm (Ionides et al., 2015). IF2 has been successfully employed for likelihood-based inference in various POMP models, particularly in epidemiology and ecology, as reviewed by Bretó (2018). However, both particle filters and IF2 face scalability issues as model dimensions increase, and IF2 cannot be applied to each unit separately when the PanelPOMP model includes shared parameter values.

A panel iterated filtering (PIF) algorithm was developed by Bretó et al. (2020), extending IF2 to panel data. An implementation of PIF in **panelPomp** is provided by the `mif2` method for class `panelPomp`, following the pseudocode in Algorithm 1. The pseudocode sometimes omits explicit specification of ranges over which variables are to be computed when this is apparent from the context: it is understood that j takes values in $1:J$, a in $1:A$ and b in $1:B$. The $N[0,1]$ notation corresponds to the construction of independent standard normal random variables, leading to Gaussian perturbations of parameters on a transformed scale. The theory allows considerable flexibility in how the parameters are perturbed, but Gaussian perturbations on an appropriate scale are typically adequate.

At a conceptual level, the PIF algorithm has an evolutionary analogy: successive iterations mutate parameters and select among the fittest outcomes measured by Monte Carlo likelihood evaluation. Most often, the perturbation parameters $\sigma^{\Phi}_{a,n}$ and $\sigma^{\Psi}_{b,u,n}$ in Algorithm 1 will not depend on n. For parameters that have uncertainty on a unit scale, the value 0.02 demonstrated here has been commonly used. The help documentation on the `rw_sd` argument gives instruction on using additional structure should it become necessary.

The unmarginalized PIF was proposed by Bretó et al. (2020), who provided theoretical convergence results for the algorithm. When MARGINALIZE = TRUE, the PIF algorithm

Algorithm 1: `mif2(pp, Nmif =` M`, Np =` J`, start =` $(\phi_a^0, \psi_{b,u}^0)$`, rw_sd =` $(\sigma_{a,n}^{\Phi}, \sigma_{b,u,n}^{\Psi})$`, cooling.factor.50 =` ρ^{50}`)`, where pp is a `panelPomp` object containing data and defined `rprocess`, `dmeasure`, `rinit` and `partrans` components.

input: Data, $y_{u,n}^*$, u in $1:U$, n in $1:N$
Simulator of initial density, $f_{X_{u,0}}(x_{u,0}\,;\phi,\psi_u)$
Simulator of transition density, $f_{X_{u,n}|X_{u,n-1}}(x_{u,n} \mid x_{u,n-1}\,;\phi,\psi_u)$
Evaluator of measurement density, $f_{Y_{u,n}|X_{u,n}}(y_{u,n} \mid x_{u,n}\,;\phi,\psi_u)$
Number of particles, J, and number of iterations, M
Starting shared parameter swarm, $\Phi_{a,j}^0 = \phi_a^0$, a in $1:A$, j in $1:J$
Starting unit-specific parameter swarm, $\Psi_{b,u,j}^0 = \psi_{b,u}^0$, b in $1:B$, j in $1:J$
Random walk intensities, $\sigma_{a,n}^{\Phi}$ and $\sigma_{b,u,n}^{\Psi}$
Parameter transformations, h_a^{Φ} and h_b^{Ψ}, with inverses $\left(h_a^{\Phi}\right)^{-1}$ and $\left(h_b^{\Psi}\right)^{-1}$
Logical variable determining marginalization, MARGINALIZE

output: Final parameter swarm, $\Phi_{a,j}^M$ and $\Psi_{b,u,j}^M$

For m in $1:M$
  $\Phi_{a,0,j}^m = \Phi_{a,j}^{m-1}$
  For u in $1:U$
    $\Phi_{a,u,0,j}^{F,m} = \left(h_a^{\Phi}\right)^{-1}\left(h_a^{\Phi}\left(\Phi_{a,u-1,j}^m\right) + \rho^m \sigma_{a,0}^{\Phi} Z_{a,u,0,j}^{\Phi,m}\right)$ for $Z_{a,u,0,j}^{\Phi,m} \sim N[0,1]$
    $\Psi_{b,u,0,j}^{F,m} = \left(h_b^{\Psi}\right)^{-1}\left(h_b^{\Psi}\left(\Psi_{b,u,j}^{m-1}\right) + \rho^m \sigma_{b,u,0}^{\Psi} Z_{b,u,0,j}^{\Psi,m}\right)$ for $Z_{b,u,0,j}^{\Psi,m} \sim N[0,1]$
    $X_{u,0,j}^{F,m} \sim f_{X_{u,0}}\left(x_{u,0} \mid \Phi_{a,u,0,j}^{F,m}, \Psi_{b,u,0,j}^{F,m}\right)$
    For n in $1:N_u$
      $\Phi_{a,u,n,j}^{P,m} = \left(h_a^{\Phi}\right)^{-1}\left(h_a^{\Phi}\left(\Phi_{a,u,n-1,j}^{F,m}\right) + \rho^m \sigma_{a,n}^{\Phi} Z_{a,u,n,j}^{\Phi,m}\right)$ for $Z_{a,u,n,j}^{\Phi,m} \sim N[0,1]$
      $\Psi_{b,u,n,j}^{P,m} = \left(h_b^{\Psi}\right)^{-1}\left(h_b^{\Psi}\left(\Psi_{b,u,n-1,j}^{F,m}\right) + \rho^m \sigma_{b,u,n}^{\Psi} Z_{b,u,n,j}^{\Psi,m}\right)$ for $Z_{b,u,n,j}^{\Psi,m} \sim N[0,1]$
      $X_{u,n,j}^{P,m} \sim f_{X_{u,n}|X_{u,n-1}}\left(x_{u,n} \mid X_{u,n-1,j}^{F,m}\,;\, \Phi_{a,u,n,j}^{P,m}, \Psi_{b,u,n,j}^{P,m}\right)$
      $w_{u,n,j}^m = f_{Y_{u,n}|X_{u,n}}\left(y_{u,n}^* \mid X_{u,n,j}^{P,m}\,;\, \Phi_{a,u,n,j}^{P,m}, \Psi_{b,u,n,j}^{P,m}\right)$
      Draw $k_{1:J}$ with $\mathbb{P}(k_j = i) = w_{u,n,i}^m \Big/ \sum_{q=1}^J w_{u,n,q}^m$
      $\Phi_{a,u,n,j}^{F,m} = \Phi_{a,u,n,k_j}^{P,m}$, $\Psi_{b,u,n,j}^{F,m} = \Psi_{b,u,n,k_j}^{P,m}$ and $X_{u,n,j}^{F,m} = X_{u,n,k_j}^{P,m}$
      If MARGINALIZE then
        $\Psi_{b,\nu,n,j}^{F,m} = \Psi_{b,\nu,n,j}^{P,m}$ for all $\nu \neq u$
      Else
        $\Psi_{b,\nu,n,j}^{F,m} = \Psi_{b,\nu,n,k_j}^{P,m}$ for all $\nu \neq u$
    End For
    $\Phi_{a,u,j}^m = \Phi_{a,u,N_u,j}^{F,m}$ and $\Psi_{b,u,j}^m = \Psi_{b,u,N_u,j}^{F,m}$
  End For
  $\Phi_{a,j}^m = \Phi_{a,U,j}^m$
End For

is modified such that the particles representing the unit-specific parameter $\psi_{b,u}$ remain unchanged when filtering through a unit $\nu \neq u$. Recent results suggest the marginalized PIF (MPIF) algorithm has superior empirical performance in many situations, but does not yet have theoretical support. For the remainder of this article, the presented results use the unmarginalized version of the algorithm.

Parameter transformations (h_a^{Φ} and h_b^{Ψ}) are used to ensure that parameter estimates remain within accepted bounds. For example, parameters that are required to be positive can be estimated on a log-transformed scale to maintain their positivity when converted back to the natural scale. This process is facilitated by the `parameter_trans` function,

which manages the transformation automatically. This relieves users from the need to handle parameters on the transformed scale directly; they only need to specify the desired transformation. In the case of the gomp model, all model parameters must be non-negative. This requirement can be met by creating unit objects and assigning the partrans argument as follows:

```
parameter_trans(log = c("K", "r", "sigma", "tau", "X.0"))
```

This setup ensures that the parameters κ_u, r_u, σ_u, τ_u, and $X_{u,0}$ are all estimated on a log-transformed scale, thereby maintaining their non-negativity on the natural scale. Other common parameter transformations that can be implemented include the logit transformation, which ensures parameters are in the interval $(0, 1)$, and the barycentric transformation, which is used when a collection of parameters must lie in the interval $(0, 1)$, with the additional constraint that they sum to one. Finally, custom transformations can be defined using the toEst and fromEst arguments to the parameter_trans function.

We demonstrate maximum likelihood estimation using parameter transformations for the gomp object described previously. To do so, we need to specify starting parameter values from where to start the search. This can be done by explicitly providing the starting parameters using either the start or the shared.start and specific.start arguments of the mif2() function. Alternatively, the existing parameter values in the panelPomp object will be implicitly used as a starting point, as done in the following example. For simplicity, we fix $\kappa_u = 1$ and the initial condition $X_{u,0} = 1$, maximizing over two shared parameters, r and σ, and one unit-specific parameter τ_u, starting from their default values in the gomp object:

```
gomp_mif2d <- mif2(
  gomp,  # panelPomp model, which contains parameter transformations and values.
  Nmif = 25,  # Number of Iterations (M)
  Np = 250,  # Number of Particles (J)
  cooling.fraction.50 = 0.5,  # Cooling intensity, after 50 iterations
  cooling.type = "geometric",  # Cooling Style
  rw.sd = rw_sd(r = 0.02, sigma = 0.02, tau = 0.02)  # Random Walk SD
)
```

The output gomp_mif2d is a mif2d.ppomp object, which is a child class of panelPomp. The algorithmic parameters are very similar to those of the mif2 method for class pomp. The perturbations, determined by the rw_sd argument, may be a list giving separate instructions for each unit. When only one specification for a unit-specific parameter is given (as we do for τ_u here) the same perturbation is used for all units. As such, functions for coefficient extraction can be used to get the final parameter estimates, for instance:

```
shared(gomp_mif2d)

#>          r      sigma
#> 0.06833055 0.09123509
```

For Monte Carlo maximization, replication from diverse starting points is recommended. Further, larger values of Nmif and Np are typically needed in order to reliably maximize model likelihoods in practice. Smaller initial searches for parameter estimates are useful in that they can be used to estimate the computational cost of a larger search or to help determine values of hyperparameters. The computational complexity of the PIF algorithm is $O(JMNU)$, and so the cost of a larger search may be estimated by noting that the complexity is linear in each of the arguments J and M.

We demonstrate such a maximization search on gomp. The small, single PIF maximization took 40.8 seconds to compute. By increasing the number of iterations (Nmif) and particles (Np) each by a factor of 6, we expect the computation time for a single parameter initialization to

take roughly 24.5 minutes, noting that there is some overhead in the maximization function that is not increased on larger jobs. Using 36 cores, we then expect the maximization routine to take 24.5 minutes for 36 distinct starting values.

To define diverse starting points for the Monte Carlo replicates, we make uniform draws from a specified box. The parameter bounds on this box are not intended to be bounds on the final parameter estimates, as there is a possibility that the data lead the parameter search elsewhere. However, if replicated searches started from this box reliably reach a consensus, we claim we have carefully investigated this part of parameter space. A larger box leads to greater confidence that the relevant part of the parameter space has been searched, at the expense of requiring additional work. The `runif_panel_design()` function facilitates the construction and drawing random points from within the box.

```
starts <- runif_panel_design(
  lower = c('r' = 0.05, 'sigma' = 0.05, 'tau' = 0.05, 'K' = 1, 'X.0' = 1),
  upper = c('r' =  0.2, 'sigma' =  0.2, 'tau' =  0.2, 'K' = 1, 'X.0' = 1),
  specific_names = c('K', 'tau', 'X.0'),
  unit_names = names(gomp),
  nseq = 36
)
```

We then carry out a search from each starting point:

```
mif_results <- foreach(start=iter(starts,"row")) %dopar% {
  mif2(
    gomp, start = unlist(start),
    Nmif = 150,
    Np = 1500,
    cooling.fraction.50 = 0.5,
    cooling.type = "geometric",
    transform = TRUE,
    rw.sd = rw_sd(r = 0.02, sigma = 0.02, tau = 0.02)
  )
}
```

This took 15.6 minutes using 36 cores, producing a list of objects of class `mifd.ppomp`. We can check on convergence of the searches, and possibly diagnose improvements in the choices of algorithmic parameters, by consulting trace plots of the searches available via the `traces` method for class `mifd.ppomp`. This follows recommendations by Ionides et al. (2006) and King et al. (2016).

Block optimization

A characteristic of PanelPOMP models is the large number of parameters arising when unit-specific parameters are specified for a large number of units. For a fixed value of the shared parameters, the likelihood of the unit-specific parameters factorizes over the units. The factorized likelihood can be maximized separately over each unit, replacing a challenging high-dimensional problem with many relatively routine low-dimensional problems. This suggests a block maximization strategy where unit-specific parameters for each unit are maximized as a block. Bretó et al. (2020) used a simple block strategy where a global search over all parameters is followed by a block maximization over units for unit-specific parameters. The theoretical guarantees available for the PIF algorithm (Bretó et al., 2020) imply that this block-refinement strategy is not necessary for convergence, but empirically it can help reduce the computational effort needed to fully maximize the likelihood.

We demonstrate this here, refining each of the maximization replicates above. The following function carries out a maximization search of unit-specific parameters for a single

unit. The call to `mif2` takes advantage of argument recycling: all algorithmic parameters are re-used from the construction of `mifd_gomp` except for the re-specified random walk standard deviations which ensures that only the unit-specific parameters are perturbed.

```
mif_unit <- function(unit, mifd_gomp, reps = 6) {
  unit_gomp <- unit_objects(mifd_gomp)[[unit]]

  mifs <- replicate(
    n = reps, mif2(unit_gomp, rw.sd = rw_sd(tau = 0.02))
  )

  best <- which.max(sapply(mifs, logLik))
  coef(mifs[[best]])["tau"]
}
```

Now we apply this block maximization to find updated unit-specific parameters for each replicate, and we insert these back into the `panelPomp`.

```
mif_block <- foreach(mf=mif_results) %dopar% {
  mf@specific["tau",] <- sapply(1:length(mf), mif_unit, mifd_gomp = mf)
  mf
}
```

This took 40.4 seconds to compute all 36 block refinements in parallel.

We expect Monte Carlo estimates of the maximized log-likelihood functions to fall below the actual (usually unknown) value. This is in part because imperfect maximization can only reduce the maximized likelihood, and in part a consequence of Jensen's inequality applied to the likelihood evaluation: the unbiased SMC likelihood evaluation has a negative bias on estimation of the log-likelihood.

3.3 Parameter uncertainty

A key component of a likelihood-based inference framework is the estimation of parameter uncertainty, often achieved by the estimation of confidence intervals. This task is particularly challenging for general non-linear, latent variable models. Some potential methods include profile likelihoods, observed Fisher information, and the bootstrap method. Here, we demonstrate the profile likelihood approach, which has proven useful for mechanistic models (Simpson and Maclaren, 2023).

The profile likelihood function is constructed by fixing a focal parameter at various values and then maximizing the likelihood over all other parameters for each value of the focal parameter. Constructing a profile likelihood function offers several practical advantages:

- Evaluations at neighboring values of the focal parameter provide additional Monte Carlo replication. Typically, the true profile log-likelihood is smooth, and asymptotically close to quadratic under regularity conditions, so deviations from a smooth fitted line can be interpreted as Monte Carlo error.
- Large-scale features of the profile likelihood reveal a region of the parameter space outside which the model provides a poor explanation of the data.
- Co-plots, which show how the values of other maximized parameters vary along the profile, may provide insights into parameter trade-offs implied by the data.
- The smoothed Monte Carlo profile log-likelihood can be used to construct an approximate 95% confidence interval. The resulting confidence interval can be properly adjusted to accommodate both statistical and Monte Carlo uncertainty (Ionides et al., 2017).

Once we have code for maximizing the likelihood, only minor adaptation is needed to carry out the maximizations for a profile. The `runif_panel_design` generating the starting

values is replaced by a call to `profile_design`, which assigns the focal parameter to a grid of values and randomizes the remaining parameters. The random walk standard deviation for the focal parameter is unassigned, which leads it to be set to zero and therefore the parameter remains fixed during the maximization process. The following code combines the joint and block maximizations developed above.

```
# Names of the estimated parameters
estimated <- c(
  "r", "sigma", paste0("tau[unit", 1:length(gomp), "]")
)

# Names of the fixed parameters (not estimated)
fixed <- names(coef(gomp))[!names(coef(gomp)) %in% estimated]

profile_starts <- profile_design(
  r = seq(0.05, 0.2, length = 20),
  lower = c(coef(gomp)[estimated] / 2, coef(gomp)[fixed])[-1],
  upper = c(coef(gomp)[estimated] * 2, coef(gomp)[fixed])[-1],
  nprof = 5, type = "runif"
)

profile_results <- foreach(start = iter(profile_starts, "row")) %dopar% {
  mf <- mif2(
    mif_results[[1]],
    start = unlist(start),
    rw.sd = rw_sd(sigma = 0.02, tau = 0.02)
  )
  mf@specific["tau", ] <- sapply(1:length(mf), mif_unit, mifd_gomp = mf)
  mf
}
```

In the above code, the `profile_design` function creates $20 \times 5 = 100$ unique starting points to perform the profile search. Parallelized over 36 cores, these 100 searches took 47.6 total minutes. However, we are not quite done gathering the results for the profile. The perturbed filtering carried out by `mif2` leads to an approximate likelihood evaluation, but additional accuracy is obtained by re-evaluating the likelihood without perturbations. Also, replication is recommended to reduce and quantify Monte Carlo error. We do this, and tabulate the results.

```
profile_table <- foreach(mf=profile_results,.combine=rbind) %dopar% {
  LL <- replicate(10, logLik(pfilter(mf, Np = 2500)))
  LL <- logmeanexp(LL, se = TRUE)
  data.frame(t(coef(mf)), loglik = LL[1], loglik.se = LL[2])
}
```

The likelihood evaluations took 2.5 minutes. It is appropriate to spend comparable time evaluating the likelihood to the time spent maximizing it: a high quality maximization without high quality likelihood evaluation is hard to interpret, whereas good evaluations of the likelihood in a vicinity of the maximum can inform about the shape of the likelihood surface in this region, and this may be as relevant as knowing the exact maximum.

The Monte Carlo adjusted profile (MCAP) approach of Ionides et al. (2017) is implemented by the `mcap()` function in **pomp**. This function constructs a smoothed profile likelihood, by application of the `loess` smoother. It computes a local quadratic approximation that is used to derive an extension to the classical profile likelihood confidence interval that makes allowance for Monte Carlo error in the calculation of the profile points. Theoretically, an MCAP procedure can obtain statistically efficient confidence intervals even when

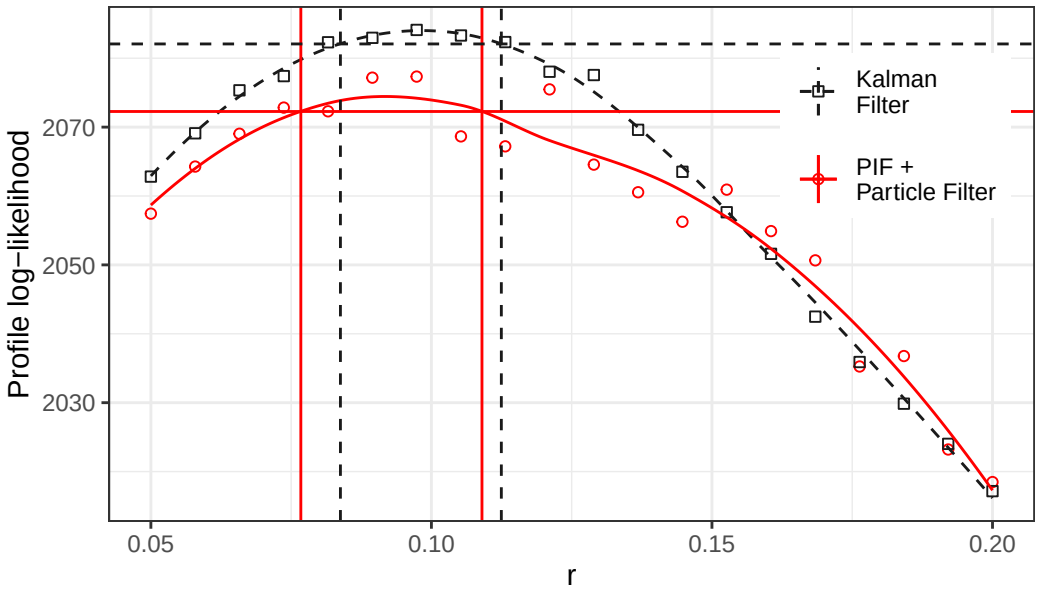


Figure 1: The Monte Carlo adjusted profile confidence interval (solid red lines, evaluation points shown as circles). Construction using deterministic optimization of the likelihood calculated by the Kalman filter (dashed lines, evaluation points show as squares).

the Monte Carlo error in the profile likelihood is asymptotically growing and unbounded (Ning et al., 2021). Log-likelihood evaluation has negative bias, as a consequence of Jensen's inequality for an unbiased likelihood estimate. This bias produces a vertical shift in the estimated profile, which fortunately does not have consequence for the confidence interval if the bias is slowly varying.

The profile points evaluated above, and stored in `profile_table`, can be used to compute a 95% MCAP confidence interval as follows:

```
profile_table <- profile_table |>
  as.data.frame() |>
  dplyr::group_by(r) |>
  dplyr::slice_max(n = 1, order_by = loglik)

gomp_mcap <- pomp::mcap(
  logLik = profile_table$loglik,
  parameter = profile_table$r,
  level = 0.95
)
```

The construction of the confidence interval is best shown by a plot of the smoothed profile likelihood, shown in Fig. 1 (Wickham, 2016). In this toy example, the exact likelihood can be calculated using the Kalman filter, and this is carried out by the `panelGompertzLikelihood` function. The likelihood can then be maximized using a general-purpose optimization procedure such as `optim()` in R. With large numbers of parameters, and no guarantee of convexity, this numerical optimization is not entirely routine. One might consider a block optimization strategy, but here we carry out a simple global search, which took 1.7 minutes to compute the profile likelihood, once parallelized. The deterministic search is also not entirely smooth, and so we apply MCAP as for the Monte Carlo search. Both deterministic and Monte Carlo optimizations can benefit from a block optimization strategy which alternates between shared and unit-specific parameters (Bretó et al., 2020). Such algorithms can be built using the **panelPomp** functions we have demonstrated, and they will be incorporated into the package once they have been more extensively researched.

4 Conclusion

The analysis using the `gomp` model illustrates one approach to plug-and-play inference for PanelPOMP models, but the scope of **panelPomp** is far from limited to this approach. **panelPomp** is a general and extensible framework which encourages the development of additional functionality. The `panelPomp` class, along with its associated workhorse functions, provide an adaptable interface that can accommodate future methodologies. In this sense, **panelPomp** provides an environment for sharing and developing PanelPOMP models and methods, both through future contributions to the **panelPomp** package and through open-source applications that leverage the package. This framework will facilitate the comparison of new methodologies with existing ones, promoting continuous improvement and innovation. Other software packages, such as **pomp**, **spatPomp**, **nimble**, among others summarized in Section 4.2 of Newman et al. (2023), can also be used to perform inference on non-linear mechanistic models. However, none of these packages specifically address the unique challenges posed by high-dimensional PanelPOMP models.

A special class of POMP models arises when the latent process takes values in a discrete and finite space. A model of this type is often referred to as a hidden Markov model (HMM) (Eddy, 2004; Doucet et al., 2001; Glennie et al., 2023; Newman et al., 2023), though this terminology has also been used as a synonym for POMP (King et al., 2016). Under this additional constraint, efficient dynamic-programming algorithms can be used to perform inference, as summarized by McClintock et al. (2020). In principle, **panelPomp** can be leveraged to implement these existing approaches for longitudinal data, though the current version of the package emphasizes methodologies for models that have latent processes with states taking values in spaces that cannot be programmatically searched.

In our example, likelihood evaluation and maximization was used to construct confidence intervals. These calculations also provide a foundation for other techniques of likelihood-based inference, such as likelihood ratio hypothesis tests and model selection via Akaike's information criterion (AIC). The examples discussed provide case studies in the use of these methods for scientific work.

Data analysis using large data sets or complex models may require considerable computing time. Simulation-based methodology is necessarily computationally intensive, and access to a cluster computing environment extends the size of problems that can be tackled. Our example workflow has a simple parallel structure that can readily take advantage of additional resources. Embarrassingly parallel computations, such as computing the profile likelihood function at a grid of points, or replicated evaluations of the likelihood function, can be parallelized using the **foreach** package.

Panel data are widely available: for many experimental and observational systems it is more practical to collect short time series on many units than to obtain one long time series. For time series data, fitting mechanistic models specified as partially observed Markov processes has found numerous applications for formulating and answering scientific hypotheses (Bretó et al., 2009; King et al., 2016). However, there are remarkably few examples in the literature fitting mechanistic nonlinear non-Gaussian partially observed stochastic dynamic models to panel data. The **panelPomp** package offers opportunities to remedy this situation.

5 Availability, documentation and code quality control

panelPomp is available on CRAN and can be installed by executing `install.packages('panelPomp')`. The source code and developmental version of the package are available on GitHub: `https://github.com/panelPomp-org/panelPomp`. Package documentation is created using `roxygen2` and is shipped with the installation of the package, but can also be found at the package website: `https://panelpomp-org.github.io`. Two tutorials are provided on the website; an elementary "Getting Started" guide, and an in-depth introduction which contains examples that are similar to those in this article

(Breto et al., 2024). Continuous integration based on GitHub actions is used to build and test the package. As of writing, unit tests have a 100% line coverage (measured by the **covr** package).

All major computations were performed on a high-performance computing (HPC) node equipped with 2x 3.0 GHz Intel Xeon Gold 6154 processors, totaling 36 cores. The system ran on Red Hat Enterprise Linux 8.8 (Ootpa) on an x86_64-pc-linux-gnu platform, with 180 GB RAM. We used R version 4.4.0 (2024-04-24) for our analyses. This paper introduces **panelPomp** version `1.7.0.0`, and requires **pomp** version 6.2.1.0 or higher.

6 Acknowledgements and funding

This work was supported by National Science Foundation grants DMS-1761603 and DMS-1646108; National Institutes of Health grants 1-U54-GM111274, 1-U01-GM110712, and 1-R01-AI143852; and by MCIN/AEI/10.13039/501100011033 grants PID2020-116242RB-I00 and PID2023-152348NB-I00.

This research was additionally supported in part through computational resources and services provided by Advanced Research Computing at the University of Michigan, Ann Arbor.

References

M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear, non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188, 2002. doi: 10.1109/78.978374. [p187]

K. Asfaw, J. Park, A. A. King, and E. L. Ionides. spatpomp: An R package for spatiotemporal partially observed Markov process models. *Journal of Open Source Software*, 9(104):7008, 2024. doi: 10.21105/joss.07008. URL https://doi.org/10.21105/joss.07008. [p181]

M. Auger-Méthé, K. Newman, D. Cole, F. Empacher, R. Gryba, A. A. King, V. Leos-Barajas, J. Mills Flemming, A. Nielsen, G. Petris, et al. A guide to state–space modeling of ecological time series. *Ecological Monographs*, 91(4):e01470, 2021. doi: https://doi.org/10.1002/ecm.1470. [p182]

C. Bretó. Modeling and inference for infectious disease dynamics: A likelihood-based approach. *Statistical Science*, 33(1):57–69, 2018. doi: 10.1214/17-STS636. [p188]

C. Bretó, D. He, E. L. Ionides, and A. A. King. Time series analysis via mechanistic models. *Annals of Applied Statistics*, 3:319–348, 2009. doi: 10.1214/08-AOAS201. [p180, 187, 195]

C. Bretó, E. L. Ionides, and A. A. King. Panel data analysis via mechanistic models. *Journal of the American Statistical Association*, 115(531):1178–1188, 2020. doi: 10.1080/01621459.2019.1604367. [p180, 182, 187, 188, 191, 194]

C. Breto, J. Wheeler, A. A. King, and E. L. Ionides. A tutorial on panel data analysis using partially observed Markov processes via the R package panelpomp. *arXiv*, 2409.03876, 2024. doi: https://doi.org/10.48550/arXiv.2409.03876. [p196]

Y. Croissant and G. Millo. Panel data econometrics in R: The plm package. *Journal of Statistical Software*, 27(2):1–43, 2008. doi: 10.18637/jss.v027.i02. [p180]

J. E. Domeyer, J. D. Lee, H. Toyoda, B. Mehler, and B. Reimer. Driver-pedestrian perceptual models demonstrate coupling: Implications for vehicle automation. *IEEE Transactions on Human-Machine Systems*, 52(4):557–566, 2022. doi: 10.1109/THMS.2022.3158201. [p180]

A. Doucet, N. De Freitas, N. J. Gordon, et al. *Sequential Monte Carlo methods in practice*, volume 1. Springer, 2001. doi: 10.1007/978-1-4757-3437-9. [p195]

S. R. Eddy. What is a hidden Markov model? *Nature Biotechnology*, 22(10):1315–1316, 2004. doi: 10.1038/nbt1004-1315. [p195]

R. Glennie, T. Adam, V. Leos-Barajas, T. Michelot, T. Photopoulou, and B. T. McClintock. Hidden Markov models: Pitfalls and opportunities in ecology. *Methods in Ecology and Evolution*, 14(1):43–56, 2023. doi: 10.1111/2041-210X.13801. [p195]

U. Halekoh, S. Højsgaard, and J. Yan. The R package geepack for generalized estimating equations. *Journal of Statistical Software*, 15(2):1–11, 2006. doi: 10.18637/jss.v015.i02. [p180]

D. He, E. L. Ionides, and A. A. King. Plug-and-play inference for disease dynamics: Measles in large and small towns as a case study. *Journal of the Royal Society Interface*, 7:271–283, 2010. doi: 10.1098/rsif.2009.0151. [p184, 187]

E. L. Ionides, C. Bretó, and A. A. King. Inference for nonlinear dynamical systems. *Proceedings of the National Academy of Sciences of the USA*, 103:18438–18443, 2006. doi: 10.1073/pnas.0603181103. [p191]

E. L. Ionides, D. Nguyen, Y. Atchadé, S. Stoev, and A. A. King. Inference for dynamic and latent variable models via iterated, perturbed Bayes maps. *Proceedings of the National Academy of Sciences of the USA*, 112(3):719–724, 2015. doi: 10.1073/pnas.1410597112. [p188]

E. L. Ionides, C. Bretó, J. Park, R. A. Smith, and A. A. King. Monte carlo profile confidence intervals for dynamic systems. *Journal of the Royal Society Interface*, 14(132):1–10, 2017. doi: 10.1098/rsif.2017.0126. [p192, 193]

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. doi: 10.18637/jss.v069.i12. URL https://www.jstatsoft.org/index.php/jss/article/view/v069i12. [p181, 182, 191, 195]

A. Lindén and J. Knape. Estimating environmental effects on population dynamics: Consequences of observation error. *Oikos*, 118(5):675–680, 2009. doi: 10.1111/j.1600-0706.2008.17250.x. [p182]

J. A. Marino, S. D. Peacor, D. B. Bunnell, H. A. Vanderploeg, S. A. Pothoven, A. K. Elgin, J. R. Bence, J. Jiao, and E. L. Ionides. Evaluating consumptive and nonconsumptive predator effects on prey density using field time series data. *Ecology*, 100:e02583, 2019. doi: 10.1002/ecy.2583. [p180]

P. Marjoram, J. Molitor, V. Plagnol, and S. Tavare. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, 2003. doi: 10.1073/pnas.0306899100. [p187]

B. T. McClintock, R. Langrock, O. Gimenez, E. Cam, D. L. Borchers, R. Glennie, and T. A. Patterson. Uncovering ecological state dynamics with hidden Markov models. *Ecology Letters*, 23(12):1878–1903, 2020. doi: https://doi.org/10.1111/ele.13610. [p195]

Microsoft and S. Weston. *doParallel: Foreach parallel adaptor for the 'parallel' package*, 2022a. URL https://CRAN.R-project.org/package=doParallel. R package version 1.0.17. [p187]

Microsoft and S. Weston. *foreach: Provides foreach looping construct*, 2022b. URL https://CRAN.R-project.org/package=foreach. R package version 1.5.2. [p187]

K. Newman, R. King, V. Elvira, P. de Valpine, R. S. McCrea, and B. J. T. Morgan. State-space models for ecological time-series data: Practical model-fitting. *Methods in Ecology and Evolution*, 14(1):26–42, 2023. doi: 10.1111/2041-210X.13833. [p195]

N. Ning, E. L. Ionides, and Y. Ritov. Scalable monte carlo inference and rescaled local asymptotic normality. *Bernoulli*, 27:2532–2555, 2021. doi: 10.3150/20-BEJ1321. [p194]

S. L. Ranjeva, E. B. Baskerville, V. Dukic, L. L. Villa, E. Lazcano-Ponce, A. R. Giuliano, G. Dwyer, and S. Cobey. Recurring infection with ecologically distinct HPV types can explain high prevalence and diversity. *Proceedings of the National Academy of Sciences of the USA*, 114(51):13573–13578, 2017. doi: 10.1073/pnas.1714712114. [p180]

S. L. Ranjeva, R. Subramanian, V. J. Fang, G. M. Leung, D. K. M. Ip, R. A. P. M. Perera, J. S. M. Peiris, B. J. Cowling, and S. Cobey. Age-specific differences in the dynamics of protective immunity to influenza. *Nature Communications*, 10(1):1660, 2019. doi: 10.1038/s41467-019-09652-6. [p180]

E. Romero-Severson, E. Volz, J. Koopman, T. Leitner, and E. L. Ionides. Dynamic variation in sexual contact rates in a cohort of HIV-negative gay men. *American Journal of Epidemiology*, 182:255–262, 2015. doi: 10.1093/aje/kwv044. [p183, 186]

M. J. Simpson and O. J. Maclaren. Profile-wise analysis: A profile likelihood-based workflow for identifiability analysis, estimation, and prediction with mechanistic mathematical models. *PLoS Computational Biology*, 19(9):e1011515, 2023. doi: 10.1371/journal.pcbi.1011515. [p192]

S. A. Sisson, Y. Fan, and M. M. Tanaka. Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765, 2007. doi: 10.1073/pnas.0607208104. [p187]

J. W. Smith, R. Q. Thomas, and L. R. Johnson. Parameterizing lognormal state space models using moment matching. *Environmental and Ecological Statistics*, 30(3):385–419, 2023. doi: 10.1007/s10651-023-00570-x. [p182]

E. Vittinghoff, J. Douglas, F. Judon, D. McKiman, K. MacQueen, and S. P. Buchbinder. Per-contact risk of human immunodeficiency virus transmission between male sexual partners. *American Journal of Epidemiology*, 150(3):306–311, 1999. ISSN 0002-9262. doi: 10.1093/oxfordjournals.aje.a010003. [p183]

H. Wickham. *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p194]

H. Wickham. *Advanced R*. Chapman and Hall/CRC, 2019. ISBN 9781351201315. doi: 10.1201/9781351201315. [p182]

C. P. Winsor. The Gompertz curve as a growth curve. *Proceedings of the National Academy of Sciences of the USA*, 18:1–8, 1932. doi: 10.1073/pnas.18.1.1. [p182]

Carles Bretó
Universitat de València
Department of Economic Analysis
Valencia, Spain
ORCiD: 0000-0003-4695-4902
carles.breto@uv.es

Jesse Wheeler
University of Michigan
Department of Statistics
Ann Arbor, Michigan
ORCiD: 0000-0003-3941-3884
jeswheel@umich.edu

Aaron A. King
University of Michigan and Santa Fe Institute
Department of Ecology and Evolutionary Biology

Ann Arbor, Michigan
ORCiD: 0000-0001-6159-3207
kingaa@umich.edu

Edward L. Ionides
University of Michigan
Department of Statistics
Ann Arbor, Michigan
ORCiD: 0000-0002-4190-0174
ionides@umich.edu

SimEngine: A Modular Framework for Statistical Simulations in R

by Avi Kenny and Charles J. Wolock

Abstract This article describes **SimEngine**, an open-source R package for structuring, maintaining, running, and debugging statistical simulations on both local and cluster-based computing environments. Several R packages exist for facilitating simulations, but **SimEngine** is the only package specifically designed for running simulations in parallel via job schedulers on high-performance cluster computing systems. The package provides structure and functionality for common simulation tasks, such as setting simulation levels, managing seeds for random number generation, and calculating summary metrics (such as bias and confidence interval coverage). **SimEngine** also brings several unique features, such as automatic calculation of Monte Carlo error and information sharing across simulation replicates. We provide an overview of the package and demonstrate some of its advanced functionality.

1 Introduction

For the past several decades, the design and execution of simulation studies has been a pillar of methodological research in statistics (Hauck and Anderson, 1984). Simulations are commonly used to evaluate finite sample performance of proposed statistical procedures, but can also be used to identify problems with existing methods, ensure that statistical code functions as designed, and test out new ideas in an exploratory manner. Additionally, simulation can be a statistical method in itself; two common examples are study power calculation (Arnold et al., 2011) and sampling from a complex distribution (Wakefield, 2013).

Although the power of personal computers has increased exponentially over the last four decades, many simulation studies require far more computing power than what is available on even a high-end laptop. Accordingly, many academic (bio)statistics departments and research institutions have invested in so-called cluster computing systems (CCS), which are essentially networks of servers available for high-throughput parallel computation. A single CCS typically serves many researchers, can be securely accessed remotely, and operates job scheduling software (e.g., Slurm) designed to coordinate the submission and management of computing tasks between multiple groups of users.

Thankfully, the majority of statistical simulations can be easily parallelized, since they typically involve running the same (or nearly the same) code many times and then performing an analysis of the results of these replicates. This allows for a simulation study to be done hundreds or thousands of times faster than if the user ran the same code on their laptop. Despite this, many potential users never leverage an available CCS; a major reason for this is that it can be difficult to get R and the CCS to "work together," even for an experienced programmer.

To address this gap, we created **SimEngine**, an open-source R package (R Core Team, 2021) for structuring, maintaining, running, and debugging statistical simulations on both local and cluster-based computing environments. Several R packages exist for structuring simulations (see Section 2.6); however, **SimEngine** is the only package specifically designed for running simulations both locally and in parallel on a high-performance CCS. The package provides structure and functionality for common simulation tasks, such as setting simulation levels, managing random number generator (RNG) seeds, and calculating summary metrics (such as bias and confidence interval coverage). In addition, **SimEngine** offers a number of unique features, such as automatic calculation of Monte Carlo error (Koehler et al., 2009) and information sharing across simulation replicates.

This article is organized as follows. In Section 2.2, we outline the overarching design principles of the package. In Section 2.3, we give a broad overview of the **SimEngine**

simulation workflow. In Section 2.4, we describe how simulations can be parallelized both locally and on a CCS. Section 2.5 contains details on advanced functionality of the package. The Appendix includes two example simulation studies carried out using **SimEngine**.

2 Design principles

There are four main principles that guided the design of **SimEngine**, which we refer to as (1) generality, (2) modularity, (3) parallelizability, and (4) appropriate scope. We discuss each in turn.

The first principle is *generality*. Many simulation frameworks assume that users will have a particular type of workflow that involves a data generation step, an analysis step, and an evaluation step. This three-step workflow is common — in fact, we use it to illustrate the use of the package in the introductory documentation — but we also do not want to impose this workflow on the user and disallow other possible workflows. In **SimEngine**, instead of providing facilities for the user to specify a data-generating step, an analysis step, and an evaluation step, the user writes R code representing a single simulation replicate within a special function called the *simulation script*. This imposes virtually no constraints on what the user is able to do within their simulations, and allows for workflows that cannot be shoehorned into this three-step process, such as simulations involving resampling from existing datasets, complex data transformations, saving intermediate objects or plots, and so on. The only requirement of the simulation script is that it returns data in a specific format (to facilitate processing of results), but this format is completely general and allows for arbitrarily complex objects (matrices, dataframes, model objects, etc.) to be returned in addition to simple numeric values.

The second principle is *modularity*. In short, it should be simple and straightforward to change one component of a simulation without breaking other components, such as adding a new estimator or changing a sample size. Additionally, it should be easy to run additional simulation replicates, possibly with certain elements changed, without having to rerun the entire simulation. Finally, the step of evaluating the results of a simulation should be completely separated from the step of actually running the simulation replicates. This implies that a user does not need to decide in advance how they are going to summarize or visualize their results, and that they can calculate new summary statistics, produce new visualizations, and otherwise process their simulation results in an exploratory manner, possibly long after the simulation itself has been run. In short, we designed **SimEngine** to mirror the real-world workflow of statistical simulations, in which researchers often make small changes in response to new ideas or results.

The third principle is *parallelizability*. As mentioned in Section 2.1, a primary reason for creating **SimEngine** was to build a package that encapsulated the repetitive tasks related to both local and CCS parallelization. We have seen firsthand that many statistical researchers run time-consuming simulations serially on their laptops because the barrier to entry for parallelizing code, particularly on a CCS, is daunting. It is our hope that making parallelization as easy as possible will allow researchers to easily leverage parallelization hardware and boost productivity.

The fourth principle is *appropriate scope*. We designed **SimEngine** to *only* provide functionality related to simulations. While some other packages provide functionality to plot results, for example, we intentionally choose to not do so, since all analysts have their own preferences when it comes to displaying data. Similarly, unlike many packages, we do not provide functionality to generate certain data structures, since this would bloat the package and still only satisfy the needs of a small handful of users.

Finally, although not a software design principle per se, an additional goal of **SimEngine** was to create documentation that assumes as little statistical knowledge as possible. Different statisticians have different areas of background knowledge, and we wanted to avoid having potential users sidetracked by trying to understand the statistics of a particular example rather than understand the functions and workflow of **SimEngine**. Accordingly, we strove

to create the simplest possible illustrations and examples in the package documentation. In general, we aimed to make the documentation as comprehensive and accessible as possible, while remaining concise and minimizing the barriers to entry.

3 Overview of simulation workflow and primary functions

The latest stable version of **SimEngine** can be installed from CRAN using `install.packages`. The current development version can be installed using `devtools::install_github`.

```
R> install.packages("SimEngine")
R> devtools::install_github(repo="Avi-Kenny/SimEngine")
```

The goal of many statistical simulations is to compare the behavior of two or more statistical methods; we use this framework to demonstrate the **SimEngine** workflow. Most statistical simulations of this type include three basic phases: (1) generate data, (2) run one or more methods using the generated data, and (3) compare the performance of the methods.

To briefly illustrate how these phases are implemented using **SimEngine**, we use a simple example of estimating the rate parameter λ of a Poisson(λ) distribution. To anchor the simulation in a real-world situation, one can imagine that a sample of size n from this Poisson distribution models the number of patients admitted daily to a hospital over the course of n consecutive days. Suppose that the data consist of n independent and identically distributed observations $X_1, X_2, \ldots, X_n$ drawn from a Poisson(λ) distribution. Since the λ parameter of the Poisson distribution is equal to both the mean and the variance, one may ask whether the sample mean $\hat{\lambda}_{M,n} := \frac{1}{n}\sum_{i=1}^{n} X_i$ or the sample variance $\hat{\lambda}_{V,n} := \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \hat{\lambda}_{M,n})^2$ is a better estimator of λ.

3.1 Load the package and create a simulation object

After loading the package, the first step is to create a simulation object (an R object of class `sim_obj`) using the `new_sim` function. The simulation object contains all data, functions, and results related to the simulation.

```
R> library(SimEngine)
R> set.seed(1)
R> sim <- new_sim()
```

3.2 Code a function to generate data

Many simulations involve a function that creates a dataset designed to mimic a real-world data-generating mechanism. Here, we write and test a simple function to generate a sample of `n` observations from a Poisson distribution with $\lambda = 20$.

```
R> create_data <- function(n) {
+    return(rpois(n=n, lambda=20))
+  }
R> create_data(n=10)
 [1] 18 25 25 21 13 22 23 22 18 26
```

3.3 Code the methods (or other functions)

With **SimEngine**, any functions declared (or loaded via `source`) are automatically stored in the simulation object when the simulation runs. In this example, we test the sample mean and sample variance estimators of the λ parameter. For simplicity, we write this as a single function and use the `type` argument to specify which estimator to use.

```
R> est_lambda <- function(dat, type) {
+    if (type=="M") { return(mean(dat)) }
+    if (type=="V") { return(var(dat)) }
+  }
R> dat <- create_data(n=1000)
R> est_lambda(dat=dat, type="M")
[1] 19.646
R> est_lambda(dat=dat, type="V")
[1] 20.8195
```

3.4 Set the simulation levels

Often, we wish to run the same simulation multiple times. We refer to each run as a *simulation replicate*. We may wish to vary certain features of the simulation between replicates. In this example, perhaps we choose to vary the sample size and the estimator used to estimate λ. We refer to the features that vary as *simulation levels*; in the example below, the simulation levels are the sample size (`n`) and the estimator (`estimator`). We refer to the values that each simulation level can take on as *level values*; in the example below, the `n` level values are `10`, `100`, and `1000`, and the `estimator` level values are ``M'' (for "sample mean") and ``V'' (for "sample variance"). We also refer to a combination of level values as a *scenario*; in this example, the combination of `n=10` and `estimator=``M''` is one of the six possible scenarios defined by the two values of `n` and the three values of `estimator`. By default, **SimEngine** runs one simulation replicate for each scenario, although the user will typically want to increase this; 1,000 or 10,000 replicates per scenario is common. An appropriate number of replicates per scenario may be informed by the desired level of Monte Carlo error; see Section 2.5.6.

```
R> sim %<>% set_levels(
+    estimator = c("M", "V"),
+    n = c(10, 100, 1000)
+  )
```

Note that we make extensive use of the pipe operators (`%>%` and `%<>%`) from the **magrittr** package (Bache and Wickham, 2022). Briefly, the operator `%>%` takes the object to the left of the operator and "pipes" it to (the first argument of) the function to the right of the operator; the operator `%<>%` does the same thing, but then assigns the result back to the variable to the left of the operator. For example, `x %>% mean()` is equivalent to `mean(x)` and `x %<>% mean()` is equivalent to `x <-mean(x)`. See the **magrittr** documentation for further detail.

3.5 Create a simulation script

The simulation script is a user-written function that assembles the pieces above (generating data, analyzing the data, and returning results) to code the flow of a single simulation replicate. Within a script, the level values for the current scenario can be referenced using the special variable `L`. For instance, in the running example, when the first simulation replicate is running, `L$estimator` will equal ``M'' and `L$n` will equal `10`. In the next replicate, `L$estimator` will equal ``M'' and `L$n` will equal `100`, and so on. The simulation script will automatically have access to any functions or objects that have been declared in the global environment.

```
R> sim %<>% set_script(function() {
+    dat <- create_data(n=L$n)
+    lambda_hat <- est_lambda(dat=dat, type=L$estimator)
+    return (list("lambda_hat"=lambda_hat))
+  })
```

The simulation script should always return a list containing one or more key-value pairs, where the keys are syntactically valid names. The values may be simple data types (numbers, character strings, or boolean values) or more complex data types (lists, dataframes, model objects, etc.); see Section 2.5.3 for how to handle complex data types. Note that in this example, the estimators could have been coded instead as two different functions and then called from within the script using the `use_method` function.

3.6 Set the simulation configuration

The `set_config` function controls options related to the entire simulation, such as the number of simulation replicates to run for each scenario and the parallelization type, if desired (see Section 2.4). Packages needed for the simulation should be specified using the `packages` argument of `set_config` (rather than using `library` or `require`). We set `num_sim` to 100, and so **SimEngine** will run a total of 600 simulation replicates (100 for each of the six scenarios).

```
R> sim %<>% set_config(
+    num_sim = 100,
+    packages = c("ggplot2", "stringr")
+  )
```

3.7 Run the simulation

All 600 replicates are run at once and results are stored in the simulation object.

```
R> sim %<>% run()
  |########################################| 100%
Done. No errors or warnings detected.
```

3.8 View and summarize results

Once the simulation replicates have finished running, the `summarize` function can be used to calculate common summary statistics, such as bias, variance, mean squared error (MSE), and confidence interval coverage.

```
R> sim %>% summarize(
+    list(stat="bias", name="bias_lambda", estimate="lambda_hat", truth=20),
+    list(stat="mse", name="mse_lambda", estimate="lambda_hat", truth=20)
+  )
  level_id estimator    n n_reps bias_lambda   mse_lambda
1        1         M   10    100 -0.17600000   1.52480000
2        2         V   10    100 -1.80166667 103.23599630
3        3         M  100    100 -0.03770000   0.19165100
4        4         V  100    100  0.22910707   7.72262714
5        5         M 1000    100  0.01285000   0.01553731
6        6         V 1000    100  0.02514744   0.92133037
```

In this example, we see that the MSE of the sample variance is much higher than that of the sample mean and that MSE decreases with increasing sample size for both estimators, as expected. From the `n_reps` column, we see that 100 replicates were successfully run for each scenario. Results for individual simulation replicates can also be directly accessed via the `sim$results` dataframe.

```
R> head(sim$results)
  sim_uid level_id rep_id estimator  n      runtime lambda_hat
1       1        1      1         M 10 0.0003290176       20.1
```

```
2       7        1      2         M 10 0.0002038479       20.5
3       8        1      3         M 10 0.0001709461       17.3
4       9        1      4         M 10 0.0001630783       20.3
5      10        1      5         M 10 0.0001599789       18.3
6      11        1      6         M 10 0.0001561642       20.4
```

Above, the `sim_uid` uniquely identifies a single simulation replicate and the `level_id` uniquely identifies a scenario (i.e., a combination of level values). The `rep_id` is unique within a given scenario and identifies the index of that replicate within the scenario. The `runtime` column shows the runtime of each replicate (in seconds).

3.9 Update a simulation

After running a simulation, a user may want to update it by adding additional level values or replicates; this can be done with the `update_sim` function. Prior to running `update_sim`, the functions `set_levels` and/or `set_config` are used to declare the updates that should be performed. For example, the following code sets the total number of replicates to 200 (i.e., adding 100 replicates to those that have already been run) for each scenario, and adds one additional level value for `n`.

```
R> sim %<>% set_config(num_sim = 200)
R> sim %<>% set_levels(
+    estimator = c("M", "V"),
+    n = c(10, 100, 1000, 10000)
+  )
```

After the levels and/or configuration are updated, `update_sim` is called.

```
R> sim %<>% update_sim()
  |######################################| 100%
Done. No errors or warnings detected.
```

Another call to `summarize` shows that the additional replicates were successful:

```
R> sim %>% summarize(
+    list(stat="bias", name="bias_lambda", estimate="lambda_hat", truth=20),
+    list(stat="mse", name="mse_lambda", estimate="lambda_hat", truth=20)
+  )
  level_id estimator     n n_reps  bias_lambda   mse_lambda
1        1         M    10    200 -0.205500000  1.875450000
2        2         V    10    200 -1.189166667 96.913110494
3        3         M   100    200 -0.055000000  0.197541000
4        4         V   100    200  0.023244949  7.955606709
5        5         M  1000    200  0.017495000  0.017497115
6        6         V  1000    200  0.053941807  0.874700025
7        7         M 10000    200 -0.005233000  0.002096102
8        8         V 10000    200 -0.007580998  0.072997135
```

It is also possible to delete level values. However, it is not possible to add or delete *levels*, as this would require updating the simulation script after the simulation has already run, which is not allowed.

4 Parallelization

User-friendly parallelization is a hallmark of **SimEngine**. There are two modes of parallelizing code using **SimEngine**, which we refer to as *local parallelization* and *cluster parallelization*.

Local parallelization refers to splitting the computational work of a simulation between multiple cores of a single computer (e.g., a multicore laptop). Cluster parallelization refers to running a simulation on a CCS using job arrays. **SimEngine** is designed to automate as much of the parallelization process as possible. We give an overview of each parallelization mode below.

4.1 Local parallelization

Local parallelization is the easiest way to parallelize code, as the entire process is handled by the package and executed on the user's computer. This mode is activated using `set_config`, as follows.

```
R> sim <- new_sim()
R> sim %<>% set_config(parallel = TRUE)
```

SimEngine handles the mechanics related to parallelization internally using the base R package **parallel** (R Core Team, 2021). If a single simulation replicate runs in a very short amount of time (e.g., less than one second), using local parallelization can actually result in an *increase* in total runtime. This is because there is a certain amount of computational overhead involved in the parallelization mechanisms inside **SimEngine**. A speed comparison can be performed by running the code twice, once with `set_config(parallel = TRUE)` and once with `set_config(parallel = FALSE)`, each followed by `sim %>% vars("total_runtime")`, to see the difference in total runtime. The exact overhead involved with local parallelization will differ between machines. A simple example of a speed comparison is given below:

```
R> for (p in c(FALSE, TRUE)) {
+    sim <- new_sim()
+    sim %<>% set_config(num_sim=20, parallel=p)
+    sim %<>% set_script(function() {
+      Sys.sleep(1)
+      return (list("x"=1))
+    })
+    sim %<>% run()
+    print(paste("Parallelizing:", p))
+    print(sim %>% vars("total_runtime"))
}
  |######################################| 100%
Done. No errors or warnings detected.

[1] "Parallelizing: FALSE"
[1] 20.42414
Done. No errors or warnings detected.

[1] "Parallelizing: TRUE"
[1] 4.41772
```

Removing the line `Sys.sleep(1)` and rerunning the code above can give a sense of the amount of overhead time incurred for a given simulation and hardware configuration. If the user's computer has `n` cores available, **SimEngine** will use `n-1` cores by default. The `n_cores` argument of `set_config` can be used to manually specify the number of cores to use, as follows.

```
R> sim %<>% set_config(n_cores = 2)
```

4.2 Cluster parallelization

Parallelizing code using a CCS is more complicated, but **SimEngine** is built to streamline this process as much as possible. A CCS is a supercomputer that consists of a number of nodes, each of which may have multiple cores. Typically, a user logs into the CCS and runs programs by submitting "jobs" to the CCS using a special program called a job scheduler. The job scheduler manages the process of running the jobs in parallel across multiple nodes and/or multiple cores. Although there are multiple ways to run code in parallel on a CCS, **SimEngine** makes use of job arrays. The main cluster parallelization function in **SimEngine** is `run_on_cluster`. Throughout this example, we use Slurm as an example job scheduler, but an analogous workflow will apply to other job scheduling software.

To illustrate the cluster parallelization workflow, consider the simulation from Section 2.3.

```
R> sim <- new_sim()
R> create_data <- function(n) { return(rpois(n=n, lambda=20)) }
R> est_lambda <- function(dat, type) {
+    if (type=="M") { return(mean(dat)) }
+    if (type=="V") { return(var(dat)) }
+  }
R> sim %<>% set_levels(estimator = c("M","V"), n = c(10,100,1000))
R> sim %<>% set_script(function() {
+    dat <- create_data(L$n)
+    lambda_hat <- est_lambda(dat=dat, type=L$estimator)
+    return(list("lambda_hat"=lambda_hat))
+  })
R> sim %<>% set_config(num_sim=100)
R> sim %<>% run()
R> sim %>% summarize()
```

To run this code on a CCS, we simply wrap it in the `run_on_cluster` function. To use this function, we must break the code into three blocks, called `first`, `main`, and `last`. The code in the `first` block will run only once, and will set up the simulation object. When this is finished, **SimEngine** will save the simulation object in the filesystem of the CCS. The code in the `main` block will then run once for each simulation replicate, and will have access to the simulation object created in the `first` block. In most cases, the code in the `main` block will simply include a single call to `run` (or to `update_sim`, as detailed below). Finally, the code in the `last` block will run after all simulation replicates have finished running, and after **SimEngine** has automatically compiled the results into the simulation object. Use of the `run_on_cluster` function is illustrated below:

```
R> run_on_cluster(
+    first = {
+      sim <- new_sim()
+      create_data <- function(n) { return(rpois(n=n, lambda=20)) }
+      est_lambda <- function(dat, type) {
+        if (type=="M") { return(mean(dat)) }
+        if (type=="V") { return(var(dat)) }
+      }
+      sim %<>% set_levels(estimator = c("M","V"), n = c(10,100,1000))
+      sim %<>% set_script(function() {
+        dat <- create_data(L$n)
+        lambda_hat <- est_lambda(dat=dat, type=L$estimator)
+        return(list("lambda_hat"=lambda_hat))
+      })
+      sim %<>% set_config(num_sim=100, n_cores=20)
```

```
+    },
+    main = {
+      sim %<>% run()
+    },
+    last = {
+      sim %>% summarize()
+    },
+    cluster_config = list(js="slurm")
+  )
```

Note that none of the actual simulation code changed (with the exception of specifying `n_cores=20` in the `set_config` call); we simply divided the code into chunks and placed these chunks into the appropriate block (`first`, `main`, or `last`) within `run_on_cluster`. Additionally, we specified which job scheduler to use in the `cluster_config` argument list. The command `js_support` can be run in R to see a list of supported job scheduler software; the value in the `js_code` column is the value that should be specified in the `cluster_config` argument. Unsupported job schedulers can still be used for cluster parallelization, as detailed below. Note that the `cluster_config` argument can also be used to specify which folder on the cluster should be used to store the simulation object and associated simulation files (the default is the working directory).

Next, we must give the job scheduler instructions on how to run the above code. In the following, we assume that the R code above is stored in a file called `my_simulation.R`. We also need to create a simple shell script called `run_sim.sh` with the following two lines, which will run `my_simulation.R` (we demonstrate this using BASH scripting language, but any shell scripting language may be used).

```
> #!/bin/bash
> Rscript my_simulation.R
```

If created on a local machine, the two simulation files (`my_simulation.R` and `run_sim.sh`) must be transferred to the filesystem of the CCS. Finally, we use the job scheduler to submit three jobs. The first will run the `first` code, the second will run the `main` code, and the third will run the `last` code. With Slurm, we run the following three shell commands:

```
> sbatch --export=sim_run='first' run_sim.sh
Submitted batch job 101
> sbatch --export=sim_run='main' --array=1-20 --depend=afterok:101 run_sim.sh
Submitted batch job 102
> sbatch --export=sim_run='last' --depend=afterok:102 run_sim.sh
Submitted batch job 103
```

In the first line, we submit the `run_sim.sh` script using the `sim_run=`first'` environment variable, which tells **SimEngine** to only run the code in the `first` block. After running this, Slurm returns the message `Submitted batch job 101`. The number `101` is called the "job ID" and uniquely identifies the job on the CCS.

In the second line, we submit the `run_sim.sh` script using the `sim_run=`main'` environment variable and tell Slurm to run a job array with "task IDs" 1-20. Each task corresponds to one core, and so in this case 20 cores will be used. This number should equal the `n_cores` number specified via `set_config`. **SimEngine** handles the work of dividing the simulation replicates between the cores; the only restriction is that the number of cores cannot exceed the total number of simulation replicates.

Also note that we included the option `--depend=afterok:101`, which instructs the job scheduler to wait until the first job finishes before starting the job array. (In practice, the number 101 must be replaced with whatever job ID Slurm assigned to the first job.) Once this command is submitted, the code in the `main` block will be run for each replicate. A temporary

folder called `sim_results` will be created and filled with temporary objects containing data on the results and/or errors for each replicate.

In the third line, we submit the `run_sim.sh` script using the `sim_run='last'` environment variable. Again, we use `--depend=afterok:102` to ensure this code does not run until all tasks in the job array have finished. When this job runs, **SimEngine** will compile the results from the `main` block, run the code in the `last` block, save the simulation object to the filesystem, and delete the temporary `sim_results` folder and its contents. If desired, the user can leave the `last` block empty, but this third `sbatch` command should be run anyway to compile the results and save the simulation object for further analysis.

Further automating job submission

Advanced users may wish to automatically capture the job IDs so that they don't need to be entered manually; sample code showing how this can be done is shown below:

```
> jid1=$(sbatch --export=sim_run='first' run_sim.sh | sed 's/Submitted batch job //')
> jid2=$(sbatch --export=sim_run='main' --array=1-20 --depend=afterok:$jid1 \
    run_sim.sh | sed 's/Submitted batch job //')
> sbatch --export=sim_run='last' --depend=afterok:$jid2 run_sim.sh
Submitted batch job 103
```

While this is slightly more complicated, this code allows all three lines to be submitted simultaneously without the need to copy and paste the job IDs manually every time.

4.3 Additional cluster parallelization functionality

Running locally

The `run_on_cluster` function is programmed such that it can also be run locally. In this case, the code within the `first`, `main`, and `last` blocks will be executed in the calling environment of the `run_on_cluster` function (typically the global environment); this can be useful for testing simulations locally before sending them to a CCS.

Using unsupported job schedulers

There may be job schedulers that **SimEngine** does not natively support. If this is the case, **SimEngine** can still be used for cluster parallelization; this requires identifying the environment variable that the job scheduler uses to uniquely identify tasks within a job array. For example, Slurm uses the variable `"SLURM_ARRAY_TASK_ID"` and Grid Engine uses the variable `"SGE_TASK_ID"`. Once this variable is identified, it can be specified in the `cluster_config` block, as follows:

```
R> run_on_cluster(
+    first = {...},
+    main = {...},
+    last = {...},
+    cluster_config = list(tid_var="SLURM_ARRAY_TASK_ID")
+  )
```

Updating a simulation on a CCS

To update a simulation on a CCS, the `update_sim_on_cluster` function can be used. The workflow is similar to that of `run_on_cluster`, with several key differences. Instead of creating a new simulation object in the `first` block using `new_sim`, the existing simulation object (which would have been saved to the filesystem when `run_on_cluster` was called

originally) is loaded using `readRDS`. Then, the functions `set_levels` and/or `set_config` are called to specify the desired updates (see Section 2.3.9). In the `main` block, `update_sim` is called (instead of `run`). In the `last` block, code can remain the same or change as needed. These differences are illustrated in the code below.

```
R> update_sim_on_cluster(
+    first = {
+      sim <- readRDS("sim.rds")
+      sim %<>% set_levels(n=c(100,500,1000))
+    },
+    main = {
+      sim %<>% update_sim()
+    },
+    last = {
+      sim %>% summarize()
+    },
+    cluster_config = list(js="slurm")
+  )
```

Submission of this code via a job scheduler proceeds in the same manner as described earlier for `run_on_cluster`.

5 Advanced functionality

In this section, we review the following functionality, targeting advanced users of the package:

- using the `batch` function, which allows for information to be shared across simulation replicates;
- using *complex simulation levels* in settings where simple levels (e.g., a vector of numbers) are insufficient;
- handling *complex return data* in settings where a single simulation replicate returns nested lists, dataframes, model objects, etc.;
- managing RNG seeds;
- best practices for debugging and handling errors and warnings;
- capturing Monte Carlo error using the `summarize` function.

5.1 Using the batch function to share information across simulation replicates

The `batch` function is useful for sharing data or objects between simulation replicates. Essentially, it allows simulation replicates to be divided into "batches;" all replicates in a given batch will then share a certain set of objects. A common use case for this is a simulation that involves using multiple methods to analyze a shared dataset, and repeating this process over a number of dataset replicates. This may be of interest if, for example, it is computationally expensive to generate a simulated dataset.

To illustrate the use of `batch` using this example, we first consider the following simulation:

```
R> sim <- new_sim()
R> create_data <- function(n) { rnorm(n=n, mean=3) }
R> est_mean <- function(dat, type) {
+    if (type=="est_mean") { return(mean(dat)) }
```

```
+    if (type=="est_median") { return(median(dat)) }
+  }
R> sim %<>% set_levels(est=c("est_mean","est_median"))
R> sim %<>% set_config(num_sim=3)
R> sim %<>% set_script(function() {
+    dat <- create_data(n=100)
+    mu_hat <- est_mean(dat=dat, type=L$est)
+    return(list(
+      "mu_hat" = round(mu_hat,2),
+      "dat_1" = round(dat[1],2)
+    ))
+  })
R> sim %<>% run()
  |######################################| 100%
Done. No errors or warnings detected.
```

From the ``dat_1'' column of the results object (equal to the first element of the `dat` vector created in the simulation script), we see that a unique dataset was created for each simulation replicate:

```
R> sim$results[order(sim$results$rep_id),c(1:7)!=5]
  sim_uid level_id rep_id        est mu_hat dat_1
1       1        1      1   est_mean   3.05  4.09
4       2        2      1 est_median   3.06  3.23
2       3        1      2   est_mean   3.03  2.99
5       5        2      2 est_median   3.02  2.78
3       4        1      3   est_mean   2.85  1.47
6       6        2      3 est_median   3.03  2.35
```

Suppose that instead, we wish to analyze each simulated dataset using multiple methods (in this case corresponding to ``est_mean'' and ``est_median''), and repeat this procedure a total of three times. We can do this using the `batch` function, as follows:

```
R> sim <- new_sim()
R> create_data <- function(n) { rnorm(n=n, mean=3) }
R> est_mean <- function(dat, type) {
+    if (type=="est_mean") { return(mean(dat)) }
+    if (type=="est_median") { return(median(dat)) }
+  }
R> sim %<>% set_levels(est=c("est_mean","est_median"))
R> sim %<>% set_config(num_sim=3, batch_levels=NULL)
R> sim %<>% set_script(function() {
+    batch({
+      dat <- create_data(n=100)
+    })
+    mu_hat <- est_mean(dat=dat, type=L$est)
+    return(list(
+      "mu_hat" = round(mu_hat,2),
+      "dat_1" = round(dat[1],2)
+    ))
+  })
R> sim %<>% run()
  |######################################| 100%
Done. No errors or warnings detected.
```

In the code above, we changed two things. First, we added `batch_levels=NULL` to the `set_config` call; this will be explained below. Second, we wrapped the code line `dat`

`<-create_data(n=100)` inside the `batch` function. Whatever code goes inside the `batch` function will produce the same output for all simulations in a batch.

```
R> sim$results[order(sim$results$rep_id),c(1:7)!=5]
  sim_uid level_id rep_id        est mu_hat dat_1
1       1        1      1   est_mean   3.02  2.74
4       2        2      1 est_median   3.19  2.74
2       3        1      2   est_mean   2.91  3.71
5       5        2      2 est_median   2.95  3.71
3       4        1      3   est_mean   3.10  3.52
6       6        2      3 est_median   3.01  3.52
```

In this case, from the ``dat_1'' column of the results object, we see that one dataset was created and shared by the batch corresponding to `sim_uids` 1 and 2 (likewise for `sim_uids` $\{3,5\}$ and $\{4,6\}$).

However, the situation is often more complicated. Suppose we have a simulation with multiple levels, some that correspond to creating data and some that correspond to analyzing the data. Here, the `batch_levels` argument of `set_config` plays a role. Specifically, this argument should be a character vector equal to the names of the simulation levels that are referenced (via the special variable `L`) from within a `batch` block. In the example below, the levels `n` and `mu` are used within the `batch` call, while the level `est` is not.

```
R> sim <- new_sim()
R> create_data <- function(n, mu) { rnorm(n=n, mean=mu) }
R> est_mean <- function(dat, type) {
+    if (type=="est_mean") { return(mean(dat)) }
+    if (type=="est_median") { return(median(dat)) }
+  }
R> sim %<>% set_levels(n=c(10,100), mu=c(3,5), est=c("est_mean","est_median"))
R> sim %<>% set_config(num_sim=2, batch_levels=c("n", "mu"), return_batch_id=T)
R> sim %<>% set_script(function() {
+    batch({
+      dat <- create_data(n=L$n, mu=L$mu)
+    })
+    mu_hat <- est_mean(dat=dat, type=L$est)
+    return(list(
+      "mu_hat" = round(mu_hat,2),
+      "dat_1" = round(dat[1],2)
+    ))
+  })
R> sim %<>% run()
  |########################################| 100%
Done. No errors or warnings detected.
R> sim$results[order(sim$results$batch_id),c(1:10)!=8]
   sim_uid level_id rep_id batch_id   n mu        est mu_hat dat_1
1        1        1      1        1  10  3   est_mean   2.87  4.29
9        5        5      1        1  10  3 est_median   2.94  4.29
2        9        1      2        2  10  3   est_mean   2.79  2.77
10      13        5      2        2  10  3 est_median   2.73  2.77
3        2        2      1        3 100  3   est_mean   2.93  1.77
11       6        6      1        3 100  3 est_median   3.01  1.77
4       10        2      2        4 100  3   est_mean   2.80  4.44
12      14        6      2        4 100  3 est_median   2.71  4.44
5        3        3      1        5  10  5   est_mean   5.49  4.78
13       7        7      1        5  10  5 est_median   5.25  4.78
6       11        3      2        6  10  5   est_mean   4.57  4.48
```

```
14      15       7      2       6  10  5 est_median   4.62  4.48
7        4       4      1       7 100  5   est_mean   4.98  5.66
15       8       8      1       7 100  5 est_median   4.95  5.66
8       12       4      2       8 100  5   est_mean   5.08  5.55
16      16       8      2       8 100  5 est_median   5.14  5.55
```

The batches were created such that each batch contained two replicates, one for each level value of `est`. For expository purposes, we also specified the `return_batch_id=T` option in `set_config` so that the results object would return the `batch_id`. This is not necessary in practice. The `batch_id` variable defines the batches; all simulations that share the same `batch_id` are in a single batch. The `return_batch_id=T` option can be useful to ensure correct usage of the `batch` function.

We note the following about the `batch` function:

- The code within the `batch` code block must *only* create objects; this code should not change or delete existing objects, as these changes will be ignored.
- In the majority of cases, the `batch` function will be called just once, at the beginning of the simulation script. However, it can be used anywhere in the script and can be called multiple times. The `batch` function should never be used outside of the simulation script.
- Although we have illustrated the use of the `batch` function to create a dataset to share between multiple simulation replicates, it can be used for much more, such as taking a sample from an existing dataset or computing shared nuisance function estimators.
- If the simulation is being run in parallel (either locally or on a CCS), `n_cores` cannot exceed the number of batches, since all simulations within a batch must run on the same core.
- If the simulation script uses the `batch` function, the simulation cannot be updated using the `update_sim` or `update_sim_on_cluster` functions, with the exception of updates that only entail removing simulation replicates.

5.2 Complex simulation levels

Often, simulation levels are simple, such as a vector of sample sizes:

```
R> sim <- new_sim()
R> sim %<>% set_levels(n = c(200,400,800))
```

However, there are many instances in which more complex objects are needed. For these cases, instead of a vector of numbers or character strings, a named list of lists can be used. The toy example below illustrates this.

```
R> sim <- new_sim()
R> sim %<>% set_levels(
+    n = c(10,100),
+    distribution = list(
+      "Beta 1" = list(type="Beta", params=c(0.3, 0.7)),
+      "Beta 2" = list(type="Beta", params=c(1.5, 0.4)),
+      "Normal" = list(type="Normal", params=c(3.0, 0.2))
+    )
+  )
R> create_data <- function(n, type, params) {
+    if (type=="Beta") {
+      return(rbeta(n, shape1=params[1], shape2=params[2]))
```

```
+    } else if (type=="Normal") {
+      return(rnorm(n, mean=params[1], sd=params[2]))
+    }
+  }
R> sim %<>% set_script(function() {
+    x <- create_data(L$n, L$distribution$type, L$distribution$params)
+    return(list("y"=mean(x)))
+  })
R> sim %<>% run()
  |########################################| 100%
Done. No errors or warnings detected.
```

Note that the list names (``Beta 1'', ``Beta 2'', and ``Normal'') become the entries in the `sim$results` dataframe, as well as the dataframe returned by `summarize`.

```
R> sim %>% summarize(list(stat="mean", x="y"))
  level_id   n distribution n_reps    mean_y
1        1  10       Beta 1      1 0.1635174
2        2 100       Beta 1      1 0.2740965
3        3  10       Beta 2      1 0.6234866
4        4 100       Beta 2      1 0.7664522
5        5  10       Normal      1 3.0903062
6        6 100       Normal      1 3.0179944
```

5.3 Complex return data

In most situations, the results of simulations are numeric. However, we may want to return more complex data, such as matrices, lists, or model objects. To do this, we add a key-value pair to the list returned by the simulation script with the special key `".complex"` and a list (containing the complex data) as the value. This is illustrated in the toy example below. Here, the simulation script estimates the parameters of a linear regression and returns these as numeric, but also returns the estimated covariance matrix and the entire model object.

```
R> sim <- new_sim()
R> sim %<>% set_levels(n=c(10, 100, 1000))
R> create_data <- function(n) {
+    x <- runif(n)
+    y <- 3 + 2*x + rnorm(n)
+    return(data.frame("x"=x, "y"=y))
+  }
R> sim %<>% set_config(num_sim=2)
R> sim %<>% set_script(function() {
+    dat <- create_data(L$n)
+    model <- lm(y~x, data=dat)
+    return(list(
+      "beta0_hat" = model$coefficients[[1]],
+      "beta1_hat" = model$coefficients[[2]],
+      ".complex" = list(
+        "model" = model,
+        "cov_mtx" = vcov(model)
+      )
+    ))
+  })
R> sim %<>% run()
  |########################################| 100%
Done. No errors or warnings detected.
```

After running this simulation, the numeric results can be accessed directly via `sim$results` or using the `summarize` function, as usual:

```
R> head(sim$results)
  sim_uid level_id rep_id    n     runtime beta0_hat beta1_hat
1       1        1      1   10 0.012123823  2.113012  3.780972
2       4        1      2   10 0.001345873  2.058610  3.726216
3       2        2      1  100 0.006520033  2.784147  2.058041
4       5        2      2  100 0.001568794  3.066045  2.097569
5       3        3      1 1000 0.001888037  3.144964  1.783498
6       6        3      2 1000 0.002427101  3.125128  1.714405
```

To examine the complex return data, we can use the special function `get_complex`, as illustrated below:

```
R> c5 <- get_complex(sim, sim_uid=5)
R> print(summary(c5$model))
Call:
lm(formula = y ~ x, data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-2.7050 -0.7429  0.1183  0.7470  1.9673

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.0660     0.2127  14.413  < 2e-16 ***
x             2.0976     0.3714   5.647 1.59e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.003 on 98 degrees of freedom
Multiple R-squared:  0.2455,        Adjusted R-squared:  0.2378
F-statistic: 31.89 on 1 and 98 DF,  p-value: 1.593e-07
R> print(c5$cov_mtx)
            (Intercept)           x
(Intercept)  0.04525148 -0.06968649
x           -0.06968649  0.13795995
```

5.4 Random number generator seeds

In statistical research, it is desirable to be able to reproduce the exact results of a simulation study. Since R code often involves stochastic (random) functions like `rnorm` or `sample` that return different values when called multiple times, reproducibility is not guaranteed. In a simple R script, calling the `set.seed` function at the beginning of the script ensures that the stochastic functions that follow will produce the same results whenever the script is run. However, a more nuanced strategy is needed when running simulations. When running 100 replicates of the same simulation, we do not want each replicate to return identical results; rather, we would like for each replicate to be different from one another, but for *the entire set of replicates* to be the same when the entire simulation is run twice in a row. **SimEngine** manages this process, even when simulations are being run in parallel locally or on a cluster computing system. In **SimEngine**, a single "global seed" is used to generate a different seed for each simulation replicate. The `set_config` function is used to set or change this global seed:

```
R> sim %<>% set_config(seed=123)
```

If a seed is not set using `set_config`, **SimEngine** will set a random seed automatically so that the results can be replicated if desired. To view this seed, we use the `vars` function:

```
R> sim <- new_sim()
R> print(vars(sim, "seed"))
[1] 287577520
```

5.5 Debugging and error/warning handling

In the simulation coding workflow, errors are inevitable. Some errors may affect all simulation replicates, while other errors may only affect a subset of replicates. By default, when a simulation is run, **SimEngine** will not stop if an error occurs; instead, errors are logged and stored in a dataframe along with information about the simulation replicates that resulted in those errors. Examining this dataframe by typing `print(sim$errors)` can sometimes help to quickly pinpoint the issue. This is demonstrated below:

```
R> sim <- new_sim()
R> sim %<>% set_config(num_sim=2)
R> sim %<>% set_levels(
+    Sigma = list(
+      s1 = list(mtx=matrix(c(3,1,1,2), nrow=2)),
+      s3 = list(mtx=matrix(c(4,3,3,9), nrow=2)),
+      s2 = list(mtx=matrix(c(1,2,2,1), nrow=2)),
+      s4 = list(mtx=matrix(c(8,2,2,6), nrow=2))
+    )
+  )
R> sim %<>% set_script(function() {
+    x <- MASS::mvrnorm(n=1, mu=c(0,0), Sigma=L$Sigma$mtx)
+    return(list(x1=x[1], x2=x[2]))
+  })
R> sim %<>% run()
  |########################################| 100%
Done. Errors detected in 25% of simulation replicates. Warnings detected in
0% of simulation replicates.
R> print(sim$errors)
  sim_uid level_id rep_id Sigma      runtime                          message
1       5        3      1    s2 0.0004692078 'Sigma' is not positive definite
2       6        3      2    s2 0.0006608963 'Sigma' is not positive definite
                                                                        call
1 MASS::mvrnorm(n = 1, mu = c(0, 0), Sigma = L$Sigma$mtx)
2 MASS::mvrnorm(n = 1, mu = c(0, 0), Sigma = L$Sigma$mtx)
```

From the output above, we see that the code fails for the simulation replicates that use the level with `Sigma="s2"` because it uses an invalid (not positive definite) covariance matrix. Similarly, if a simulation involves replicates that throw warnings, all warnings are logged and stored in the dataframe `sim$warnings`. If an error occurs for a subset of simulation replicates and that error is fixed, it is possible to rerun the replicates that had errors, as follows:

```
R> sim %<>% update_sim(keep_errors=FALSE)
```

The workflow demonstrated above can be useful to pinpoint errors, but it has two main drawbacks. First, it is undesirable to run a time-consuming simulation involving hundreds or thousands of replicates, only to find at the end that every replicate failed because of a typo. It may therefore be useful to stop an entire simulation after a single error has occurred. Second, it can sometimes be difficult to determine exactly what caused an error without

making use of more advanced debugging tools. For both of these situations, **SimEngine** includes the following configuration option:

```
R> sim %<>% set_config(stop_at_error=TRUE)
```

Setting `stop_at_error=TRUE` will stop the simulation when it encounters any error. Furthermore, the error will be thrown by R in the usual way, so if the simulation is being run in RStudio, the built-in debugging tools (such as "Show Traceback" and "Rerun with debug") can be used to find and fix the bug. Placing a call to `browser` at the top of the simulation script can also be useful for debugging.

5.6 Monte Carlo error

Statistical simulations are often based on the principle of Monte Carlo approximation; specifically, pseudo-random sampling is used to evaluate the performance of a statistical procedure under a particular data-generating process. The performance of the procedure can be viewed as a statistical parameter and, due to the fact that only a finite number of simulation replicates can be performed, there is uncertainty in any estimate of performance. This uncertainty is often referred to as *Monte Carlo error* (see, e.g., Lee and Young, 1999). We can quantify Monte Carlo error using, for example, the standard error of the performance estimator.

Measuring and reporting Monte Carlo error is a vital component of a simulation study. **SimEngine** includes an option in the `summarize` function to automatically estimate the Monte Carlo standard error for any inferential summary statistic, e.g., estimator bias or confidence interval coverage. The standard error estimates are based on the formulas provided in Morris et al. (2019). If the option `mc_se` is set to `TRUE`, estimates of Monte Carlo standard error will be included in the summary data frame, along with associated 95% confidence intervals based on a normal approximation.

```
R> sim <- new_sim()
R> create_data <- function(n) { rpois(n, lambda=5) }
R> est_mean <- function(dat) {
+    return(mean(dat))
+  }
R> sim %<>% set_levels(n=c(10,100,1000))
R> sim %<>% set_config(num_sim=5)
R> sim %<>% set_script(function() {
+    dat <- create_data(L$n)
+    lambda_hat <- est_mean(dat=dat)
+    return (list("lambda_hat"=lambda_hat))
+  })
R> sim %<>% run()
  |########################################| 100%
Done. No errors or warnings detected.
R> sim %>% summarize(
+    list(stat="mse", name="lambda_mse", estimate="lambda_hat", truth=5),
+    mc_se = TRUE
+  )
  level_id    n n_reps lambda_mse lambda_mse_mc_se lambda_mse_mc_ci_l
1        1   10      5  0.5020000      0.274178774      -0.0353903966
2        2  100      5  0.0142800      0.012105759      -0.0094472876
3        3 1000      5  0.0031878      0.001919004      -0.0005734471
  lambda_mse_mc_ci_u
1        1.039390397
2        0.038007288
3        0.006949047
```

6 Discussion

In this article, we described **SimEngine**, an open-source R package for structuring, maintaining, running, and debugging statistical simulations. We reviewed the overall guiding design principles of the package, illustrated its workflow and main functions, highlighted local and CCS-based parallelization capabilities, and demonstrated advanced functionality. It is our hope that this article publicizes the existence of the package and leads to an increase in the size and engagement of the active user community.

SimEngine is one of several R packages aimed at users conducting statistical simulations but is, to our knowledge, the only package explicitly intended for use in a CCS environment. The **simulator** package (Bien, 2016) provides a modular and flexible framework to structure simulations but contains no functionality designed specifically for debugging simulations or leveraging a CCS. The **simpr** package (Brown and Bye, 2023) uses tidy principles but does not emphasize flexibility, with a relatively limited scope. Likewise, **simFrame** (Alfons et al., 2010), an object-oriented simulation package, makes available a suite of relatively specific simulation tools, which are intended primarily for survey statistics. Both **SimDesign** (Chalmers and Adkins, 2020) and **simsalapar** (Hofert and Mächler, 2013) are designed for ease of use but are somewhat less modular than other packages, with a single function for generating simulated data, running analyses, and summarizing results. The recent **simChef** package (Duncan et al., 2024) uses a tidy grammar framework and has additional functionality for preparing reports on the results of a simulation study. Additionally, there are a host of packages available that aid in simulating particular data structures, such as **riskSimul** (Hormann and Basoglu, 2023), **GlmSimulatoR** (McMahan, 2023), and **PhenotypeSimulator** (Meyer and Birney, 2018), which can be used in conjunction with a package for structuring simulations if applicable.

We chose to write this package specifically for the R programming language since this language is widely used by statistical methods developers; as evidence of this claim, a review of the top ten studies from a simple Google Scholar search of the term "simulation study" (restricted to the last five years) shows that, out of the eight studies that stated the programming language used, six were coded using R (Belias et al., 2019; Edelsbrunner et al., 2023; Todorov et al., 2020; Rusticus and Lovato, 2019; Manolov, 2019; Bower et al., 2021; Hamza et al., 2021; Thompson et al., 2021). That being said, it could be of practical value in the future to port the package to other programming environments.

Moving forward, future development of the package will be driven mainly by requests from active users, screening and prioritizing potential additions or modifications based on the design principles articulated in Section 2.2. In particular, we hope to focus on improving parallelization capabilities, including expansion of SimEngine to other parallelization platforms (e.g., Apache Spark), support for futures/promises (as in the **future** package), and further automation of the cluster parallelization process (possibly by having **SimEngine** automatically create and submit `sbatch` commands, as is done in the **rslurm** package). Users can navigate to `https://github.com/Avi-Kenny/SimEngine/issues` to submit feature requests and view current open issues. Additionally, because SimEngine is built using the R S3 class system, it is extensible and allows for users to write additional methods customized to their particular needs, workflow, and style, such as functionality for plotting simulation results.

Computational details

The results in this paper were obtained using R 4.3.2 with the **SimEngine** 1.4.0 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`.

References

A. Alfons, M. Templ, and P. Filzmoser. An object-oriented framework for statistical simulation: The R package simframe. *Journal of Statistical Software*, 37:1–36, 2010. [p218]

B. F. Arnold, D. R. Hogan, J. M. Colford, and A. E. Hubbard. Simulation methods to estimate design power: an overview for applied research. *BMC Medical Research Methodology*, 11(1): 1–10, 2011. [p200]

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2022. URL https://CRAN.R-project.org/package=magrittr. R package version 2.0.3. [p203]

M. Belias, M. M. Rovers, J. B. Reitsma, T. P. Debray, and J. IntHout. Statistical approaches to identify subgroups in meta-analysis of individual participant data: a simulation study. *BMC Medical Research Methodology*, 19:1–13, 2019. [p218]

J. Bien. The simulator: an engine to streamline simulations. *arXiv preprint arXiv:1607.00021*, 2016. [p218]

H. Bower, M. J. Crowther, M. J. Rutherford, T. M.-L. Andersson, M. Clements, X.-R. Liu, P. W. Dickman, and P. C. Lambert. Capturing simple and complex time-dependent effects using flexible parametric survival models: A simulation study. *Communications in Statistics-Simulation and Computation*, 50(11):3777–3793, 2021. [p218]

E. Brown and J. Bye. *simpr: Flexible 'Tidyverse'-Friendly Simulations*, 2023. URL https://CRAN.R-project.org/package=simpr. R package version 0.2.6. [p218]

R. P. Chalmers and M. C. Adkins. Writing effective and reliable monte carlo simulations with the simdesign package. *The Quantitative Methods for Psychology*, 16(4):248–280, 2020. [p218]

J. Duncan, T. Tang, C. F. Elliott, P. Boileau, and B. Yu. simchef: High-quality data science simulations in R. *Journal of Open Source Software*, 9(95):6156, 2024. [p218]

P. A. Edelsbrunner, M. Flaig, and M. Schneider. A simulation study on latent transition analysis for examining profiles and trajectories in education: Recommendations for fit statistics. *Journal of Research on Educational Effectiveness*, 16(2):350–375, 2023. [p218]

T. Hamza, A. Cipriani, T. A. Furukawa, M. Egger, N. Orsini, and G. Salanti. A Bayesian dose–response meta-analysis model: A simulations study and application. *Statistical Methods in Medical Research*, 30(5):1358–1372, 2021. [p218]

W. W. Hauck and S. Anderson. A survey regarding the reporting of simulation studies. *The American Statistician*, 38(3):214–216, 1984. [p200]

M. Hofert and M. Mächler. Parallel and other simulations in R made easy: An end-to-end study. *arXiv preprint arXiv:1309.4402*, 2013. [p218]

W. Hormann and I. Basoglu. *riskSimul: Risk Quantification for Stock Portfolios under the T-Copula Model*, 2023. URL https://CRAN.R-project.org/package=riskSimul. R package version 0.1.2. [p218]

E. Koehler, E. Brown, and S. J.-P. Haneuse. On the assessment of monte carlo error in simulation-based statistical analyses. *The American Statistician*, 63(2):155–162, 2009. [p200]

S. M. Lee and G. A. Young. The effect of monte carlo approximation on coverage error of double-bootstrap confidence intervals. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 61(2):353–366, 1999. [p217]

R. Manolov. A simulation study on two analytical techniques for alternating treatments designs. *Behavior Modification*, 43(4):544–563, 2019. [p218]

G. McMahan. *GlmSimulatoR: Creates Ideal Data for Generalized Linear Models*, 2023. URL https://CRAN.R-project.org/package=GlmSimulatoR. R package version 1.0.0. [p218]

H. V. Meyer and E. Birney. PhenotypeSimulator: A comprehensive framework for simulating multi-trait, multi-locus genotype to phenotype relationships. *Bioinformatics*, 34(17):2951–2956, 2018. [p218]

T. P. Morris, I. R. White, and M. J. Crowther. Using simulation studies to evaluate statistical methods. *Statistics in Medicine*, 38(11):2074–2102, 2019. [p217]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL https://www.R-project.org/. [p200, 206]

S. A. Rusticus and C. Y. Lovato. Impact of sample size and variability on the power and type I error rates of equivalence tests: A simulation study. *Practical Assessment, Research, and Evaluation*, 19(1):11, 2019. [p218]

J. Thompson, K. Hemming, A. Forbes, K. Fielding, and R. Hayes. Comparison of small-sample standard-error corrections for generalised estimating equations in stepped wedge cluster randomised trials with a binary outcome: a simulation study. *Statistical Methods in Medical Research*, 30(2):425–439, 2021. [p218]

H. Todorov, E. Searle-White, and S. Gerber. Applying univariate vs. multivariate statistics to investigate therapeutic efficacy in (pre) clinical trials: A monte carlo simulation study on the example of a controlled preclinical neurotrauma trial. *PLoS One*, 15(3):e0230798, 2020. [p218]

J. Wakefield. *Bayesian and Frequentist Regression Methods*, volume 23. Springer, 2013. [p200]

Avi Kenny
Department of Biostatistics and Bioinformatics, Duke University
Global Health Institute, Duke University
2424 Erwin Rd
Durham, NC 27710, USA
ORCiD: 0000-0002-9465-7307
avi.kenny@duke.edu

Charles J. Wolock
Department of Biostatistics and Computational Biology
University of Rochester
265 Crittenden Boulevard,
Rochester, New York 14642, USA
ORCiD: 0000-0003-3527-1102
c.wolock@rochester.edu

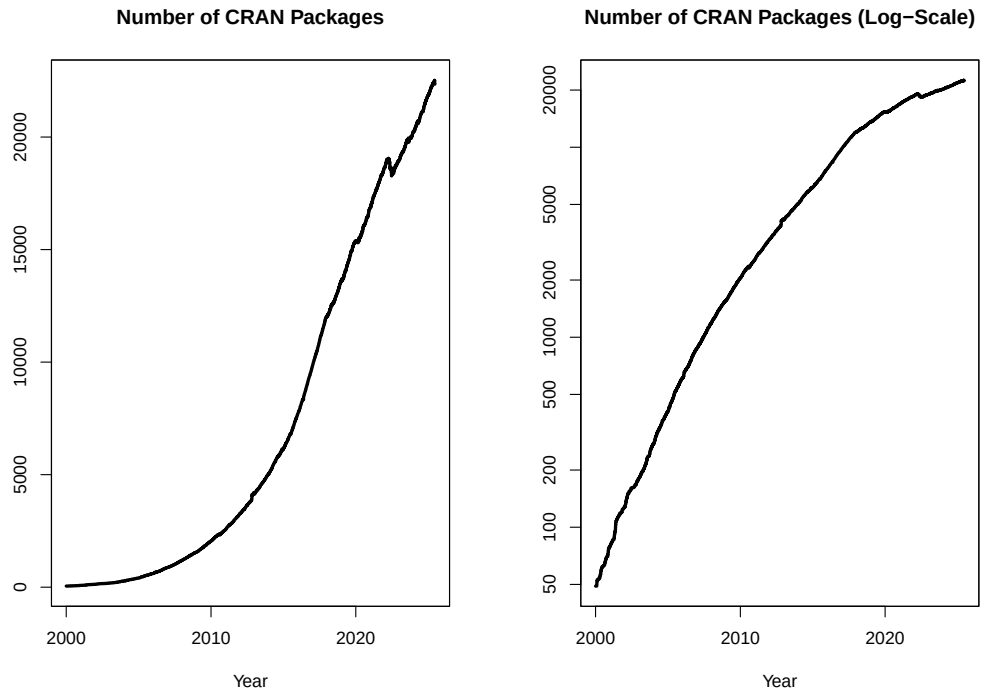
Changes on CRAN

2025-01-01 to 2025-03-31

by Kurt Hornik, Uwe Ligges, and Achim Zeileis

1 CRAN growth

In the past 3 months, 528 new packages were added to the CRAN package repository. 142 packages were unarchived, 274 were archived and 0 had to be removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2025-03-31, the number of active packages was around 22253.

2 CRAN package submissions

From January 2025 to March 2025 CRAN received 7690 package submissions. For these, 12213 actions took place of which 9271 (76%) were auto processed actions and 2942 (24%) manual actions.

Minus some special cases, a summary of the auto-processed and manually triggered actions follows:

	archive	inspect	newbies	pending	pretest	publish	recheck	waiting
auto	2798	623	1622	152	0	2401	931	356
manual	1171	4	13	6	60	1307	276	98

These include the final decisions for the submissions which were

	archive	publish
auto	2672 (35.5%)	2179 (28.9%)
manual	1157 (15.4%)	1522 (20.2%)

where we only count those as *auto* processed whose publication or rejection happened automatically in all steps.

3 CRAN mirror security

Currently, there are 93 official CRAN mirrors, 77 of which provide both secure downloads via '`https`' *and* use secure mirroring from the CRAN master (via rsync through ssh tunnels). Since the R 3.4.0 release, `chooseCRANmirror()` offers these mirrors in preference to the others which are not fully secured (yet).

4 CRAN Task View Initiative

There are two new task views:

- Compositional Data Analysis: Maintained by Karel Hron, Javier Palarea-Albaladejo, Matthias Templ, Alessandra Menafoglio.
- Network Analysis: Maintained by Fabio Ashtar Telarico, Pavel N. Krivitsky, James Hollway.

Currently there are 48 task views (see `https://CRAN.R-project.org/web/views/`), with median and mean numbers of CRAN packages covered 108 and 122, respectively. Overall, these task views cover 4906 CRAN packages, which is about 22% of all active CRAN packages.

Kurt Hornik
WU Wirtschaftsuniversität Wien
Austria
ORCiD: 0000-0003-4198-9911
`Kurt.Hornik@R-project.org`

Uwe Ligges
TU Dortmund
Germany
ORCiD: 0000-0001-5875-6167
`Uwe.Ligges@R-project.org`

Achim Zeileis
Universität Innsbruck
Austria
ORCiD: 0000-0003-0918-3766
`Achim.Zeileis@R-project.org`

R Foundation News

by Torsten Hothorn

1 Donations and members

Membership fees and donations received between 2025-03-26 and 2025-07-09.

2 Supporting institutions

Alfred Mueller Analytic Services, München (Germany), Departement Klinische Forschung, Basel (Switzerland), Ef-prime, Inc., Tokyo (Japan), NIFU Nordic Institute for Studies in Innovation, Research and Education, Oslo (Norway), The University of Auckland, Statistics Department, Auckland (New Zealand).

3 Supporting members

Vedo Alagic (Austria), Jose Alaya (Peru), Kristoffer Winther Balling (Denmark), Ashanka Beligaswatte (Australia), Emmanuel Blondel (France), Tom Boulay (United States), Andreas Büttner (Germany), Robert Carnell (United States), Chao Cheng (China), Giuseppe Corbelli (Italy), Alistair Cullum (United States), Ajit de Silva (United States), Elliott Deal (United States), Dubravko Dolic (Germany), Serban Dragne (United Kingdom), Guenter Faes (Germany), Susan Gruber (United States), Chris Hanretty (United Kingdom), James Harris (United States), Knut Helge Jensen (Norway), Brian Johnson (United States), Christian Kampichler (Netherlands), Katharina Kesy (Germany), Sebastian Koehler (Germany), Sebastian Krantz (Germany), Luca La Rocca (Italy), Teemu Daniel Laajala (Finland), Jindra Lacko (Czechia), Thierry Lecerf (Switzerland), Eric Lim (United Kingdom), Michal Majka (Austria), Ivan Marino (Italy), Harvey Minnigh (Puerto Rico), David Monterde (Spain), Maciej Nasinski (Poland), Mark Niemann-Ross (United States), Jens Oehlschlägel (Germany), Jaesung James Park (Korea, Republic of), Josiah Parry (United States), Bill Pikounis (United States), Dominic Schuhmacher (Germany), Murray Sondergard (Canada), Berthold Stegemann (Germany), Tim Taylor (United Kingdom), Fredrik Wartenberg (Sweden).

Torsten Hothorn
Universität Zürich
Switzerland
ORCiD: 0000-0001-8301-0471
Torsten.Hothorn@R-project.org